**Docker Course Content**

\*\*\*\*\*\*\*\*\* Miscellaneous commands from class \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Mandatory steps start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Changes instructor to Avinash .

Class start from jun2 nd.

# <https://aws.amazon.com/console/>

1. **Use EC2/ goto dashboard /Key Pair**
2. **Create a key pair and save it on your local drive for as security key us.**
3. **Click on running instances/ Launch instance/Select free teir/ and select**
4. **Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-07c1207a9d40bc3bd (64-bit x86)**
5. **Select and click review and launch**
6. **Use your key pair from local driver to as key**
7. **Once instance is running – click on Connect button and copy ssh command string.**
8. **Now run Mobaxterm: and execute the following commands**
9. **/home/mobaxterm cd desktop/key**
10. **desktop/key chmod 400 MD.pem**
11. **desktop/key ssh -i "MD.pem"** [**ubuntu@ec2-13-59-158-14.us-east-2.compute.amazonaws.com**](mailto:ubuntu@ec2-13-59-158-14.us-east-2.compute.amazonaws.com)
12. **ubuntu@ip-172-31-25-228:~$ sudo su –** ( change to root user )
13. **root@ip-172-31-25-228:~# apt-get update && apt-get install docker.io -y && service docker start**
14. **(yum install docker -y && service docker start ) – for centos.**
15. **docker info -- to make sure docker is running and all the defaults are as expected.**
16. **docker images**
17. **docker ps**
18. **docker ps -a**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Mandatory step end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **docker pull ubuntu**
2. **docker run ubuntu**
3. **docker run -dit –-restart always ubuntudocker run -dit –-restart no redis**
4. **docker run -dit --restart on-failure --name 009 ecf50a2f867d**
5. **docker run -dit --restart on-failure --name 009 nginx**
6. **docker start 349ace8cb00f**
7. **docker kill 349ace8cb00f c482b42f78a5 0aec03019f02**
8. **docker rm -f 689a420d3afa 525cb5cdc27e 525cb5cdc27e ecf50a2f867d 349ace8cb00f**
9. **docker rmi -f apache2-hhtpd-webapp pparihar09/apache2-hhtpd-webapp ubuntu nginx**
10. **cat /var/lib/docker/image/overlay2/repositories.json**
11. **hostname**
12. **hostname -I -o/p is private IP address**
13. **uname -m - gives architecture x86\_64**
14. **uname -s - gives output OS Linux**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**sudo su - --- login as root user to install docker.**

**yum install docker -y --– yum is package manager for centos**

**service docker start**

**docker pull hello-world**

**docker run hello-world**

**docker pull mysql**

**docker run -dit -p 8000:80 -name mydemo mysql:5.7 (8000 incoming port, 80 internal port mapped d- detach,i-interactive, t-tty, 5.7 mysql version, mydemo name of container)**

**docker create ubuntu**

**docker stop 4ae87a137302**

**docker start 4ae87a137302**

**docker pause demo1**

**docker unpause 5cc6972e4f0a**

**docker stop demo**

**docker start demo**

**docker events demo1 --since 10m**

**docker logs -f demo1**

**docker top demo1**

**docker stats demo2**

**docker run --help**

**docker info ( all docker details )**

**Docker volume create my-vol**

**docker volume inspect my-vol**

**docker volume rm -f c7ac70fb20a5cf8f7fa3a0ad24b67baf924db682fb4e2a7619cbc2790303ac68**

**docker run -dit -p 80:80 – mount source=my-vol,destination=my-vol --name myubuntu ubuntu**

**docker exec -it myubuntu bash**

**docker commit ubuntu my-ubuntu-apache2**

**docker history ubuntu**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Version management software:

SCM Admin knowledge (GitHub) CI Admin (Jenkins/ Team city) CD (Ansible / Chef)

Source is maintained in version control sites. Here it is maintained all meta to changes and versions of all users.

Git

SVN ( Subversion )

IBM Rational ClearCase

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ubuntu installation**

1. sudo su – ( need to be root user )

2. apt-get update && apt-get install docker.io -y && service docker start

5. docker pull ubuntu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Volumes:

**Docker volume create my-vol**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6. **docker run -dit -p 80:80 – mount source=my-vol,destination=my-vol --name myubuntu ubuntu**

Notes:

CONTAINER ID IMAGE PORTS NAMES

e265d0616871 ubuntu 0.0.0.0:80->80/tcp myubuntu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7. root@ip-172-31-41-181:~# **docker exec -it myubuntu bash**

( only -it here, and need linux /bin/bash shell to go inside the container, also can use contained id instead of name )

8. root@e265d0616871:/#

This tells that you are in container 265d0616871

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Next install apache2 inside ubuntu

9. root@e265d0616871:/# apt-get update

10. root@e265d0616871:/# apt-get install apache2 -y

11. root@e265d0616871:/# service apache2 start

12. root@e265d0616871:/# mkdir test123

13. root@e265d0616871:/test123# echo "madhu" > md.txt

14. root@e265d0616871:/test123# exit ( to come out of the container )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Make your own DOCKER image

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**15. root@ip-172-31-41-181:~# docker commit ubuntu my-ubuntu-apache2**

sha256:98b23935dcf45f34a3cdf0e9b74ad1118ab385ba2214dca5e92046a4e326285b

root@ip-172-31-41-181:~# docker images

REPOSITORY IMAGE ID SIZE

my-ubuntu-apache2 98b23935dcf4 208MB (my own docker with enhancements )

ubuntu 1d622ef86b13 73.9MB

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* You can run your updated Docker container with new name mywebapp \*\*\*

16. **root@ip-172-31-41-181:~# docker run -dit --name mywebapp my-ubuntu-apache2**

e76a60329dcb891fac7830ef97c03a6d00a8f54edfe3e58beb465dab4ee46c33

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* you can go in the container with exec command \*\*

17. root@ip-172-31-41-181:~# docker exec -it mywebapp bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Restart the apache2 service

root@e76a60329dcb:/#

root@e76a60329dcb:/# cat test123/m\*

root@e76a60329dcb:/# service apache2 status

root@e76a60329dcb:/# service apache2 start ( to start apache2 you need to be in mywebapp container )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# To upload to docker hub

Format : registryName/userid/imageName

ex: Registry: <https://index.docker.io/v1/>

user id : mduggi

image name: mywebapp

18. root@ip-172-31-41-181:~# **docker tag my-ubuntu-apache2 mduggi/my-ubuntu-apache2**

root@ip-172-31-41-181:~# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

my-ubuntu-apache2 latest 98b23935dcf4 23 minutes ago 208MB

mduggi/my-ubuntu-apache2 latest 98b23935dcf4 23 minutes ago 208MB

ubuntu latest 1d622ef86b13 7 days ago 73.9MB

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## \*\* **Login to hub.docker.com from inside the container where the image is sitting \*\***

1. **root@ip-172-31-41-181:~# docker login**
2. **username : mduggi**
3. **password:$F$**
4. **docker push mduggi/my-ubuntu-apache2 ( this has been uploaded successfully )**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## \*\*\* Restart on failure run policy command \*\*

1. docker run -dit --restart on-failure --name 009 nginx

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mount a volume in container, remove a volume and inspect volume

docker run -dit --mount source=my-vol,destination=/my-vol --name my-nginx nginx

**docker volume inspect my-vol**

**docker volume rm -f c7ac70fb20a5cf8f7fa3a0ad24b67baf924db682fb4e2a7619cbc2790303ac68**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Image Layers and Overlay2

overlay2 is a storage driver.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker images as multiple layers inside the final image. You can see them thru like example.

The top layer is R/W container layer..

the changes are saved in this storage layer : /var/lib/docker/image/overlay2

**docker history ubuntu**

Docker Root Dir: /var/lib/docker

Docker image and all its layers are listed in Docker Dir: /var/lib/docker/image/overlay2/repositories.json

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker uses Union file system (UFS) one top another in multiple layers as a union, copy-on-write strategy (COW strategy) where only is installed(copied) from previous layers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Volumes and Bind Mounts

\*\* Volumes commands \*\*

\*\*Volumes first you have to create them using create command and then mount them with run command

Docker default root dir : /var/lib/docker

Volume dir : /var/lib/docker/volume

Volumes driver is local ( found with docker info command )

**docker volume create my-vol**

**docker volume ls**

/var/lib/docker/volumes/my-vol/\_data ( it creates a \_data where data files are stored when created by default )

docker run -dit --mount source=my-vol,destination=/tmp/prashant --name 001 redis

**$docker volume inspect /tmp/Prashant**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# **\*\*\* Bind Mounts are nothing but shared folders \*\***

Create a dir customer

**#mkdir customer**

**#docker run -dit -v /root/customer/:/customer –name mynginx nginx**

( shared folder we use : whereas for mounts we use , as separator of source and destination)

#docker exec -it mynginx bash

#cd customer ( it should be here )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\* DockerFile \*\*

Docker is light weight, with few instructions the application cab be up and running

Tips:

Named as dockerfile no extension:

12 common commands: **FROM,RUN,CMD,ADD,COPY,ENV,MAINTAINER,USER,WORKDIR,VOLUME,EXPOSE,ENTRYPOINT,ARG**

All commands are in caps

First command should be : FROM

There can be only one CMD instruction in a docker file, if there are more than one, only the last CMD will be executed

CMD can be replaced by RUN or ENTRYPOINT

CMD does not create a layer it is just a command execution. It needs arguments to execute. It is a subset of ENTRYPOINT. ( it is outdated but still in use)

ENTRYPOINT is used to start services as soon as you enter the container and initialized. Is independent command, it will execute even if you do not pass arguments. It is superset command to CMD.

ENV : there can be multiple ENV variables.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Welcome to Dockerfile commands

FROM ubuntu ( Base image to start the build )

RUN apt-get update && apt-get -y install appache2

ADD <Sourcefile> <Destination> ex: /var/madhu /var/www/html

CMD apachectl -D FOREGROUND ( There can be only one CMD instruction in a docker file, if there are moret han one, only the last one will be executed )

ENTRYPOINT apachectl -D FOREGROUND

Ex of dockerfile:

**FROM centos**

**MAINTAINER Madhu**

**USER Madhu**

**RUN yum install -y httpd**

**ADD index.html /var/www/html**

**CMD apachectl -D FOREGROUND**

**EXPOSE 8080**

**MAINTAINER Madhu**

**ENV MYHOME httpd\_home**

WORKDIR /var/Madhu

ARG

VOLUME my-vol

**yum install docker -y**

**service docker start**

**docker build -t my-httpd ./** ( current dir . has dockerfile which is default )

**Docker login**

**Docker tag my-httpd mduggi/my-httpd**

**Docker push mduggi/my-httpd**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Common DOCKERFILE useage commands:

vi dockerfile

cat dockerfile

**docker build -t my-httpd .**

docker history my-httpd

75ae4a3560f4 13 minutes ago /bin/sh -c #(nop) ENV MYHOME=httpd\_home

ba72d5441b20 13 minutes ago /bin/sh -c #(nop) MAINTAINER devops@example… 0B

0d62a0cb027e 13 minutes ago /bin/sh -c #(nop) EXPOSE 8000 0B

6b977528c6a4 13 minutes ago /bin/sh -c #(nop) CMD ["/bin/sh" "-c" "apac… 0B

9d9bfeadba2e 13 minutes ago /bin/sh -c #(nop) ADD file:8315e6d6a7c32ef57… 34B

fe66f00bfd1e 13 minutes ago /bin/sh -c yum install -y httpd 42.4MB

docker tag my-httpd pparihar09/my-httpd

docker login

docker push pparihar09/my-httpd

docker search pparihar09/my-httpd

top ( the session live until you click q to reuse it )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JENKINS :

10.Docker Dockerfile – more commands video after 45 minutes.

vi dockerfile

FROM Jenkins/Jenkins:lts

MAINTAINER mduggi

COPY index.html /var/www/html/

USER root

ENV JENKINS\_USER admin

ENV JENKINS\_APP\_VERSION LTS

#ARG Jenkins\_engine 2.22

EXPOSE 8080

WORKDIR /var/Madhu

CMD

ENTRYPOINT

RUN

VOLUME

docker build -t my-jenkins ./

docker run -dit -p 8080:8080 –name Jenkins my-jenkins

docker ps

**( now enable port number 8080 in AWS Security groups so that outside people can access this software**

**Default port number is 22. Now add 8080 and save rule.)**

**Now take the public IP address of host MC (ex: IPv4 Public IP 3.83.137.96 )from AWS and launch a new tab browser 3.83.137.96:8080 )**

Unlock Jenkins

Go back to mobaxterm and go to the container and get inside the container:

$docker exec -it jenkins bash

cd /var/Jenkins\_home/secrets

cat initialAdminPassword

( Copy this password and go back to the browser and paste it )

( now install suggested plugins, Create First admin User username: admin, password : admin, admin, administrator, [madhu@tocuhworld.com](mailto:madhu@tocuhworld.com) and save and continue, it will give )

You an Jenkins URL so that you can pass it on to your team to access it <http://3.83.137.96:8080> , save and finish.)

People : admin

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To build something from other than a docker file here is a command.

**#docker build -t webapp -f /root/demo/mydockerfilename ./**

( -f /root/demo/mydockerfilename has replaced dockerfile here and ./ is used to execute this command from current root path)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# DOCKER NETWORKING : CNM – container Network Model

Goals of Docker Networking : 1.Flexibilty, 2.Scalibilty, 3.User Friendly, 4.Cross Platform, 5. Decentralizes, 6. Support

1. Flexibility – plug/unplug as and when needed. Docker provides flexibility by enabling any number of applications on various platforms to communicate with each other.
2. Scalabilty -- can spin multiple containers , no limitations, No need to work of Maintaining IP addresses etc, IPAM will take care of it. Docker is a fully distributed network, which enables applications to grow and scale individually while ensuring performance.
3. User Friendly: IPAM is very user friendly to maintain. Docker Native is UX and API to expose to services User Friendly. Docker makes it easy to automate the deployment of services used in data to day life.
4. Cross Platform : Any type of container can be run, frontend container like nginx, backend container like oracle,mysql, postgress, redis, middle tier containers. OS based containers can also be created and run . Docker can be easily used in cross-platform which works across various servers with the help of Docker Swarm clusters.
5. Decentralized : no concept of centralized and control, created as and when needed. Highly available, Docker uses a decentralized network, which enables the capability to have the applications spread and highly available. In the event that a container or a host is suddenly missing from your pool of resource, you can either bring up an additional resource or pass over to services that are still available.
6. Support : Dedicated support for EE out of the box, docker engineers support in production with a ticket thru SLAs.

Architecture of CNM:

1. CNM is built of 5 objects : Network Controller, Driver, Network, EndPoint and sandbox.
2. Standardizes the steps to provide networking for containers using multiple network drivers ( Bridge, host, null ).
3. CNM requires distributed key-value pair like console to store the network configuration.
4. CNM has interfaces for Network plugins and IPAM Plugins.
5. IPAM plugin APIs are used for allocate/deallocate container IP addresses. Network plugin APIs are used to create/delete networks and add/remove containers from networks.

Network Drivers:

1. bridge ( docker0) - default,
2. host – connected directly to laptop, used when you do not want to use your default network, used for office or home or private on which your laptop is connected.
3. overlay – Creates an internal private network that spans across all the nodes participating in the swarm cluster. So, Overlay networks facilitate communication between a swarm service and a standalone container, or between 2 standalone containers on different docker deamons.
4. macvlan – Allows you to assign a MAC address to a container, making it appear as a physical device on your network. Then the docker daemon routes traffic to containers by their MAC addresses. MACvlan driver is the best choice when you are expected to be directly connected to the physical network, rather than routed through the docker’s host network stack.
5. null – similarly if you do not want to use docker network ( default bridge network) or host network but still want to configure your network to do some stand alone R&D. we can use null network.

Docker network commands :

#sudo su –

#apt-get install && apt-get install docker.io -y && service start docker

# docker info

#docker images

#docker ps

#docker volume ls

#docker network ls

#docker network inspect bridge ( docker0)

#docker run -dit --name demo nginx ( by default it is on bridge network )

#docker network disconnect bridge demo

#docker network connect bridge demo

#docker network connect host demo ( will not work – only works with bridge)

#docker network connect null demo ( will not work )

Ex: Name": "bridge",

"Id": "715cea05761afe36e96198295a91115eb5a593772c3bbf41e2f1ff242886c369",

"Created": "2020-05-07T19:23:09.845136437Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

"Containers": {},

Options": {

"com.docker.network.bridge.default\_bridge": "true",

"com.docker.network.bridge.enable\_icc": "true",

"com.docker.network.bridge.enable\_ip\_masquerade": "true",

"com.docker.network.bridge.host\_binding\_ipv4": "0.0.0.0",

"com.docker.network.bridge.name": "docker0",

"com.docker.network.driver.mtu": "1500"

**#docker run -dit –name ubuntu ubuntu**

**#docker network inspect bridge**

Result :

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"2c8d89f8fb9c332245964b990e30f033d1c8c8f061797f1c9e3449ee8b0af452": {

"Name": "ubuntu",

"EndpointID": "1cb5a0169dc6477f7a3543929405e12fa7ba1b6a8601677392faab024260dda0",

"MacAddress": "02:42:ac:11:00:02",

"IPv4Address": "172.17.0.2/16",

"IPv6Address": ""

To ping the IP

Go into the container :

**#docker exec -it ubuntu bash**

**#ping 172.17.0.2 -c 5** ( will ping 5 times )

# Network Advanced:

#docker info - ( shows all the networks : bridge ( docker0), host, ipvlan ,macvlan , null , overlay

Creating our own network.

**docker network create IBM --driver bridge** ( IBM is going to be our own network )

**docker network create ORACLE** ( still uses default driver bridge )

**docker network create myhost --driver host** ( This will fail as only one instance of host network is allowed )

**docker network create mynull --driver null** ( will fail as only one instance of null is allowed )

**docker network create juniper –driver juniper** ( if you have installed this special drivers )

**docker network create cisco –driver cisco**

**docker network inspect default** ( It 172.17.0.0/16, gateway 172.17.0.1 , it will have own IP series, IPAM manages it automatically )

**docker network inspect IBM** ( It 172.18.0.0/16, gateway 172.18.0.1 , it will have own IP series, IPAM manages it automatically )

**docker network inspect ORACLE** ( It 172.19.0.0/16, gateway 172.19.0.1 , it will have own IP series, IPAM manages it automatically )

**docker run -dit --name nbridge-1 ngnix** ( automatically bridge is the default network for nbridge-1 instance of nginx )

**docker network inspect bridge** ( this will show that container nbridge-1 is assigned to bridge Network)

**docker run -dit -- network IBM --name IBM-001 busybox** ( container IBM-001 is assigned to network IBM now )

**docker network inspect IBM** ( you can verify it here by seeing IBM-001 in there in containers)

**docker run -dit --name noracle1 --network ORACLE nginx ( noracle1 container is assigned to network ORACLE )**

**docker network connect ORACLE IBM-001 ( now IBM-001 container is also in ORACLE network with a new EndpointID )**

**docker network create GOOGLE --driver bridge --subnet=12.12.0.0/16 --gateway=12.12.0.1 ( You can create an Network with your own IP series as shown )**

**docker network create CISCO --subnet=25.25.0.0/16 --gateway=25.25.0.1 --ip-range=25.25.1.1/16 (own IP range )**

Ex: 1

root@ip-172-31-26-194:~# docker network inspect IBM

[ {

"Name": "IBM",

"Id": "015b5470eaf4924bb36c0798f49cde2915115af4356ace36e5bfe72b07794cd5",

"Created": "2020-05-09T14:29:41.298328664Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [ {

"Subnet": "172.18.0.0/16",

"Gateway": "172.18.0.1"

} ] },

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {},

"Options": {},

"Labels": {}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

root@ip-172-31-26-194:~# docker network create GOOGLE --driver bridge --subnet=12.12.0.0/16 --gateway=12.12.0.1

0265f828b3cdc9d8d52669f4e77e8ffd2453098be3b4265547abb314d505c610

root@ip-172-31-26-194:~# docker run -dit --name google1 --network=GOOGLE nginx

root@ip-172-31-26-194:~# docker network inspect GOOGLE

[

{

"Name": "GOOGLE",

"Id": "0265f828b3cdc9d8d52669f4e77e8ffd2453098be3b4265547abb314d505c610",

"Created": "2020-05-09T16:14:19.019989056Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "12.12.0.0/16",

"Gateway": "12.12.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"18928ee3a04449ea6a62e2a60e035644744519d8bd7cba73ac2ea17f62304eb6": {

"Name": "google1",

"EndpointID": "6c7536510b3504ba8d14860b5c2db69bcef787a0c545e8f221f4b2dfe3151411",

"MacAddress": "02:42:0c:0c:00:02",

"IPv4Address": "12.12.0.2/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {}

}

]

Example 2 ORACLE Network :

root@ip-172-31-26-194:~# docker network inspect ORACLE

[

{

"Name": "ORACLE",

"Id": "d1f6af6fd8b0fd0330819ca49db804dc7835f1c18e8669d0115968de880fc7af",

"Created": "2020-05-09T14:29:55.677906125Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.19.0.0/16", ( it is different ip series )

"Gateway": "172.19.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {},

"Options": {},

"Labels": {}

}

]

Example 3 bridge Network: :

root@ip-172-31-26-194:~# docker network inspect bridge

[

{

"Name": "bridge",

"Id": "c751bed616e023a32e9968c182f4d4e2bd78f23e5f61935f0f814da692c18a7a",

"Created": "2020-05-09T14:27:32.284298331Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {},

"Options": {

"com.docker.network.bridge.default\_bridge": "true",

"com.docker.network.bridge.enable\_icc": "true",

"com.docker.network.bridge.enable\_ip\_masquerade": "true",

"com.docker.network.bridge.host\_binding\_ipv4": "0.0.0.0",

"com.docker.network.bridge.name": "docker0", - Interview question

"com.docker.network.driver.mtu": "1500"

},

"Labels": {}

}

]

root@ip-172-31-26-194:~# docker run -dit --name bridge-1 nginx

root@ip-172-31-26-194:~# docker network inspect bridge

[

{

"Name": "bridge",

"Id": "c751bed616e023a32e9968c182f4d4e2bd78f23e5f61935f0f814da692c18a7a",

"Created": "2020-05-09T14:27:32.284298331Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"94afc0de0b935a0aaa3e512b3c708547a4520aa0045a9193e882d73f8aa7addb": {

"Name": "bridge-1",

"EndpointID": "af5c3e9711d7e8409fe5d1015021ed01f6985a9eca876d4be36ae5311be00a00",

"MacAddress": "02:42:ac:11:00:02",

"IPv4Address": "172.17.0.2/16",

"IPv6Address": ""

}

},

"Options": {

"com.docker.network.bridge.default\_bridge": "true",

"com.docker.network.bridge.enable\_icc": "true",

"com.docker.network.bridge.enable\_ip\_masquerade": "true",

"com.docker.network.bridge.host\_binding\_ipv4": "0.0.0.0",

"com.docker.network.bridge.name": "docker0",

"com.docker.network.driver.mtu": "1500"

},

"Labels": {}

}

]

\*\*\*\*\*\*\*\*\*\*\*\*\*

root@ip-172-31-26-194:~# docker run -dit --network=IBM --name IBM-001 busybox

root@ip-172-31-26-194:~# docker network inspect IBM

[

{

"Name": "IBM",

"Id": "015b5470eaf4924bb36c0798f49cde2915115af4356ace36e5bfe72b07794cd5",

"Created": "2020-05-09T14:29:41.298328664Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.18.0.0/16",

"Gateway": "172.18.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"ce8d5603fd5db3e5524a9c97e44c41370bda6354141aad4a9166e42f0f70a120": {

"Name": "IBM-001",

"EndpointID": "d7b930d085a4007776efc9bcf1b3cf23652096a505a70c56c98c35b85b93d259",

"MacAddress": "02:42:ac:12:00:02",

"IPv4Address": "172.18.0.2/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {}

}

]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

root@ip-172-31-26-194:~# docker run -dit --name oracle1 --network ORACLE nginx

root@ip-172-31-26-194:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

decdeb4af250 nginx "nginx -g 'daemon of…" 4 seconds ago Up 3 seconds 80/tcp oracle1

ce8d5603fd5d busybox "sh" 8 minutes ago Up 8 minutes IBM-001

94afc0de0b93 nginx "nginx -g 'daemon of…" 15 minutes ago Up 15 minutes 80/tcp bridge-1

root@ip-172-31-26-194:~# docker network inspect ORACLE

[

{

"Name": "ORACLE",

"Id": "d1f6af6fd8b0fd0330819ca49db804dc7835f1c18e8669d0115968de880fc7af",

"Created": "2020-05-09T14:29:55.677906125Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.19.0.0/16",

"Gateway": "172.19.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"decdeb4af250477c46b6ff1035d01f20d9514ea9bb697dc514324603559efc0f": {

"Name": "oracle1",

"EndpointID": "ce27db6606246c7317825edfc4bc3361a140b0ee8f41cc042295149d58522ee4",

"MacAddress": "02:42:ac:13:00:02",

"IPv4Address": "172.19.0.2/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {}

}

]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex; IBM-001 container is in IBM Network, you can alose add this container to ORACLE Network.

root@ip-172-31-26-194:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

decdeb4af250 nginx "nginx -g 'daemon of…" 6 minutes ago Up 6 minutes 80/tcp oracle1

ce8d5603fd5d busybox "sh" 14 minutes ago Up 14 minutes IBM-001

94afc0de0b93 nginx "nginx -g 'daemon of…" 22 minutes ago Up 22 minutes 80/tcp bridge-1

root@ip-172-31-26-194:~# docker network connect ORACLE IBM-001 ( This has added container IBM-001 to ORACLE network too. Now ORACLE Network has 2 containers in it with different IP series, oracle1 conatiner has IP : 172.19.0.2 and IBM-001 has IP : 172.19.0.3 )

IBM-001 has 172.18.0.2/16 in IBM Network and in ORACLE it has IP Series : 172.19.0.3 – So IBM-001 container has 2 ENDPOINTS.

root@ip-172-31-26-194:~# docker network inspect ORACLE

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.19.0.0/16",

"Gateway": "172.19.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"ce8d5603fd5db3e5524a9c97e44c41370bda6354141aad4a9166e42f0f70a120": {

"Name": "IBM-001",

"EndpointID": "901559859b0509b2c16a3f8c5746638ac379de42d0e13236167354d4723c3672",

"MacAddress": "02:42:ac:13:00:03",

"IPv4Address": "172.19.0.3/16",

"IPv6Address": ""

},

"decdeb4af250477c46b6ff1035d01f20d9514ea9bb697dc514324603559efc0f": {

"Name": "oracle1",

"EndpointID": "ce27db6606246c7317825edfc4bc3361a140b0ee8f41cc042295149d58522ee4",

"MacAddress": "02:42:ac:13:00:02",

"IPv4Address": "172.19.0.2/16", -

"IPv6Address": ""

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

root@ip-172-31-26-194:~# docker exec -it IBM-001 sh ( you can go inside container IBM-001 and pins both the ips

/ # ping 172.18.0.2 -c 2

PING 172.18.0.2 (172.18.0.2): 56 data bytes

64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.057 ms

64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.068 ms

/ # ping 172.19.0.3 -c 3

PING 172.19.0.3 (172.19.0.3): 56 data bytes

64 bytes from 172.19.0.3: seq=0 ttl=64 time=0.051 ms

64 bytes from 172.19.0.3: seq=1 ttl=64 time=0.072 ms

64 bytes from 172.19.0.3: seq=2 ttl=64 time=0.068 ms

/ # ping IBM-001

PING IBM-001 (172.18.0.2): 56 data bytes

64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.031 ms

64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.068 ms

64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.081 ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DOCKER COMPOSE : 8th may 2020 class 1.

DOCKER-COMPOSE is a additional layer ( Plug in ) on top of Docker Engine. It is a binary file and needs to be downloaded and installed.

DOCKER-COMPOSE.yml you can create and host a complete environment , like development, testing or production environments

Docker-Compose is a tool for defining and running multi-container Docker applications.

You use YAML to configure your application services.

With a single command you create and start all your services from your configuration.

Multiple isolated environments on a single host.

Preserve volume data when containers are created.

Define app’s environment with Dockerfile so it can be reproduced anywhere.

Docker-compose service can have only one image in each service

Docker cluster environment.

Docker understand YAML, it is a key value pair program language

YAML is going to be the standard , it is also the GO ( python framework with json format ) Language of GOOGLE.

Now going forward we do not need Jenkins to setup CI/CD pipeline, it is past technology. We can use YAML instead of it.

GIT for version control where the source code is stored here and use YAML BUILDs, for deployment environment and also the production environment..

YAML (Yet another markup language/yet ain’t markup language ) are template files. Is also called a config or recipe file. All services can be created thru this including calling docker files.

YAML helps breakdown a monolithic application into small services. Each service will have a small function to perform. Also can call dockerfile in YAML.

Dockerfile -- is used to customize and create your own image

Curl – (Client URL) use it to make http or restful call from a command line. ( upload, download, publish files, apps etc)

Automated test environments: create and destroy your testing environments for your test suite with few commands.

$docker-compose up -d

$./run\_tests

$docker-compose down

Installation Steps: --- In video from 1:02 Minutes to follow installations steps

$yum install docker -y or ( apt-get update && apt-get install docker.io -y ) ------ ( install docker first )

$service docker start ( start docker service next only then you can install docker compose )

$sudo curl -L [https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)](https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname%20-s)-$(uname%20-m)) -o /usr/local/bin/docker-compose ( install compose )

ls -l /usr/local/bin/docker\* -- to check if docker-compose has been installed

$sudo chmod +x /usr/local/bin/docker-compose ----- ( apply executable permissions to the binary file, sudo is optional )

$sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose -- create symbolic link to /usr/bin or another dir in your path

$docker-compose --version -- test the installation ( Now docker-compose is now installed.)

docker-compose up -d --- to check if it is up and running – it will need a docker-compse.yaml file

vi docker-compose.yml --- ( Create yaml file )

docker-compose config --- validate the file for snytax errors

$docker-compose up -d ( you need docker-compose.yaml or docker-compose.yml file to bring this up else it fails )

YAML file main root level properties

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker-compose-yml ( all keys are in small characters – no caps ) – cannot have tabs or any special characters. Use only spaces and always align to parent with 2 spaces.

version: “3.8” ( Version is the docker-compose file format which is currently at “3.8” for Docker Engine release 19.03.0+, always use :, 1 space between Key- version: and value – “3.8”)

services: ( services: This is plural, they are root level properties. )

web-app: --- ( child should have only spaces 2 spaces practice, can have multiple servcies)

Image: nginx --- ( - each service can have only one image)

container\_name: demo

my-db --- ( each service can have only one image )

Image: mysql

container\_name: mydb

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

docker-compose up -d -- ( it creates a default network root\_default )

docker-compose ps -a

docker-compose ps --services ( gives the service name )

docker network ls

docker-compose stop

docker-compose start ( also supports all pause, unpause etc )

docker-compose exec web bash ( \* IMPORTANT, docker-compose recognizes only service not container\_name as in docker command )

docker-compose ps --services

docker-compose logs mynginx

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker compose class 2 : 12052020 –

vi docker-compose-yml FILE FORMAT - ( all lower cases ) child properties should be in a vertical line spaces should align with children.)

version: “3.8” ( no spaces, services: )

services: ( no spaces, services: )

web: ( 2 spaces, web: )

image: nginx ( 4 spaces, image:, 1 space, nginx )

container\_name: my-nginx ( 4 spaces, container\_name:, 1 space, my-nginx )

ports: (4 spaces, ports: )

* 80 ( 6 spaces , -, 1 space, 80 )
* 8080 ( 6 spaces, -, 1 space, 8080)

networks:

* Ibm

networks: ( no spaces, networks: )

ibm:

driver: bridge ( 4 spaces, driver:, 1 space, bridge )

ORACLE:

driver: bridge

from mobaxtrem: docker-compose.yaml file

version: "3.8"

services:

web:

image: nginx

container\_name: mynginx

ports:

- 80

networks:

- ibm

mydb:

image: redis

container\_name: myredis

ports:

- 9000:9000

networks:

- ORACLE

networks:

ibm:

driver: bridge

ORACLE:

driver: bridge

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DOCKER SWARM: 5/14/2020

Why swarm: scalability,

fault tolerance – ( for disaster recorvery )

simplicity and automation.

5000 m/c can be connected in a cluster. 100 containers in a single M/C

Scale up/down – traffic is increasing/decreasing – increase/decrease infrastructure

Default port is 2377 for docker swarm. You need to activate it when you are spinning up the nodes. Source is 0.0.0.0/0. You need to configure this is in step 6. Configure Security Group of AWS and then do a review and launch.

Docker raft can be accessed by manager nodes only. Worker nodes cannot access it.

Manager Node Responsibilities:

API: takes the command ( docker service create) f-rom CLI- and create a service object . all services will be created into tasks.

Once the instruction is put in the API service will pass on the command to docker servr which also called the docker daemon it creates a service object. Objects of service : network, port name, volume

Orchestrator: reconciliation loop for service objects which waits instructions to create tasks and creates the task.

Allocator: IP address is allocated by allocator. So a worker node is assigned to a task.

Scheduler: Assigns nodes to a task, it pushes the tasks to the nodes. Gives the task one by one to the dispatcher with the ip address.

Dispacther: Dispatcher will assign the task to the work node and asks it to perform the task. Once the tasks is done it updates the status back to the manager node.

RAFT database collects all this information and stores it.

So if the active manager node dies for some reason. The next manager node is activated and it gets all the information from the RAFT database as it has all the information and status that the previous Manager node was handling.

* Ensure only trusted servers run our containers - security from hackers. Most tools have inbuilt inbuilt security layer and are hardened.
* Store secrets, keys, passwords and get them to the right container. (usernames, tokens, ssh keys, encrypt and put in a inbuilt database)

Node is a M/c .

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DOCKER SWARM INIT

1.Spin up three instances in AWS. Name instances as MANAGER1, WORKER1,WORKER2.

At step 6. Configure Security Group: Assign port 2377 and 0.0.0.0/0. THIS IS A INTERVIEW QUESTION.

Default port is 2377 for docker swarm. You need to activate it when you are spinning up the nodes. Source is 0.0.0.0/0. You need to configure this is in step 6. Configure Security Group of AWS and then do a review and launch.

2.Have 3 mobaxterm terminals.

3.Run chmod , ssh, apt-get and start docker service or ( yum install docker -y for centos ), docker info --- on all three terminals.

4.Docker info will say docker swarm in inactive.

First terminal execute the following command ( this will automatically become the manager. It has to be run only on node that has to become a manager )

**docker swarm init:** -- Once it is executed , it creates a security layer. Certificate is created and issued to the first manager node. And then join tokens for workers syntax is created which we should copy and save it and use it to assign nodes as worker nodes to this Mangere node.

Raft Database will be created automatically, it is an embedded database: it will store root Certificate, configs and secrets. No need to for another key/value system to hold orchestration/secrets

Replication is done thru TLS, logs, etc.

docker node ls – This command will work in only swarm Manager node and not in Worker node.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Installation steps. 5.16.2020

Docker swarm commands enabled after INIT.

docker swarm

docker node

docker service

docker stack

docker secret

CLI: Initiates:

Manager Node : API, Orchestrator, Allocator, Scheduler, Dispatcher, embedded RAFT Database.

Worker Node : Worker, Executor

Start AWS/EC2/Launch/free/Ubuntu/instances - 6/Configure Security Group – Port 2377 and 0.0.0.0/0/review and launch

mobaxterm

cd 400 MD.pem

ssh

sudo su –

apt-get update && apt-get install docker.io -y

service docker start

docker swarm init

**Docker info :**

Swarm: active

**NodeID**: or0j4x891t3h4ga2tddezl2br

Is Manager: true

**ClusterID**: 03lgsfdnqbuval250x5h2uzxl

Managers: 1

Nodes: 1

Default Address Pool: 10.0.0.0/8

SubnetSize: 24

Data Path Port: 4789

**Orchestration**:

Task History Retention Limit: 5

**Raft:**

Snapshot Interval: 10000

Number of Old Snapshots to Retain: 0

Heartbeat Tick: 1

Election Tick: 10

**Dispatcher**:

--More--WARNING: No swap limit support

Heartbeat Period: 5 seconds

**CA Configuration**:

Expiry Duration: 3 months

Force Rotate: 0

Autolock Managers: false

Root Rotation In Progress: false

Node Address: 172.31.27.112

Manager Addresses:

**172.31.27.112:2377**

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version:

runc version:

init version:

Security Options:

apparmor

seccomp

Profile: default

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

docker node ls – will work only in swarm Manager Node

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

or0j4x891t3h4ga2tddezl2br \* ip-172-31-27-112 Ready Active Leader 19.03.6

uhdyy6e54l2t9n9vwscz3k9tv ip-172-31-29-20 Ready Active 19.03.6

\*\*\*\*\*\*

Docker service ls

docker service create --name md-service nginx -- create a service

docker service ps md-service

Docker service ls

docker service rm md-service - remove a service

docker service create --name md-service -p 80:80 -- replicas=3 nginx - creating 3 replicas of nginx

docker service ps md-service

c1qaw12zmwbc md-service.1 nginx:latest ip-172-31-27-36 Running Running 5 minutes ago

achxxjf3injo md-service.2 nginx:latest ip-172-31-27-112 Running Running 5 minutes ago

bbpy89re8ap3 md-service.3 nginx:latest ip-172-31-29-20 Running Running 5 minutes ago

**docker service scale md-service=5 -- scaling up from 3 to 5 instances.**

c1qaw12zmwbc md-service.1 nginx:latest ip-172-31-27-36 Running Running 15 minutes ago

achxxjf3injo md-service.2 nginx:latest ip-172-31-27-112 Running Running 15 minutes ago

bbpy89re8ap3 md-service.3 nginx:latest ip-172-31-29-20 Running Running 15 minutes ago

rt03j2gh8y9o md-service.4 nginx:latest ip-172-31-22-216 Running Running about a minute ago

yygddlsawgzx md-service.5 nginx:latest ip-172-31-22-216 Running Running about a minute ago

ri4jp8q1widz md-service.6 nginx:latest ip-172-31-29-20 Running Running 19 seconds ago

t7p0ewcw1guq md-service.7 nginx:latest ip-172-31-27-112 Running Running 19 seconds ago

qfidxf1xa009 md-service.8 nginx:latest ip-172-31-27-36 Running Running 19 seconds ago

Add port number 80 AWS/security groups to the manager node and other worker nodes so that they can be accessed from outside

You can test it internally using the following command:

curl <http://3.19.238.32/>

On the web browser using the following command:

3.19.238.32:80

**docker service scale md-service=2** - now the instances have been reduced to 2 with scale command.

c1qaw12zmwbc md-service.1 nginx:latest ip-172-31-27-36 Running Running 34 minutes ago

achxxjf3injo md-service.2 nginx:latest ip-172-31-27-112 Running Running 34 minutes ago

**docker service update --image nginx:1.16 md-service ( update nginx without bringing the service down )**

**docker service ls**

lxtxxyv84xu1 md-service replicated 5/5 nginx:1.16 \*:80->80/tcp

**docker service rollback md-service: ( to rollback upgrade to previous version )**

**docker service ps my-db md-service | grep Run**

**docker node update --availability drain uhdyy6e54l2t9n9vwscz3k9tv ( to bring down a node for maintenance )**

**docker node ls**

xkq0cnnmneriu5vry81kw88by ip-172-31-22-216 Ready Active 19.03.6

rycjjn15nbcg26fch8a1bouf6 ip-172-31-27-36 Ready Active 19.03.6

or0j4x891t3h4ga2tddezl2br \* ip-172-31-27-112 Ready Active Leader 19.03.6

uhdyy6e54l2t9n9vwscz3k9tv ip-172-31-29-20 Ready Drain 19.03.6

**docker swarm leave** ( will make the node leave the swarm )

xkq0cnnmneriu5vry81kw88by ip-172-31-22-216 Ready Active 19.03.6

rycjjn15nbcg26fch8a1bouf6 ip-172-31-27-36 Ready Active 19.03.6

or0j4x891t3h4ga2tddezl2br \* ip-172-31-27-112 Ready Active Leader 19.03.6

uhdyy6e54l2t9n9vwscz3k9tv ip-172-31-29-20 **Down** Drain 19.03.6

To join back..

**docker swarm join --token SWMTKN-1-34vtewu2dtrzu5pulo2ds9dbzc60ipp02qedq7dhogynu1eswm-1ho48qosq38o19r11qli27rxg 172.31.27.112:2377**

**Docker stack ( using yaml file )**

**Docker stack deploy –compose-file docker-compose.yaml my-stack ( will read from docker-compose.yaml file and creates a stack my-stack )**

**Output :**

**Ignoring deprecated options:**

**container\_name: Setting the container name is not supported.**

**Creating network my-stack\_ibm**

**Creating network my-stack\_ORACLE**

**Creating service my-stack\_web**

**failed to create service my-stack\_web: Error response from daemon: The network my-stack\_ibm cannot be used with services. Only networks scoped to the swarm can be used, such as those created with the overlay driver.**

**Used the following yaml file:**

**version: "3.8"**

**services:**

**web:**

**image: nginx**

**container\_name: mynginx**

**ports:**

**- 80**

**networks:**

**- ibm**

**mydb:**

**image: redis**

**container\_name: myredis**

**ports:**

**- 9000:9000**

**networks:**

**- ORACLE**

**networks:**

**ibm:**

**driver: bridge**

**ORACLE:**

**driver: bridge**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**docker stack services my-stack**

**3fqmyyx8idx7 my-stack\_mydb replicated 1/1 redis:latest \*:9000->9000/tcp**

**pze00r48m65i my-stack\_web replicated 1/1 nginx:latest \*:30000->80/tcp**

**docker stack ps my-stack**

**rzjwzcdsgipm my-stack\_mydb.1 redis:latest ip-172-31-29-20 Running Running 3 minutes ago**

**xac1fwpre0wt my-stack\_web.1 nginx:latest ip-172-31-29-20 Running Running 3 minutes ago**