## ###### Linux containers and Dockers ######

Docker is to use the phrase from the Docker web site—Docker is “an open source project to pack, ship and run any application as a lightweight container.” The idea is to provide a comprehensive abstraction layer that allows developers to “containerize” or “package” any application and have it run on any infrastructur.

The use of container here refers more to the consistent, standard packaging of applications rather than referring to any underlying technology (a distinction that will be important in a moment). The most common analogy used to help people understand Docker is saying that Docker containers are like shipping containers: they provide a standard, consistent way of shipping just about anything. Docker containers provide a standard, consistent way of packaging just about any application.

## To start with Dockers:

### Installation of Docker  On linux based system:

**Note:**  we are using  Redhat 7.2 / centos 7.2

## A)  Installing docker and its dependencies

```
[root@localhost ~]# yum  install  docker  docker-selinux  -y
```

## Starting and enabling service

```
[root@localhost ~]# systemctl  restart   docker

[root@localhost ~]# systemctl  enable   docker

[root@localhost ~]# systemctl  status   docker


docker.service - Docker Application Container Engine

   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled)

   Active: active (running) since Thu 2016-01-07 22:46:00 IST; 5s ago

     Docs: http://docs.docker.com

 Main PID: 11000 (docker)

   CGroup: /system.slice/docker.service

           └─11000 /usr/bin/docker daemon --selinux-enabled
```

## B) Installation of DOcker  inside microsoft windows

Because Docker relies on Linux-specific features, you can’t run Docker natively in Windows. Instead, you must install the Docker Toolbox application. The application installs a VirtualBox Virtual Machine (VM), Docker itself, and the Docker Toolbox management tool. These three things allow you to run Docker on Windows.

### Step 1: Check your version

Your machine must be running Windows 7.1, 8/8.1 or newer to run Docker Toolbox

Make sure your Windows system supports Hardware Virtualization Technology and that virtualization is enabled.

**Step 2: Install Docker Toolbox**

In this section, you install the Docker Toolbox software and several “helper” applications. The installation adds the following software to your machine:

Docker Client for Windows

Docker Toolbox management tool and ISO

Oracle VM VirtualBox

Git MSYS-git UNIX tools

**Note:** URL of Docker Toolbox is right below

**https://www.docker.com/docker-toolbox**

## iii) Now check for some basic things

**a) Docker version**

```
[root@localhost ~]# docker -v

Docker version 1.8.2-el7.centos, build a01dc02/1.8.2
```

**OR**

```
[root@localhost ~]# docker version

Client:

 Version:      1.8.2-el7.centos

 API version:  1.20

 Package Version: docker-1.8.2-10.el7.centos.x86_64

 Go version:   go1.4.2

 Git commit:   a01dc02/1.8.2

 Built:

 OS/Arch:      linux/amd64
```

```
Server:
 Version:      1.8.2-el7.centos
 API version:  1.20
 Package Version:
 Go version:   go1.4.2
 Git commit:   a01dc02/1.8.2
 Built:
 OS/Arch:      linux/amd64
[root@localhost ~]#
```

**b) Info about kernel version and Storage Drivers:**

```
[root@localhost ~]# docker info
Containers: 0
Images: 4
Storage Driver: devicemapper
 Pool Name: docker-253:1-17112523-pool
 Pool Blocksize: 65.54 kB
 Backing Filesystem: xfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 2.056 GB
 Data Space Total: 107.4 GB
 Data Space Available: 7.88 GB
 Metadata Space Used: 1.729 MB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.146 GB

 Udev Sync Supported: true

 Deferred Removal Enabled: false

 Data loop file: /var/lib/docker/devicemapper/devicemapper/data

 Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata

 Library Version: 1.02.107-RHEL7 (2015-10-14)

Execution Driver: native-0.2

Logging Driver: json-file

Kernel Version: 3.10.0-123.el7.x86_64
```

## iv) Now searching for docker base images on docker hub

For example looking for MongoDB based docker Images:

You can use "Docker Search <imagename>"

```
[root@localhost ~]# docker  search  mongodb

INDEX      NAME                        DESCRIPTION                              STARS    OFFICIAL   AUTOMATED

docker.io   docker.io/tutum/mongodb        MongoDB Docker image – listens in port 2...     86
[OK]

docker.io   docker.io/frodenas/mongodb       A Docker Image for MongoDB                   5
[OK]

docker.io   docker.io/sameersbn/mongodb                                        4                [OK]

docker.io   docker.io/waitingkuo/mongodb     MongoDB 2.4.9                          4               [OK]

docker.io   docker.io/azukiapp/mongodb       Docker image to run MongoDB by Azuki - htt...   3
[OK]
```

## v) Download the images from Docker HUB and check in local system

### a) CHecking for local system : list of available images

```
[root@localhost ~]# docker images

REPOSITORY        TAG            IMAGE ID        CREATED          VIRTUAL SIZE

ubuntu14_04        latest          8251da35e7a7      5 months ago       188.3 MB

[root@localhost ~]#
```

### b) Pulling image from docker hub:

```
[root@localhost ~]# docker pull ubuntu


Using default tag: latest

Trying to pull repository docker.io/library/ubuntu ... latest: Pulling from library/ubuntu


895b070402bd: Pulling fs layer

02e5bca4149b: Pulling fs layer

b2ae0a712b39: Pulling fs layer

af88597ec24b: Pulling fs layer
```

**After downloading  check images again :**

```
[root@localhost ~]# docker images
REPOSITORY         TAG              IMAGE ID         CREATED           VIRTUAL SIZE
docker.io/ubuntu    latest           af88597ec24b       2 days ago        187.9 MB
mongodb_new        latest           dd2527ea18bd       4 days ago        968.2 MB
ubuntu14_04        latest           8251da35e7a7       5 months ago       188.3 MB
[root@localhost ~]#
```

## vi) Running Docker for testing some basic commands

### a)  for checking  date  command  testing

```
[root@localhost ~]# docker run -it  ubuntu14_04  date
Thu Jan  7 18:03:31 UTC 2016
[root@localhost ~]#
```

**Important:**

**Docker syntax:**

docker  run  -i (interactive)  -t (terminal)   ubuntu14_04  (images name)  command [date]

### b)  Running  a docker image with bash shell for holding image

```
[root@localhost ~]# docker run -it  ubuntu14_04  /bin/bash
root@e0e13a40ce3c:/#
```

### c) Checking docker images running

```
[root@localhost ~]# docker ps

CONTAINER ID       IMAGE             COMMAND           CREATED          STATUS            PORTS
NAMES

e0e13a40ce3c       ubuntu14_04       "/bin/bash"       4 minutes ago       Up 4 minutes
loving_ardinghelli
```

## v) save work inside docker images by commiting

When you launch a docker image and start working like you have created some files or make some change in internal os when you exit from docker

it will not be saved when you rerun that images

**For example:**

```
[root@localhost ~]# docker run -it  ubuntu14_04  /bin/bash

bash-4.1# touch  /tmp/hii.txt

bash-4.1# ls /tmp/

hii.txt

bash-4.1#  exit

[root@localhost ~]#
```

**Note:** now run again the same images

```
[root@localhost ~]# docker run -it  ubuntu14_04  /bin/bash

bash-4.1# ls /tmp/
```

Here no content saved

## vi) Now commiting images

**First at running time check container ID:**

```
[root@localhost ~]# docker ps

CONTAINER ID       IMAGE            COMMAND          CREATED          STATUS           PORTS
NAMES

0f1ef17948e0      mongodb_new       "/bin/bash"       12 minutes ago     Up 6 minutes       3000/tcp
```

**OR**

```
[root@localhost ~]# docker ps -a

CONTAINER ID       IMAGE            COMMAND          CREATED          STATUS                   PORTS
NAMES

0f1ef17948e0      mongodb_new       "/bin/bash"       20 minutes ago     Exited (0) 7 minutes ago
condescending_kowalevski

e0e13a40ce3c      ubuntu14_04       "/bin/bash"       8 hours ago       Exited (137) 38 minutes ago
loving_ardinghelli

797c6d796cef      ubuntu14_04       "/bin/bash"       8 hours ago       Exited (0) 8 hours ago
```

**now  you commit with a new name**

```
[root@localhost ~]# docker commit   0f1ef17948e0  ashutoshh/mongonew:v1
```

**Here:**  ashutsohh/mongonew:v1--------------- username/imagename:tag

**Now check by docker images:**

```
[root@localhost ~]# docker images
REPOSITORY         TAG              IMAGE ID         CREATED           VIRTUAL SIZE
ashutoshh/mongonew   v1              f646903bbed1      5 minutes ago      968.2 MB
docker.io/ubuntu     latest           af88597ec24b      3 days ago         187.9 MB
```

## vii)  Now  creating  images from docker images

### a)  Check for images

```
[root@localhost ~]# docker images
REPOSITORY         TAG              IMAGE ID         CREATED           VIRTUAL SIZE
ashutoshh/mongonew   v1              f646903bbed1      20 minutes ago     968.2 MB
docker.io/ubuntu     latest           af88597ec24b      3 days ago         187.9 MB
mongodb_new        latest           dd2527ea18bd      4 days ago         968.2 MB
ubuntu14_04        latest           8251da35e7a7      5 months ago       188.3 MB
```

**b) save image from listed above**

```
[root@localhost ~]# docker  save  -o  /root/myubuntu14.tar  ubuntu14_04
```

**c) check in your base system**

```
[root@localhost ~]# ls  /root
anaconda-ks.cfg  mongodb.tar  myubuntu14.tar
[root@localhost ~]#
```

**Note:** you share this image with others who have installed docker engine or docker plateform in there system they can use this images.

**To use above save image**:

```
[root@localhost ~]# docker  load -i  /root/myubuntu14.tar
```

## viii) Creating Images from Dockerfile.

this is most efficient way of creating image in your local system from DockerHub .

**Advantages:**

a) you can predefine any packages that you want to installed by default in your docker image

b) you also can predefine any command that must run if image is started.

**Precautions:**

**a)** You must create Dockerfile in a newdirectory because it includes the content of your base directory so location like /root and /etc will cause some damage or take long time

**b)** name of Dockerfile must be like this : "Dockerfile"

```
[root@localhost ~]# mkdir /test

[root@localhost ~]# cd  /test

[root@localhost test]# touch Dockerfile
```

**Note:** Dockerfile will look like this

```
#ubuntu based hello world image     # just  a comment

FROM  ubuntu:15.04              #   from what base image you want to build image

MAINTAINER  ashutoshh@linux.com     #   who is the mainter

RUN  apt-get install  nginx  -y     #   package you want to install by default

RUN  apt-get install  apache2  -y

CMD  ["echo","Hello World"]        #   command that you want to run on startup of container
```

# TO RUN Dockerfile

```
[root@localhost ~]#  docker build  -t  dockerubu15:0.1  /test
```

## here  dockerub15:0.1 is the name of docker which will be created

**Important:** some advanced trick in Dockerfile

if want to build a new image from the same Dockerfile then

```
[root@localhost ~]#  docker  build  -t  "testimage1"  /test
```

here i have created i new tag name  "testimage1"

**Note:** Here some Dockerfile related commands you go for as per your requirement

## Dockerfile Commands

ADD

CMD

ENTRYPOINT

ENV

EXPOSE

FROM

MAINTAINER

RUN

USER

VOLUME

WORKDIR

## ix)  The concept  of Docker Volumes

The biggest point of confusion is that Docker filesystems are temporary by default. If you start up a Docker image you'll end up with a container that on the surface behaves much like a virtual machine. You can create, modify, and delete files to your heart's content. But if you stop the container and start it up again, all your changes will be lost: any files you previously deleted will now be back, and any new files or edits you made won't be present.

So Volume in docker images are playing role for making your data persistent and share host machine data inside container

### A)  Launching container with a volume

```
[root@localhost ~]# docker run -it -v  /data  --name=vol3   8251da35e7a7 /bin/bash

root@d87bf9607836:/# cd /data/

root@d87bf9607836:/data# touch abc{1..10}

root@d87bf9607836:/data# ls

abc1  abc10  abc2  abc3  abc4  abc5  abc6  abc7  abc8  abc9
```

### b)  now press [cont +P+Q] to move out from container without terminating the container

checking for container that is running

```
[root@localhost ~]# docker ps

CONTAINER ID       IMAGE             COMMAND           CREATED           STATUS           PORTS
NAMES

d87bf9607836       8251da35e7a7       "/bin/bash"       About a minute ago   Up 31 seconds
vol3

[root@localhost ~]#
```

**c) Fire docker inspect to check out more info about volume**

```
[root@localhost ~]# docker inspect  d87bf9607836


"Mounts": [

      {

          "Name": "cdf78fbf79a7c9363948e133abe4c572734cd788c95d36edea0448094ec9121c",

          "Source":
"/var/lib/docker/volumes/cdf78fbf79a7c9363948e133abe4c572734cd788c95d36edea0448094ec9121
c/_data",

          "Destination": "/data",

          "Driver": "local",

          "Mode": "",

          "RW": true
```

**d) You can attach a running  containers voluem to another containers**

```
[root@localhost ~]# docker run -it  --volumes-from  vol3  8251da35e7a7  /bin/bash


root@ef2f5cc545be:/# ls

bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var

root@ef2f5cc545be:/# ls  /data

abc1  abc10  abc2  abc3  abc4  abc5  abc6  abc7  abc8  abc9

root@ef2f5cc545be:/#
```

### e) you can also mount you base directory inside container

```
[root@localhost ~]# docker run -it  -v  /etc:/etc1 8251da35e7a7 /bin/bash
```

**Here:** /etc is host machine directory and /etc1 is the target inside container

## X) public and Private Registry in Docker :

### Docker have two types of repository :

### a) Public repository

The place from where we have downloaded every base continer images and make commit to push to public repo

### b) private repository

Like public we can also created

## Repository are in two version :

REpo V1....(written in python)

Reop V2....(written in GO)

**Now to create  docker private registry:**

```
root@ashulinux:~# docker  run -d -p 5000:5000  registry
```

**Note:** IT will pull a registry image from public registry

```
root@ashulinux:~# docker  run -d -p 5000:5000  registry
```

**Note:** IT will pull a registry image from public registry

## Important:

**Now go to another  system which is running  docker daemon**

```
[root@localhost docker]# docker images
REPOSITORY              TAG              IMAGE ID          CREATED           VIRTUAL SIZE
ashutoshh/ubuntu14new     04              1e74bbef4d58       9 days ago        188.3 MB
mongodb_new              latest           dd2527ea18bd       2 weeks ago       968.2 MB
ubuntu                14.04           8251da35e7a7       5 months ago       188.3 MB
```

**Now tag any available images with  syntax given below**

```
[root@localhost docker]# docker  tag  8251da35e7a7   ashulinux:5000/myubuntu
```

**Now push this image to Local Docker Hub:**

```
[root@localhost docker]# docker push  ashulinux:5000/myubuntu
```

## Docker Swarm Cluster :

```
# systemctl stop docker

# ps aux | grep docker
```

No docker process run till this time

**Note:** Here we have 4 Redhat 7.1 Machine

**1 node for docker swarm master and 2 for swarm nodes and 1 for client**

**Step 1:** First stop docker on all the 3 nodes except docker client machine and start it in TCP mode like given below

**Master node IP:** 192.168.0.254

**Node 1:** 192.168.0.200

**Node 2:** 192.168.0.201

**Client:** 192.168.0.202

**Start docker engine in tcp mode :**

```
ashutoshh@ashulinux:~$  docker daemon -H  tcp://0.0.0.0:2375  &
```

**Step 2:** Go to client node and connect to docker master node then pull docker swarm image

**Pull swarm images:**

```
# export  DOCKER_HOST=192.168.0.254:2375
```

```
# docker pull swarm
```

**Create swarm cluster**

**Create cluster it or token**

```
ashutoshh@ashulinux:~ docker run swarm create  >  sid.txt
```

share this id to every node

start advertise all of your node

```
# docker run swarm list token://a1b062c17a972e6ad636404bae8e7a5c
```

**Step 3:** Now join the node1 having IP 192.168.0.200

**Note:** First go to client and connect with node1 using above steps export DOCKER_HOST=192.168.0.200:2375

```
# docker run swarm join token://a1b062c17a972e6ad636404bae8e7a5c  --addr 192.168.0.200:2375  &
```

also join from Node2 having IP 192.168.0.201

First connection with node2 from client and then fire this command

```
# docker run swarm join token://a1b062c17a972e6ad636404bae8e7a5c  --addr 192.168.0.201:2375  &
```

**Step 4:** create manager on Docker master node

```
# docker run -p  5001:2375  swarm manage token://a1b062c17a972e6ad636404bae8e7a5c  &
```

**Step 5:** connect thru client and check some basic thing

```
# docker  –H  tcp://192.168.0.200:2375  info

# docker  –H  tcp://192.168.0.201:2375  ps
```