1. INTRODUCTION

1.1 Docker Introduction:-

Docker is an open platform for developers and sys-admins to build their app using container in any language and any tool-chain, ship this app and dependencies anywhere faster and without breaking anything, and run distributed applications.

Docker is written in go. Docker is consists of two things docker engine and docker hub. The Docker engine is portable, lightweight runtime packaging tool, and docker hub is SaaS service for sharing and managing applications.

In the cloud application for isolation and resource control the Virtual Machine is generally used, but the performance of deploying application in VM is not so good because it have one more abstract layer that is of the guest operating system. To achieve that container based virtualization techniques used.

Virtualization, basically virtualization started from logically dividing the resources of the mainframe computers for the applications. The virtualization mainly used is to centralize the administrative task with improving scalability and h/w resource utilization. Creating virtual machine with an operating system that provides the environment like a real system is basically known as hardware virtualization or platform virtualization. In hardware virtualization "Host machine" is the actual machine on which virtualization takes place and "Guest Machine" is the virtual machine. The software or firmware that creates the virtual machine on host hardware is "hypervisor or virtual machine manager".

1.2 Virtual machine V/S Docker:-

In the VM the virtualized application includes the application and necessary the bin/libs that is only 10s of MB and also includes the entire guest operating system which is 10s of GB. But in the docker it just needs the apps, necessary bin/libs and docker engine. Therefore the docker is generally faster from VM.

Docker is built on top of LXC (Linux container). Like with any container technology, as far as the program is concerned, it has its own file system, storage, CPU, RAM, and so on. The key difference between containers and VMs is that while the hypervisor abstracts an entire device, containers just abstract the operating system kernel.

The one thing hypervisors can do that containers can't is to use different operating systems or kernels. So, for example, you can use VM to run both instances of Windows Server 2012 and SUSE Linux Enterprise Server, at the same time. With Docker, all containers must use the same operating system and kernel.

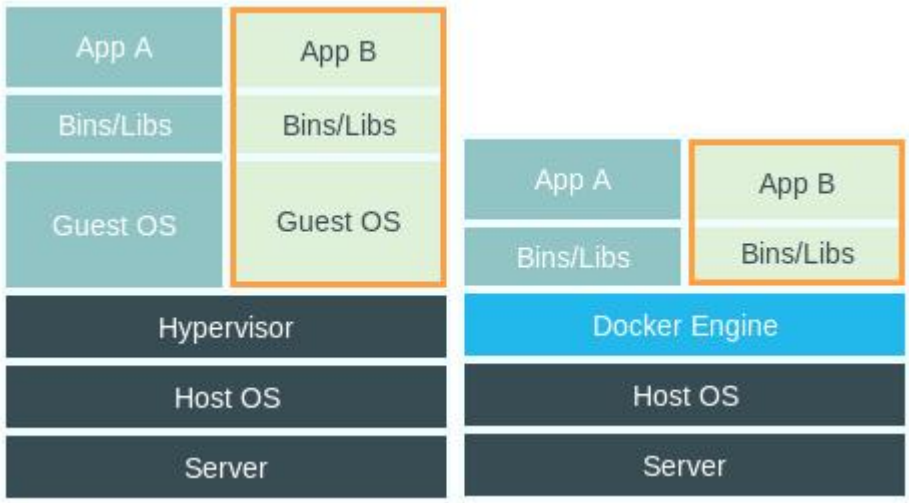

Fig.1.1 : Virtual Machine V/S Docker

1. Docker containers are lightweight therefore faster delivery of applications.

Docker provides isolation and resource control using the lightweight container virtualization platform. The docker runs almost every application which is securely isolated in containers. The containers are lightweight because it runs without the extra load of a hypervisor.

2. Docker almost run everywhere so basically it can run on desktop, server, cloud etc. So it's easily to move app from one environment to another. So it's easy to scale the app also.

3. Docker containers don't need a hypervisor, so can pack more containers on the hosts. So it gives higher density and more workloads.

How the isolation done?

Isolation is done using the namespace and cgroups. In the Isolation each containers has given namespace and it does not access outside of it and in the cgroups or control groups. Control groups allow docker to share available h/w resources but with some constraints and limits.

Docker uses the Union file systems. UnionFS operate by creating layers and making them very lightweight and fast. In the docker UnionFS If its change something and want to save that then it just create one read-write layer on the top of the image.

1.3 Docker architecture:-

Docker uses client-server architecture. The docker talks to the daemons for the tasks like building, running and distributing docker containers. The docker client and server can run on the same system or they can be remote. The docker client and daemon talk via socket or through the RESTful API.

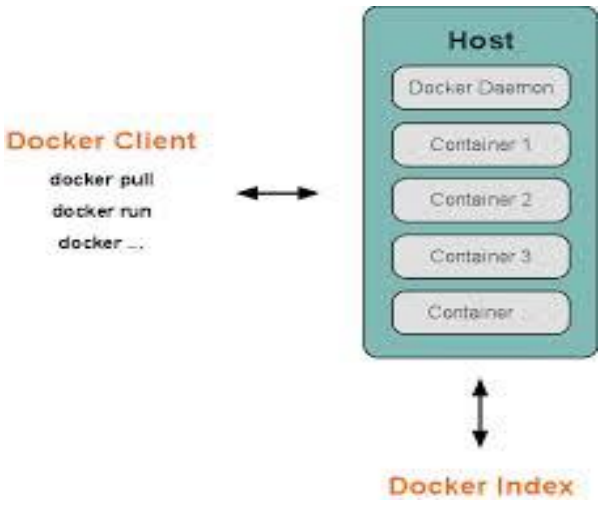

Fig 1.2: Docker Architecture

The Docker daemon runs on host machine and the user does not directly communicate with the daemon but through the docker client. The docker client is in the form of the dockrer binary and it's the primary user interface to docker. This accepts the commands from the user and makes the communication with the docker daemon.

1.4 Docker Components:-

The docker consists of three components, docker images, Registeries, and containers.

Images:-

In Docker terminology, a read-only Layer is called an image. An image never changes. Since Docker uses a Union File System, the processes think the whole file system is mounted read-write. But all the changes go to the top-most writeable layer, and underneath, the original file in the read-only image is unchanged. Since images don't change, images do not have state.

Parent Image:-

Each image may depend on one more image which forms the layer beneath it. We sometimes say that the lower image is the parent of the upper image.

Base Image: - An image that has no parent is a base image.

Image IDs: - All images are identified by a 64 hexadecimal digit string (internally a 256bit value). To simplify their use, a short ID of the first 12 characters can be used on the command line. There is a small possibility of short id collisions, so the docker server will always return the long ID.

Layers:-

In a traditional Linux boot, the kernel first mounts the root File System as read-only, checks its integrity, and then switches the whole rootfs volume to read-write mode.

Layer:-

When Docker mounts the rootfs, it starts read-only, as in a traditional Linux boot, but then, instead of changing the file system to read-write mode. It takes advantage of a union mount to add a read-write file system over the read-only file system. In fact there may be multiple read-only file systems stacked on top of each other. We think of each one of these file systems as a layer.

At first, the top read-write layer has nothing in it, but any time a process creates a file, this happens in the top layer. And if something needs to update an existing file in a lower layer, then the file gets copied to the upper layer and changes go into the copy. The version of the file on the lower layer cannot be seen by the applications anymore, but it is there, unchanged.

Union File System:-

We call the union of the read-write layer and all the read-only layers a union file system.

Container: - Once you start a process in Docker from an Image, Docker fetches the image and it's Parent Image, and repeats the process until it reaches the Base Image. Then the Union File System adds a read-write layer on top. That read-write layer, plus the information about its Parent Image and some additional information like its unique id, networking configuration, and resource limits is called a container.

Container State:-

Containers can change, and so they have state. A container may be running or exited.

When a container is running, the idea of a "container" also includes a tree of processes running on the CPU, isolated from the other processes running on the host.

When the container is exited, the state of the file system and its exit value is preserved. You can start, stop, and restart a container. The processes restart from scratch (their memory state is not preserved in a container), but the file system is just as it was when the container was stopped.

You can promote a container to an Image with docker commit. Once a container is an image, you can use it as a parent for new containers.

Container IDs

All containers are identified by a 64 hexadecimal digit string (internally a 256bit value). To simplify their use, a short ID of the first 12 characters can be used on the command line. There is a small possibility of short id collisions, so the docker server will always return the long ID.

2. DOCKER SETUP ON OPEN SUSE

Install the Docker package:-

$ sudo zypper in docker

Now that it's installed, let's start the Docker daemon.

$ sudo systemctl start docker

If we want Docker to start at boot, we should also:

$ sudo systemctl enable docker

The docker package creates a new group named docker. Users, other than root user, need to be part of this group in order to interact with the Docker daemon. You can add users with:

$ sudo /usr/sbin/usermod -a -G docker <username>

Proxy Settings:

If you are behind a HTTP proxy server, for example in corporate settings, you will need to add this configuration in the Docker systemd service file.

First, create a systemd drop-in directory for the docker service:

mkdir /etc/systemd/system/docker.service.d

Now create a file called /etc/systemd/system/docker.service.d/http-proxy.conf that adds the HTTP_PROXY environment variable:

```
[Service]
Environment="HTTP_PROXY=http://<username>:<password>@<http_proxy>:<port>/"
"HTTPS_PROXY=https://<username>:<password>@<https_proxy>:<port>/"
"NO_PROXY=localhost,127.0.0.0/8”
```

Changing the root storage path:

By default, the docker images will be stores in **/var/lib/docker**. However, if '/' has limited disk space, the docker images can be stored at any other location that has sufficient disk space. e.g: mkdir /home/docker

$ vi /etc/sysconfig/docker

Add the line:

DOCKER_OPTS="-g /home/docker"

Flush changes:

$ sudo systemctl daemon-reload

Restart Docker:

$ sudo systemctl restart docker

Note: If you want your containers to be able to access the external network you must enable the net.ipv4.ip_forward rule. This can be done using YaST by browsing to the Network Devices -> Network Settings -> Routing menu and ensuring that the Enable IPv4 Forwarding box is checked.

This option cannot be changed when networking is handled by the Network Manager. In such cases the /etc/sysconfig/SuSEfirewall2 file needs to be edited by hand to ensure the FW_ROUTE flag is set to yes like so:

FW_ROUTE="yes"

3. CASSANDRA SETUP ON DOCKER

Cassandra:-

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous master less replication allowing low latency operations for all clients.

Cassandra's data model is a partitioned row store with tunable consistency. Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns may be indexed separately from the primary key.

Tables may be created, dropped, and altered at runtime without blocking updates and queries. Cassandra does not support joins or sub-queries. Rather, Cassandra emphasizes de-normalization through features like collections.

3.1 File Structure of Setup:-

The folder Cassandra_docker is provided which has the following structure:

```
cassandra_docker
|
|--  Script_cassandra
|--  Script_cassandra_dataVolume
|
|-- docker-cassandra-master
|   |-- Dockerfile
|   |--  agent-run
|   |--  cassandra-run
|
|-- docker-opscenter-master
|   |-- Dockerfile
|   |-- run
```

3.2 Dockerfiles:-

Dockerfile for building Cassandra image

```
FROM saltstack/opensuse-13.2

MAINTAINER Rakesh Rajpurohit <rajpurohitrakesh@hotmail.com>


ENV http_proxy=http://<username>:<password>@<IP>:<port>

ENV https_proxy=http://<username>:<password>@<IP>:<port>


RUN zypper rr opensuse-network-utilities-x86_64

RUN zypper addrepo -f

http://download.opensuse.org/repositories/network:/utilities/openSUSE_13.2/ opensuse-

network-utilities-x86_64

RUN zypper --non-interactive in wget

RUN zypper --non-interactive in tar


WORKDIR /opt/


RUN wget --no-cookies --no-check-certificate --header "Cookie:

gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-

cookie" "http://download.oracle.com/otn-pub/java/jdk/8u31-b13/jdk-8u31-linux-

x64.tar.gz"

RUN tar xzf jdk-8u31-linux-x64.tar.gz
```

```
WORKDIR /opt/jdk1.8.0_31/

RUN update-alternatives --install "/usr/bin/java" "java" "/opt/jdk1.8.0_31/bin/java" 2
RUN update-alternatives --config java

WORKDIR /opt/

# Download and extract Cassandra
RUN \
  mkdir /opt/cassandra; \
  wget -O - http://www.us.apache.org/dist/cassandra/2.1.2/apache-cassandra-2.1.2-
bin.tar.gz \
  | tar xzf - --strip-components=1 -C "/opt/cassandra";

# Download and extract DataStax OpsCenter Agent
RUN \
  mkdir /opt/agent; \
  wget -O - http://downloads.datastax.com/community/datastax-agent-5.0.1.tar.gz \
  | tar xzf - --strip-components=1 -C "/opt/agent";

ADD    . /src
```

```
# Copy over daemons

RUN    \

    mkdir -p /etc/service/cassandra; \

    cp /src/cassandra-run /etc/service/cassandra/run; \

    mkdir -p /etc/service/agent; \

    cp /src/agent-run /etc/service/agent/run


# Expose ports

EXPOSE 7199 7000 7001 9160 9042


WORKDIR /opt/cassandra


CMD ["/sbin/init"]
```

Dockerfile for building OpsCentre image

```
FROM saltstack/opensuse-13.2

MAINTAINER Rakesh Rajpurohit <rajpurohitrakesh@hotmail.com>


ENV http_proxy=http://<username>:<password>@<IP>:<port>

ENV https_proxy=http://<username>:<password>@<IP>:<port>
```

```
RUN zypper rr opensuse-network-utilities-x86_64

RUN zypper addrepo -f

http://download.opensuse.org/repositories/network:/utilities/openSUSE_13.2/ opensuse-

network-utilities-x86_64

RUN zypper --non-interactive in wget

RUN zypper --non-interactive in tar


WORKDIR /opt/


RUN wget --no-cookies --no-check-certificate --header "Cookie:

gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-

cookie" "http://download.oracle.com/otn-pub/java/jdk/8u31-b13/jdk-8u31-linux-

x64.tar.gz"

RUN tar xzf jdk-8u31-linux-x64.tar.gz


WORKDIR /opt/jdk1.8.0_31/


RUN update-alternatives --install "/usr/bin/java" "java" "/opt/jdk1.8.0_31/bin/java" 2

RUN update-alternatives --config java


WORKDIR /opt/


# Download and extract OpsCenter
```

```
RUN \
  mkdir /opt/opscenter; \
  wget -O - http://downloads.datastax.com/community/opscenter-5.0.1.tar.gz \
  | tar xzf - --strip-components=1 -C "/opt/opscenter";

ADD    . /src

# Copy over daemons
RUN    \
  mkdir -p /etc/service/opscenter; \
  cp /src/run /etc/service/opscenter/;

# Expose ports
EXPOSE 8888

WORKDIR /opt/opscenter

CMD ["/sbin/init"]

RUN bash /etc/service/opscenter/run
```

Script for automated building of a 5-node Cassandra Cluster with Opscenter

```
#!/bin/bash

echo "Starting OpsCenter"

docker run -d -P --name opscenter opensuse-13.2_ops:latest

sleep 10


docker exec opscenter /etc/service/opscenter/run

sleep 30


OPS_IP=$(docker inspect -f '{{ .NetworkSettings.IPAddress }}' opscenter)


echo "Starting node cass_new1"

docker run -d -P --name cass_new1 -e OPS_IP=$OPS_IP opensuse-13.2_cass:latest

sleep 10


docker exec cass_new1 /etc/service/cassandra/run

sleep 10


docker exec cass_new1 /etc/service/agent/run

sleep 30


SEED_IP=$(docker inspect -f '{{ .NetworkSettings.IPAddress }}' cass_new1)

for name in cass_new{2..5}; do
```

```
  echo "Starting node $name"
  docker run -d -P --name $name -e SEED=$SEED_IP -e OPS_IP=$OPS_IP opensuse-
13.2_cass:latest
  sleep 10
  docker exec $name /etc/service/cassandra/run
  sleep 10
  docker exec $name /etc/service/agent/run
  sleep 30
done


echo "Registering cluster with OpsCenter"
curl \
  http://$OPS_IP:8888/cluster-configs \
  -X POST \
  -d \
  "{
     \"cassandra\": {
       \"seed_hosts\": \"$SEED_IP\"
     },
     \"cassandra_metrics\": {},
     \"jmx\": {
       \"port\": \"7199\"
     }
```

```
  }" > /dev/null

echo "Go to http://$OPS_IP:8888/"
```

Script that is executed to create a Cassandra instance within the container

```
#!/bin/bash


# Grab the container IP

ADDR=$(sed '1q;d' /etc/hosts | cut -f 1 | awk '{print $1}')

echo "IP Address is: $ADDR"


# Check if a seed was provided

SEED=${SEED:=$ADDR}

echo "Seed is: $SEED"


if [ ! -f /root/.cassconfig ]; then

    echo "Filling in the blanks inside cassandra.yaml"

    # Target files

    ENV_FILE=/opt/cassandra/conf/cassandra-env.sh

    CONF_FILE=/opt/cassandra/conf/cassandra.yaml

    # Add heap settings

    echo MAX_HEAP_SIZE="4G" >> $ENV_FILE

    echo HEAP_NEWSIZE="800M" >> $ENV_FILE

    # Add broadcast IPs
```

```
    sed -i 's/\(listen_address: \)\([[:alnum:].]*\)/\1'$ADDR'/' $CONF_FILE
    sed -i 's/\(seeds: \)\(["[:alnum:].]*\)/\1'"$SEED"'/' $CONF_FILE
    sed -i 's/\(rpc_address: \)\([[:alnum:].]*\)/\1'$ADDR'/' $CONF_FILE


    touch /root/.cassconfig
fi


# Start server
cd /opt/cassandra
bin/cassandra > out.txt &
```

Script is executed to start the OpsCenter agent within the containers running the Cassandra instance:-.

```
#!/bin/bash
if [[ -z $OPS_IP ]]; then
        echo "No OPS_IP provided, agent won't start"
        tail -f /dev/null
else
        echo "OpsCenter IP is: $OPS_IP"
        if [ ! -f /root/.agentconfig ]; then
                touch /opt/agent/conf/address.yaml
                CONF_FILE=/opt/agent/conf/address.yaml
```

```
                echo "Filling in the blanks inside address.yaml"
                cd /opt/agent/conf
                sleep 10
                echo "stomp_interface: $OPS_IP" >> $CONF_FILE
                touch /root/.agentconfig
        fi

        # Wait 2 mins and start agent
   echo "Sleeping 2min before starting agent"
   sleep 30
        cd /opt/agent
        bin/datastax-agent > out.txt &
fi
```

Script that is executed to create an instance of OpsCenter within the container

```
#!/bin/bash
cd /opt/opscenter
bin/opscenter
```

3.3 Automated script to create a 5 node Cassandra Cluster with OpsCeter

```
#!/bin/bash
echo "Starting OpsCenter"
docker run -d -P --name opscenter opensuse-13.2_ops:latest
sleep 10
docker exec opscenter /etc/service/opscenter/run
sleep 30
OPS_IP=$(docker inspect -f '{{ .NetworkSettings.IPAddress }}' opscenter)
echo "Starting node cass_new1"
docker run -d -P --name cass_new1 -e OPS_IP=$OPS_IP opensuse-13.2_cass:latest
sleep 10
docker exec cass_new1 /etc/service/cassandra/run
sleep 10
docker exec cass_new1 /etc/service/agent/run
sleep 30


SEED_IP=$(docker inspect -f '{{ .NetworkSettings.IPAddress }}' cass_new1)
for name in cass_new{2..5}; do
  echo "Starting node $name"
  docker run -d -P --name $name -e SEED=$SEED_IP -e OPS_IP=$OPS_IP opensuse-
13.2_cass:latest
  sleep 10
```

```
  docker exec $name /etc/service/cassandra/run
  sleep 10
  docker exec $name /etc/service/agent/run
  sleep 30
done
echo "Registering cluster with OpsCenter"
curl \
  http://$OPS_IP:8888/cluster-configs \
  -X POST \
  -d \
  "{
     \"cassandra\": {
       \"seed_hosts\": \"$SEED_IP\"
     },
     \"cassandra_metrics\": {},
     \"jmx\": {
       \"port\": \"7199\"
     }
  }" > /dev/null

echo "Go to http://$OPS_IP:8888/"
```

Build Cassandra image:

$cd docker-cassandra-master

$chmod +x cassandra-run agent-run

$docker build –t opensuse-13.2_cass .

Build OpsCenter image:

$cd docker-opscenter-master

$chmod +x run

$docker build –t opensuse-13.2_ops .

From the parent directory:

$bash Script_cassandra

Once completed, verify the installation of Cassandra and Opscenter.

$docker ps –a

The above command lists all the Docker containers with Container ID, Port and Name of the container. An example is as follows:

The above mapping indicates that the OpsCenter service is running on some port of the host machine. On the browser, go to the following link to view the Cassandra cluster.

http://<IP_ADDRESS_OF_HOST_MACHINE>:<port>/opscenter

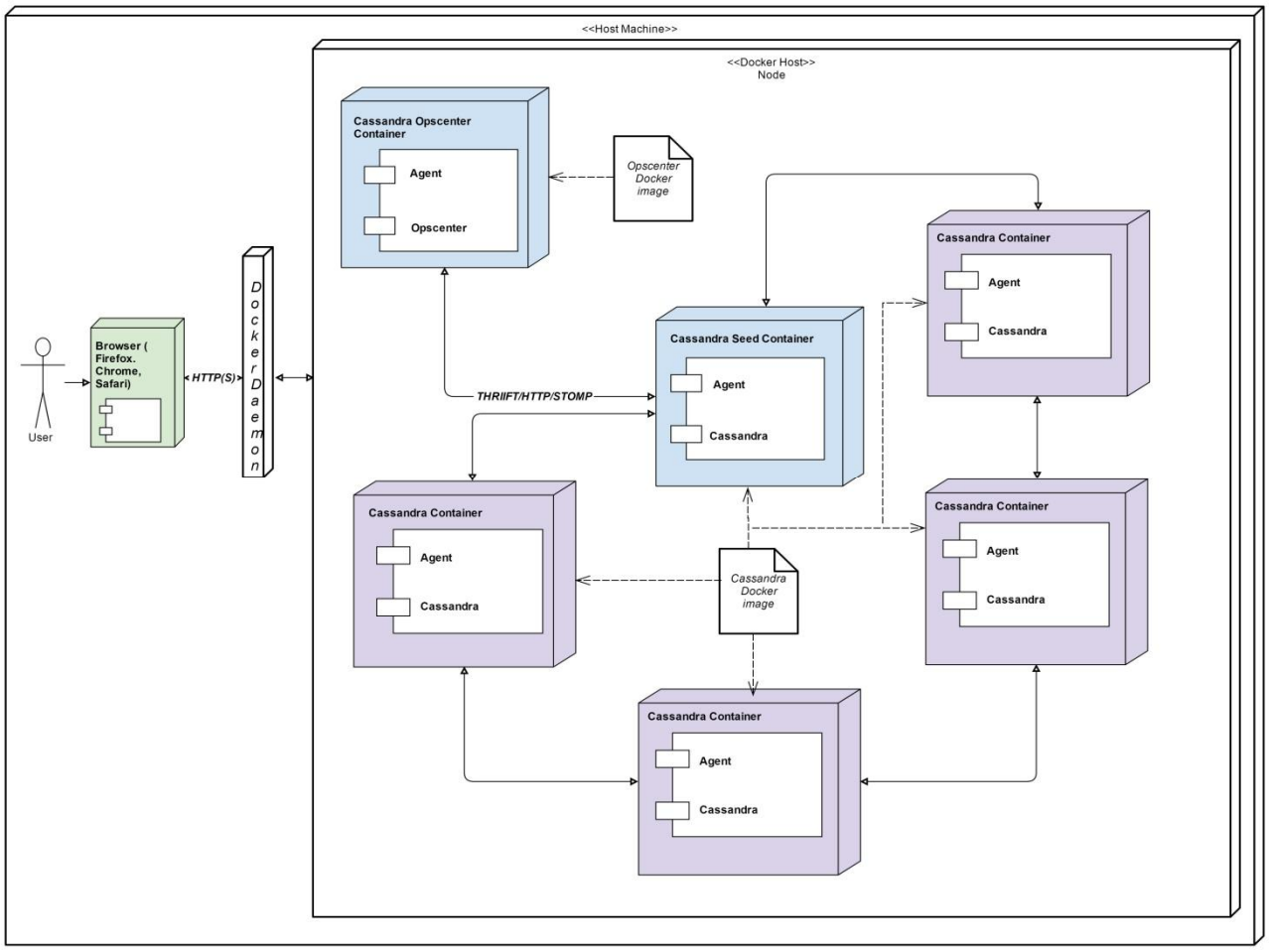

Fig 3.1: Cassandra cluster and OpsCenter setup on Docker

4. MONGO DB SETUP ON DOCKER

MongoDB:-

MongoDB (from humongous) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

4.1 Docker File for Mongodb:-

DockerFile:-

```
# Dockerizing MongoDB: Dockerfile for building MongoDB images
# Based on ubuntu:latest, installs MongoDB following the instructions from:
# http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/
# Format: FROM    repository[:version]
FROM       ubuntu:latest

# Format: MAINTAINER Name <email@addr.ess>
MAINTAINER M.Y. Name myname@addr.ess
#environment variable
ENV http_proxy=http://<username>:<password>@<IP>:<port>
ENV https_proxy=http://<username>:<password>@<IP>:<port>

# Installation:
```

```
# Import MongoDB public GPG key AND create a MongoDB list file

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80    --recv 7F0CEB10

RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | tee

/etc/apt/sources.list.d/10gen.list

# Update apt-get sources AND install MongoDB

RUN apt-get update -o Acquire::http::timeout=500

RUN apt-get install -y mongodb-org -o Acquire::http::timeout=500



# Create the MongoDB data directory

RUN mkdir -p /data/db



# Expose port 27017 from the container to the host

EXPOSE 27017



# Set usr/bin/mongod as the dockerized entry-point application

ENTRYPOINT usr/bin/mongod
```

Build the Mongodb docker image:-

```
sudo docker build –tag=”myadmin/repo”  .
```

4.2 MongoDB –Sharding:-

-Create the directory like:-

```
docker_mongodb_cluster
|
|-- mongod
|   |-- Dockerfile
|
|-- mongos
    |-- Dockerfile
```

-Docker file for mongod is:-

```
FROM   rakeshraj/mongodb1

EXPOSE   27017

ENTERYPOINT   [“usr/bin/mongod”]
```

-Docker file for mongos:-

```
FROM   rakeshraj/mongodb1

EXPOSE   27017

ENTERYPOINT   [“usr/bin/mongos”]
```

-switch to folder docker_mongodb_cluster and run the commands:-

```
sudo docker build \

  -t rakeshraj/mongodb mongod



sudo docker build \

  -t rakeshraj/mongos mongos
```

-Creating the replica sets:-

Replica Set 1:-

```
sudo docker run \
  -P –name rs1_srv1 \
  -d rakeshraj/mongodb \
  --replSet rs1 \
  --noprealloc –smallfiles


sudo docker run \
  -P -name rs1_srv2 \
  -d rakeshraj/mongodb \
  --replSet rs1 \
  --noprealloc –smallfiles
```

```
sudo docker run \
  -P -name rs1_srv3 \
  -d rakeshraj/mongodb \
  --replSet rs1 \
  --noprealloc –smallfiles
```

-Replica Set 2:-

```
sudo docker run \
  -P -name rs2_srv1 \
  -d rakeshraj/mongodb \
  --replSet rs2 \
  --noprealloc --smallfiles
sudo docker run \
  -P -name rs2_srv2 \
  -d rakeshraj/mongodb \
  --replSet rs2 \
  --noprealloc --smallfiles
sudo docker run \
  -P -name rs2_srv3 \
  -d rakeshraj/mongodb \
  --replSet rs2 \
  --noprealloc –smallfiles
```

Initialize the replica sets:-

First take a note of the IP address and port bindings of all Docker containers.

$ sudo docker inspect rs1_srv1

$ sudo docker inspect rs1_srv2

...

Initialize Replica Set 1:-

Connect to MongoDB running in container `rs1_srv1` (you need the local port bound for 27017/tcp figured out in the previous step).

Note: - to use the shell of running conainer use the command

$ docker exec -it rs1_srv1 bash

mongo --port <port>

MongoDB shell

```
rs.initiate()
rs.add("<IP_of_rs1_srv2>:27017")
rs.add("<IP_of_rs1_srv3>:27017")
rs.status()
```

Docker will assign an auto-generated hostname for containers and by default MongoDB will use this hostname while initializing the replica set. As we need to access the nodes of the replica set from outside the container we will use IP addresses instead.

Use these MongoDB shell commands to change the hostname to the IP address.

MongoDB shell

```
cfg = rs.conf()
cfg.members[0].host = "<IP_of_rs1_srv1>:27017"
rs.reconfig(cfg)
rs.status()
```

Initialize replica set 2:-

The exact same procedure is required for the second replica set. Connect to MongoDB running in container `rs2_srv1`.

mongo --port <port>

MongoDB shell

```
rs.initiate()
rs.add("<IP_of_rs2_srv2>:27017")
rs.add("<IP_of_rs2_srv3>:27017")
rs.status()
```

Use these MongoDB shell commands to change the hostname to the IP address.

MongoDB shell

```
cfg = rs.conf()
cfg.members[0].host = "<IP_of_rs2_srv1>:27017"
rs.reconfig(cfg)
rs.status()
```

Create Config Servers:-

Now we create three MongoDB config servers to manage our shard. For development one config server would be sufficient but this shows how easy it is to start more.

```
$ sudo docker run \
  -P -name cfg1 \
  -d rakeshraj/mongodb \
  --noprealloc --smallfiles \
  --configsvr \
  --dbpath /data/db \
  --port 27017

$ sudo docker run \
  -P -name cfg2 \
  -d rakeshraj/mongodb \
  --noprealloc --smallfiles \
  --configsvr \
  --dbpath /data/db \
  --port 27017

$ sudo docker run \
  -P -name cfg3 \
  -d rakeshraj/mongodb \
```

```
  --noprealloc --smallfiles \
  --configsvr \
  --dbpath /data/db \
  --port 27017
```

Use **docker inspect** to figure out the IP addresses of each config server. We will need this to start the MongoDB router in the next step.

Create a Router:-

A MongoDB router is the point of contact for all the clients of the shard. The `--configdb` parameter takes a comma separated list of IP addresses and ports of the config servers.

Note :- Here using the mongos image created in the first part.

```
sudo docker run \
  -P -name mongos1 \
  -d rakeshraj/mongos \
  --port 27017 \
```

```
  --configdb \
    <IP_of_container_cfg1>:27017, \
    <IP_of_container_cfg2>:27017, \
    <IP_of_container_cfg3>:27017
```

Initialize the shard:-

Finally we need to initialize the shard. Connect to the MongoDB router (you need the local port bound for 27017/tcp) and execute these shell commands:

```
mongo --port <port>


# MongoDB shell


sh.addShard("rs1/<IP_of_rs1_srv1>:27017")
sh.addShard("rs2/<IP_of_rs2_srv1>:27017")
sh.status()
```

5. CREATING DOCKER PRIVATE REGISTRY WITH AUTHENTICATION

5.1 Starting Registry Container:-

$cd /mnt/

$mkdir container_storage

$cd container_storage

$mkdir docker-registry

$cd /mnt/container_storage/docker-registry/

-Download the dockerized container registry from Docker Hub (publicly available Docker registry)

$docker run -d -p 5000:5000 -v /mnt/container_storage/docker-registry/:/tmp/registry registry

$docker ps –a

Note: Check if the docker registry container is started or not. If not started, run **docker start <container-name>**

5.2 Setup and configure Nginx:-

$sudo zypper install apache2-utils

$wget http://nginx.org/packages/sles/12/x86_64/RPMS/nginx-

1.6.21.sles12.ngx.x86_64.rpm

$rpm -ivh nginx-1.6.2-1.sles12.ngx.x86_64.rpm

$cd /etc/nginx

Creating the first user as <username>

$htpasswd2 -c /etc/nginx/docker-registry.htpasswd <username>

Add the following lines to the nginx.conf file under the http block:

$vi nginx.conf

```
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/docker-registry;
```

$mkdir sites-enabled sites-available

$cd sites-available

Create Nginx configuration file for 'docker-registry:-

File contents of /etc/nginx/sites-available/docker-registry

```
upstream docker-registry {
 server 0.0.0.0:5000;
}
server {
  listen 8080;
  server_name docker-registry.example.com;
  ssl on;
  ssl_certificate /etc/ssl/certs/docker-registry.example.com.1.pem;
  ssl_certificate_key /etc/ssl/private/docker-registry;
```

```
  proxy_set_header Host       $http_host;   # required for Docker client sake
  proxy_set_header X-Real-IP  $remote_addr; # pass on real client IP
  client_max_body_size 0; # disable any limits to avoid HTTP 413 for large image
uploads
  # required to avoid HTTP 411: see Issue #1486   chunked_transfer_encoding on;
  location / {
    # let Nginx know about our auth file
    auth_basic              "Restricted";
    auth_basic_user_file    docker-registry.htpasswd;
    proxy_pass http://docker-registry.example.com:5000;
  }
  location /_ping {
    auth_basic off;
    proxy_pass http://docker-registry.example.com:5000;
  }
  location /v1/_ping {
    auth_basic off;
    proxy_pass http://docker-registry.example.com:5000;
  }
}
```

$cd sites-enabled

$ln -s /etc/nginx/sites-available/docker-registry docker-registry

5.3 Creating SSL certificates

$mkdir ~/certs

$cd ~/certs

Generating a new root key:

$openssl genrsa -out devdockerCA.key 2048

Generating a root certificate by entering the correct credentials when prompted:

Note: ***When prompted for the "Common Name", make sure to type in the domain of the server (e.g: docker-registry.example.com)***

$openssl req -x509 -new -nodes -key devdockerCA.key -days 10000 -out devdockerCA.crt

Generating a key for the server:

$openssl genrsa -out dev-docker-registry.com.key 2048

Certificate signing request, by entering the correct credentials when prompted:

Note: ***When prompted for the "Common Name", make sure to type in the domain of the server (e.g: docker-registry.example.com)***

$openssl req -new -key dev-docker-registry.com.key -out dev-docker-registry.com.csr

Self signing the certificate request:

openssl x509 -req -in dev-docker-registry.com.csr -CA devdockerCA.crt -CAkey devdockerCA.key -CAcreateserial -out dev-docker-registry.com.crt -days 10000

$cp dev-docker-registry.com.key /etc/ssl/private/docker-registry

$cp devdockerCA.crt /etc/pki/trust/anchors/

Note: The above directory structure is for SUSE Linux and maybe different for other flavors of Linux.

$update-ca-certificates

$service nginx restart

$systemctl restart docker

$docker login https://docker-registry.example.com:8080

Enter the credentials when prompted.

Adding images to the private registry:

$docker tag <image_name> <docker-registry.example.com:8080/image_name>

$docker push <docker-registry.example.com:8080/image_name>

6. DOCKER ORCHESTRATION

6.1 Introduction:-

The Docker and its ecosystem are very good for managing images and running containers in a specific host. Docker is not able to manage when containers are across multiple nodes, or schedule and manage tasks at multiple nodes. Docker is good when you can manually configure and manipulate host. But this is not look feasible when you have hundreds or thousands of nodes. To automate the workflow in the cloud environment when you have to run big applications on many host and these application build up using linking more the one containers.

To manage the workload on the distribution system it needs help of other tools. There are many tools to provide the orchestration in docker. There is no easy answer that which one is better than other one. The selection is mostly based on the requirement of your distributed system.

6.2 Configuration Management tools Pros and Cons:-

Configuration Management Tool	Pros	Cons
Fig/Compose	-Docker Compose uses the same API used by other Docker commands and tools. -Easy to learn and use with docker.	-It can use for one app per environment.

	-It uses Yaml file for configuration	
Ansible	-Excellent Performance, agent-less installs and deploy. - Low overhead, playbook based. -It uses Yaml file for configuration entries. Defining features and role for deployment now looks more like compiling a grocery list than writing algorithm with routines and methods. -Based on ubiquitous python language. -CLI accepts command in almost any language. -Provide security using SSh/ SSH2.	-Very new, Not so matured or too much used and tested. -It can work better for small number of hosts.

Puppet	- Mature Solution. - Good GUI	-Ruby-based, performance questionable compare to python-based CM tools. -It is not easy to use (Yaml configuration more easy to use). -It can work better for small number of hosts.
Chef	-Mature solution. -Larger community, with large number of modules and configuration recipes.	-Ties users to ruby, not so easy to learn and deploy. -Relies on JSON which is not as easy to use as Yaml -It can work better for small number of hosts.
Helios	-Helios provides a HTTP API as well as a command-line client to interact with servers running your containers. -It also keeps history of	-Not works well for large number of nodes.

	events in your cluster including information such as deploys, restarts and version changes. -It provides repeatable, straightforward, fault-tolerant solution.	
Clocker	-Automatically create and manage multiple Docker hosts in cloud infrastructure -Intelligent container placement, providing fault tolerance, easy scaling and better utilization of resources -Use of any public or private cloud as the underlying infrastructure for Docker Hosts -Deployment of existing Brooklyn/CAMP blueprints to Docker locations, without	-For very large scale application mesos is better solution.

	modifications -Clocker support single-click deployment and runtime management of multi-node applications that can run on containers distributed across multiple hosts, using the Weave SDN. -Clocker is responsible for assigning the Weave network IP addresses; it can ensure that a container's address stays the same, even after rebooting or moving to a different host, making applications much more resilient in the face of failures.	
Mesos + Marathon	-Generic solution to resource usage problem. -Open Source Project	-Mesos is complicated to deploy, manage and use. -It's not better solution for

	(Apache Foundation). -It can work better for very large number of nodes. -In production use at twitter, ebay etc.	small number of nodes.

Table 6.1 Configuration Management tools Pros and Cons

GLOSSARY

Aufs:-

aufs is a Linux file-system that docker supports as a storage backend. It implements the union mount for Linux file systems.

Boot2docker:-

boot2docker is a lightweight Linux distribution made specifically to run docker containers. It is common to choice for a VM to run docker on windows and Max OS X. boot2docker can also refer to the boot2docker management tool on windows and Mac OS X which manages the boot2docker VM.

Btrfs:-

btrfs is a Linux filesystem that docker supports a storage backend. It's a copy-on-write file-system.

Build:-

build is the process of building docker images using a Dockerfile. The build uses a Dockerfile and a 'context'. The context is that the directory in which the image is built.

Cgroups:-

cgroups (control groups) is a Linux kernel feature that limits, accounts for and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes/ docker relies on cgroups to control and isolate resources limits.

Compose:-

Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, and then spin your application up in a single command which does everything that needs to be done to get it running.

Container:-

A container is a runtime instance of a docker image. If an image is like a binary, a container is like a running instance of the binary or process.

The concept is taken from shipping containers, which define a standard to ship globally. Docker defines a standard to ship software.

Data Volume:-

A data volume is a specifically-designed directory within one or more containers that bypasses the Union File System. Data Volumes provide several features for persistent or shared data, independent of container's life cycle. Docker therefore never automatically deletes volumes when the container is removed, nor will it "garbage collect" volumes that are no longer referenced by a container.

Docker:-

Docker is a platform for developers and sysadmins to develop, ship, and run applications. Docker lets you quickly assemble applications from components and eliminates the friction that can come when shipping code. Docker lets you get your code tested and deployed into production as fast as possible.

Docker Hub:-

The Docker hub is a centralized resource for working with docker and its component. It provides the following services:-

-Docker image hosting

-User authentication

-Automated image builds and work-flow tools such as build triggers and web hooks

-Integration with GitHub and BitBucket

DockerFile:-

A Dockerfile is a text document that contains all the commands you would normally execute in order to build a Docker image. Docker can build images automatically by reading the instruction from a Dockerfile.

File-system:-

A file system is the method an operating system uses to name files and assign them locations for efficient storage and retrieval.

Examples:-

Linux : ext4, aufs, btrfs, zfs

Windows : NTFS

Image:-

Docker images are the basis of containers. An image is an ordered collection of root file system changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union file system stacked on top of each other. An image does not have state it never changes.

libconatiner:-

libcontianer provides a native Go implementation for creating containers with namespace, cgroups, capabilities, and file-system access controls. It allows you to manage the lifecycle of the container performing additional operations after the container is created.

Link:-

Links provide an interface to connect Docker containers running on the same host to each other without exposing the hosts network ports. When set up a link, its create a conduit between a source container and a recipient container. The recipient can then access select data about the source. To create a link, you can use the –link flag.

Machine:-

Docker Machine is a docker tool which makes it really easy to create a docker hosts on computer, on cloud providers and inside your own data center. It creates servers, installs Docker on them, then configure the docker client to talk to them.

Overlay:-

OverlayFS is a filesystem service for Linux which implements union mount for other file systems. It is supported by the Docker daemon as a storage driver.

Registry:-

A registry is a hosted service containing repositories of images which responds to the Registry API.

The default registry can be accesed by using a browser at docker hub or using the ‘docker search’ command.

Repository:-

A repository is a set of Docker images. A repository can be shared by pushing it to a registry server. The different images in the repository can be labeled using tags.

Swarm:-

Swarm is a native clustering tool for Docker. Swarm pools together several Docker hosts and exposes them as a single virtual Docker host. It serves the standard Docker API, so any tool that already works with Docker can now transparently scale up to multiple hosts.

Tag:-

A tag is label applied to a Docker image in a repository tags are how various images in a repository distinguished from each other.

Union File System:-

Uniom file system, or UnionFS, is file systems that operate by creating layers, making them very lightweight and fast. Docker uses union file systems to provide the building blocks for containers.

Virtual Machine:-

A Virtual Machine (VM) is a program that emulates a complete computer and imitates dedicated hardware. It shares physical hardware resources with other users but isolates the operating system. The end user has the same experience on a Virtual Machine as they would have on dedicated hardware.

Compared to a container VM is heavier to run, provides more isolation, gets its own set of resources and does minimal sharing.

BIBLIOGRAPHIC REFERENCES

1. “Docker“ http://docs.docker.com/
2. “Docker SUSE Installation“ http://docs.docker.com/installation/SUSE/
3. “Docker Hub“ https://hub.docker.com/account/signup/
4. “Docker MongoDB“ http://docs.docker.com/examples/mongodb/
5. “Docker Private Registry“ https://docs.docker.com/registry/deploying/
6. “Cassandra“ http://www.datastax.com/products/datastax-enterprise-production-certified-cassandra
7. “Docker Compose” http://docs.docker.com/compose/yml/
8. “Docker Swarm” https://docs.docker.com/swarm/
9. “Docker Orchestration” http://blog.docker.com/tag/orchestration/