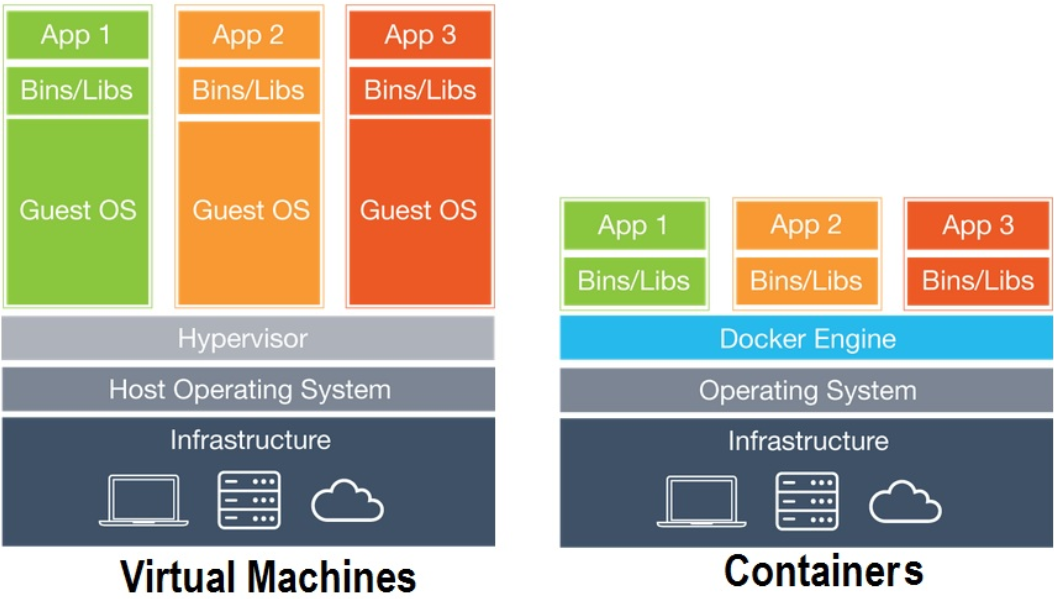
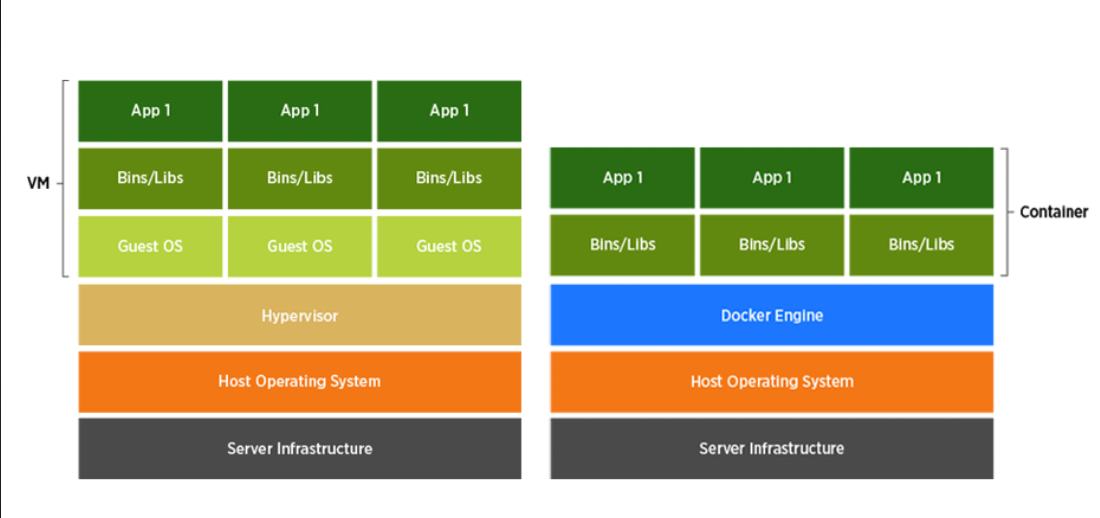
**Docker**

**Docker containers and Kubernetes:**



In the figure **Containers refers to Docker containers**



**Why containers?**

* Docker containers are light weight they consume less compute resources.
* There is no guest OS in case of containers.
* Docker Containers are portable. (i.e.) – container will have application and all its dependencies such that take a container put it on any server it works without any dependency issues.
* Container creation time is fast. (i.e) containers are created & started in seconds (i.e creation and startup time is less than 2 seconds.)
* **Deployment is easy & fast.**

**Process of creating containers:**

* Step1: Write a Docker file

In the same file docker file contains instructions which tells what should be part of our Docker image

* Step2: Using Docker file build docker image
* Step3: Using Docker image create Docker containers. (we can create 1000’s of docker containers)

**Install Docker on Linux Machine:**

**#sudo yum install docker -y**

Docker is a service we need to start the service

**#sudo usermod -aG docker ec2-user** (add ec2 user to docker group, such that ec2 user can manage docker without using Sudo)( to effect changes exit and reconnect)

**#sudo service docker start**

**#sudo chkconfig docker on** (enable docker on reboot (automatically start docker ))

**#docker info**

Run my app on container:

**cd myweb/**

**git pull**

Step1: **mvn clean package**

Step2: **docker build -t myweb:1.0 .** (this command build image with name myweb and tag 1.0 < . > indicates build context)

* **docker images**
* **docker run -d -p 8080:8080 myweb:1.0** (creating a container )

-d indicates detached mode (i.e- run containers in background),,

-p is the port mapping (host-port: container-port)

Container port is the port on which tomcat is running in a container.

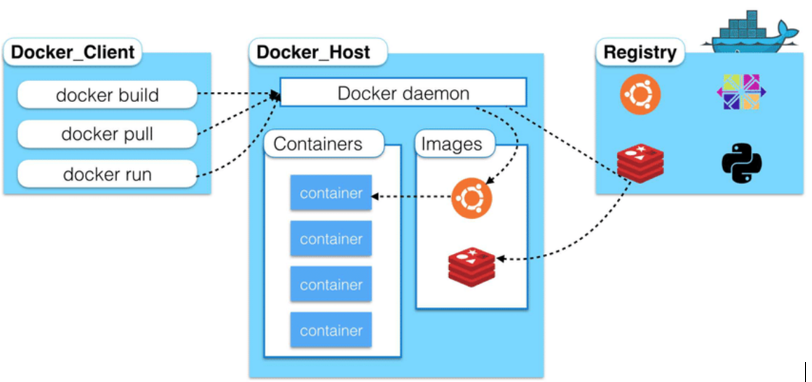
Host-port is the port allocated on the host using which we access containers on the host.

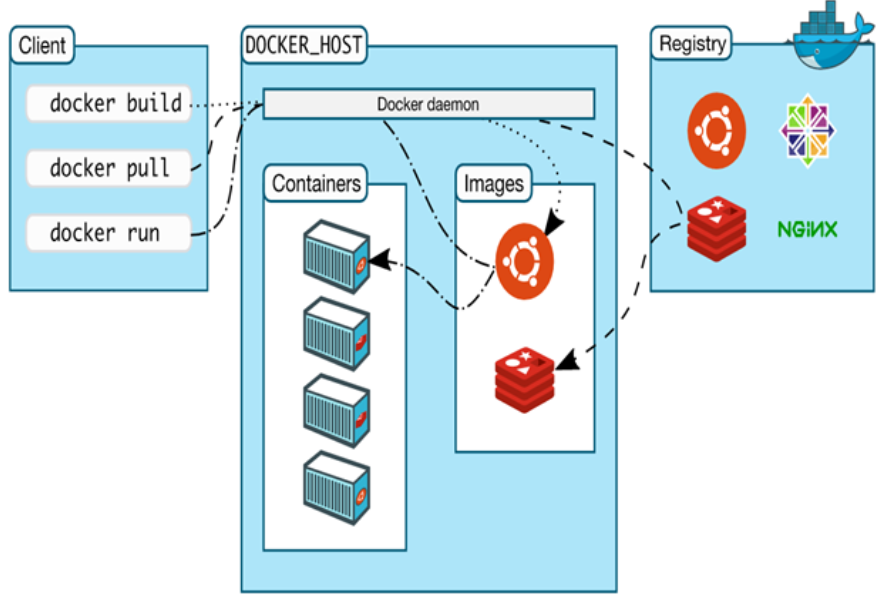
go to web browser take public Ip and paste it browser (publicip8080:8080 /myweb/)

**#docker ps** (to list containers)

-----------------------------------------------31-10-19--------------------------------------------

**Docker architecture:**





**Client:** The machine on which we run docker commands (client and docker host can be same, client and docker host can be different)

**Docker Host :** The machine on which docker is installed. It can contain containers and images locally.

**Registry:** Server which will host docker images.

* **Docker** has docker hub

hub.docker.com - it’s a registry available over internet

docker hub is hosted by docker inc it is available for all.

Create a/c in docker hub

Push images to docker hub

Step1:

In order to push images to docker hub we need to tag our images as follows.

**#docker build -t** **lokesh/hari-app:1.0 .** (dockerhub-id/image-name:tag)

**#docker images**

Login to docker hub

**#docker login**

Username:hubid (lokesh)

Password: give password

**#docker push lokesh/hari-app:1.0**

Want to pull any image :

**# docker pull lokesh/hari-app:1.0** (if image is available at local it wont pull , it says up to date)

Basic Docker commands :

**#docker images** (list all images)

**#docker rmi (image id’s or image names)** – rmi stands for remove image. ( to remove multiple images give space paste image id or image name)

If image is used by container we cant remove that.

**#docker images -q** -(returns only image id’s)

#**docker rmi $(docker images -q) –** to remove all images (If image is used by container we cant remove that)

**Containers:**

**docker run -it -p 8080:8080 lokesh/hari-app:1.0** (-it 🡪 interactive terminal (its going to show logs in the terminal))

**docker ps** – displays details about running containers.

**#docker ps -a (displays running + exited or stoped containers)**

**#docker ps -ad ( returns or diplays docker id’s )**

**#docker start container id**

**#docker stop container id**

**#docker rm container id**

If container is running we cant remove but we can force remove.

**# docker rm -f (force remove)**

**# docker rm -f $(docker ps -aq)**

**#docker run -d -p 8080:8080 --name=beere lokesh/hari-app:1.0 (to give name or else its picks random name)**

what is **-P** in docker run command?

* publish all exposed ports to random port.

**#docker run --rm -d -p --name=beere lokesh/hari-app:1.0**

--rm in docker run command automatically removes the container when container stops.

**Another person notes:**

**Bridge Network:**

-          It is used by the containers running on same host communicate with each other.

-         In bridge network has two flavors

o   1.defalult bridge

o   2. Custom bridge.

-         Default Bridge :

o    When you install Docker, Default Bridge is automatically created.

o    #Docker network ls

o    You can see the bridge details by giving above command.

o    #docker network inspect <network ID>

o    You can see who all part of this n/w.

o    #docker run –d –p 80:5000 kammana/app:1.0

o    Container created, need to check which part of the n/w is this container.

o    Run another container with other port no

o    Now you are running two containers.

o    Both the container we have created can talk each other…bcz we have created in the same n/w.

o    Lets log in to one container and ping each other and check.

o    #docker ps –aq

o    #dcoker exec –it <container ID>  ash

o    #ping  <another container IP>

o    Run the same container by giving names

o    #docker rm –f $(docker ps -aq)

o    #docker run –d –p 80:5000  --name=hostone kammana/app:1.0

o    #docker run –d –p 90:5000  --name=hosttwo kammana/app:1.0

o    Login to one container and ping with the name of the other container.

o    #docker exec **–it <container** ID>  ash

o    #ping hosttwo

o    You should be able to ping each other.

-          Custom Bridge:

o    Just remove the old container which was created earlier.

o    #docker rm –f $(docker ps -aq)

o    **#docker network create –driver=bridge javahome**

o    Created custom bridge netwrok.

o    #docker network ls

o    **#docker run –d –p 80:5000  --name=hostone  - -network=javahome kammana/app:1.0**

o    #docker run –d –p 90:5000  --name=hostone  - -network=javahome kammana/app:1.0

o

o    #docker network inspect javahome

o    #docker exec –it hostone ash

o    #ping hosttwo

**Deploying Multi container application on single host: (not working)**

-          #docker build –t kammana/python-redis:1.0 **.**

-          This image is available in hari docker hub.

-          #docker push kamanna/python-redis:1.0  (to push image to remote)

-          Lets deploy the python app

-          Docker rm $(docker –aq)

-          #docker run –d –p 80:5000 - -name=python kammana/python-redis:1.0

-          Now you can use through browser

-          Take public op of ec2 with port 80

-          #docker run –d - -name=redis redis:latest

-          No need to mention port details, bcz we are not directly accessing the redis through browser.

-          #docker rm –f $(docker ps –aq)

-          Run python conatiner

-          **#docker run –d –p 80:5000 - -name=python - -network=javahome  kammana/python-redis:1.0**

-          Run redis container

-          **#docker run –d  - -name=redis - -network=javahome  redis:latest**

Steps:

-          Create custom bridge network.

-          **#docker network create –driver=bridge javahome**

-          Run python container

-          **#docker run –d –p 80:5000 - -name=python - -network=javahome  kammana/python-redis:1.0**

-          Run redis container

-          **#docker run –d  - -name=redis - -network=javahome  redis:latest**

**Docker Compose** : (docker-compose.yml is working)

-          Using docker compose we can describe details of micro service in a yaml file.

-          Docker compose reads that file and will setup all microservices on single host with single command.

-          Docker compose is used to automate setting up dev/test environment.

**Installing Docker Compose:**

-          As of today docker compose doesn’t come with docker installation.

-          We need to install **docker compose** explicitly.

-          Google: docker compose install

-          <https://docs.docker.com/compose/install/Choose> the OS

-          Enter below commands

-

-          **sudo curl -L "**[**https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname**](https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname%0D-s)-$(uname)**-m)" -o /usr/local/bin/docker-compose**

-          **sudo chmod +x /usr/local/bin/docker-compose**

-          docker-compose –version

-          then

-          remove old containers if any.

-          We have one docker compose file in git hub “docker-compose”

-          Use the same file to create docker image.

-          Goto git hub of

-          <https://github.com/javahometech/docker-compose>

-          EC2- #wget copy the URL(of the docker-compose)

-          #docker-compose up –d

-          Go to browser and enter <publicip:8080>

-          #docker-compose down <it will stop the compose>

-          Task :

o    Inside docker-compose allocate one CPU each service.

-          <https://hub.docker.com/u/kammana/>

Practise :

-          Docker-compose file is working.

-          Commands:

-          dcoker-compose --version

-               docker-compose --version

-               wget <https://raw.githubusercontent.com/javahometech/docker-compose/master/docker-compose.yml>

-               docker-compose up -d

-               docker ps

-               docker rm -f $(docker ps -aq)

-               docker-compose up -d

-          [swarm-stack.yml](https://github.com/javahometech/docker-compose/blob/master/swarm-stack.yml)

**Day 29:**

**Play with Docker**:

-          **Google :** play with docker.

-          We should have docker hub account.

-          Login with docker hub account

-          You can Crete upto 05 VMs

-          Add instance-> docker is preinstalled.

-          #docker –v

-          #Docker run –d –p 80:80 httpd

-          It will pull apache docker

-          Right click on above 80 and paste on the browser.

-          #Docker run –d –p 8080:80 nginx

-          By 8080 you can able to access the nginx.

**IMP\*\***

**Docker Volumes:**

-          By default data stored in the container itself.

-          All tables and application data stored in container by default. To prevent this volume comes in the picture.

-          By default data generated on a container is lost, when you terminate that container.

-          You can mount data volume in to the host.

-          If you wanna persists data generated by the container, we have to create docker volume. Volume persists data on the host, if container crashers, we can create new container pointing to a volumes.

-          volume reside on host.

-

Ec2: lets create volumes

-          #docker volume list

-          #docker volume create vol1 (vol1 is volume name)

-          #docker volume inspect <volume name>

-          lets try Jenkins pull dockerhub

-          [hub.docker.com](http://hub.docker.com/)

-          Jenkins

-          Docker run –p 8080:8080 –p 50000:50000 –v /home

-          Docker run –p –d 8080:8080 –p 50000:50000  jenkins

-

|  |
| --- |
| Volume name |

It will pull the latest Jenkins.(it’s with out volume)

-

|  |
| --- |
| Image Name |

#dokcer exec –it <container id> /bin/bash

-          To enter into the Jenkins container.

-          #docker volume create Jenkins

-          #docker volume inspect  Jenkins

-          Volumes are stored in the host only.

-          #docker run –d –p 8080:8080 –p 50000:50000 –v   jenkins:/var/jenkins\_home **jenkins**

-          #cd:/var/kenkins\_home

-          here you will get intial admin password in host itself.

-          Now create one job in jenkins and crash and reinstall the jenkins, you will have all config details stored on the volume.

-          Better to point your container to volume for better practice and safe.

**Day 30:**

**Docker orchestration**:

-          These tools are used for maintaining multiple micro services on multi node production cluster.

-          Docker swarm is orchestration tool.

**Docker orchestration Tools:**

**1.**      Kubernetes from Google : it is frame work which maintains your Docker.

**2.**      Docker swarm, from Docker Inc

**3.**      Mesos, from Apache

**4.**      Openshift, from redhat(open shit is plat form, the underline is kubernetes)

a.       It is platform for kubernetes

**5.**      Rancher, from rancher Labs

a.       Is the platform supports, kubernates, swarm,mesos etc……

**6.**      AWS EKS-Elastic Kubernetes Service

a.       Is kubernates platform in AWS

**My notes start from here:**

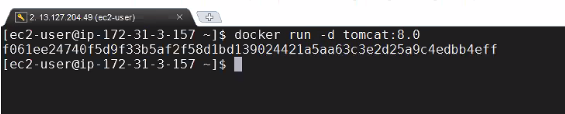
**01-11-19:**

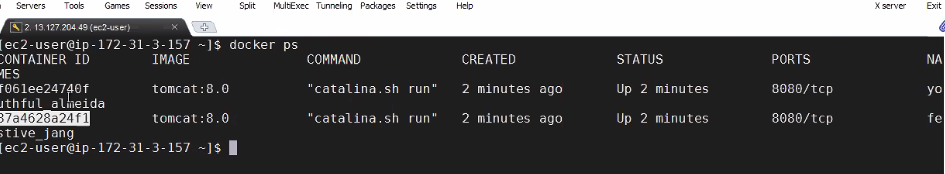
**Docker Networking:**

The objective of this discusiion is how containers talk to each other.

* First we will focous on how containers communicate when they are running on same host.

**We have bridge n/w in two flavours:**

* **Default bridge**
* **Docker network list**
* **Default bridge is implicitly created**
* **By default we create they join default bridge**
* **#Docker network list -to list all n/w’s on this host (docker network list ) gives list of n/w’s then copy id**
* **Docker network inspect paste id**
* 
* Containers joining defaut adress can talk to each other using IP adress.



* We can get ip adress of container 1 and container 2 using

Command take you in to container in below fig :



Docker ecec -it id(872……. Bash

Ping 172.182.16.12

In default bridge communicates through host names (container name) we can consider container ae as host name

Communication through host namee wont work

Docker rm -f $(docker ps-a)

Docker network create –javahome –driver=bridge

-----------------------------lets run 2 containers

Docker run -d –network=javahome –name=tomcattwo tomcat 8.0 (

Docker ps

Lets get into container 1 and ping container 2

Ping tomcat2 (

(Flask is a frame work for developing apps)

Docker ps docker rm-f id

Docker run -d redis

Git clone cd in to pythone app

Docker build -t python 1.0 . (image created

Docker run -d -p 8080:5000 python:1.0

Docker rm -f $(docker ps -aq) – run on custom bridge

Docker run -d –name=redis –network=javahome redis

Overlay n/w :overlay n/w connects multiple docker demons together and enable swarm services to comm with each other

It us used in prod for multi hosts.

Containers running on two diff hosts can talk to each other.

**Image from mobile**

Bridge is for single host & overlay is for multiple host

**Docker volumes :**

**IQ)** IF docker container generates data at runtime that data is stored within the container and data is lost when that container is deleted.

If we run databases inside a container we losse data when container is removed to solve this prob we use volummes

Wwe want to persist data generated by the container even if the container is deleted

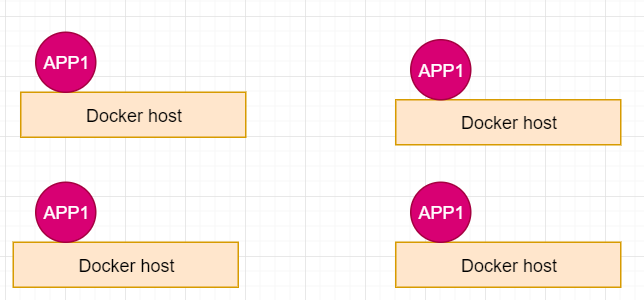
Data is available on volume

**6-11-19**

Running Docker containers in production

(If we run dcker containers in prod on a single node or single host what are the problems)

(how many hosts we required ? – depends on how many app we need)



**Monolithic application** : A big application which is packaged as single binary or 1 single war file is called monolthic app.

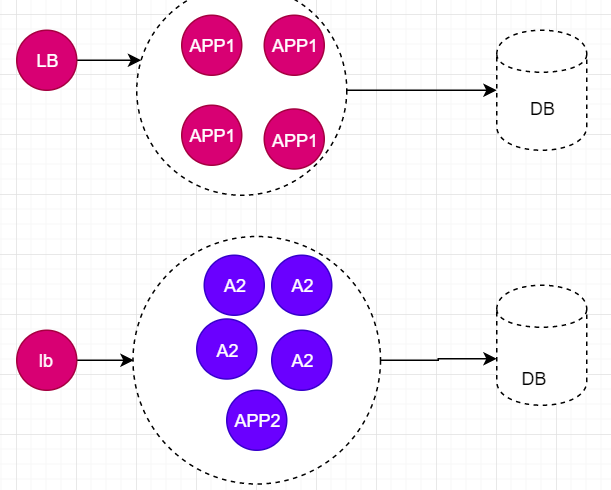
Problems with monolithic applications – to solve them we go for micro services

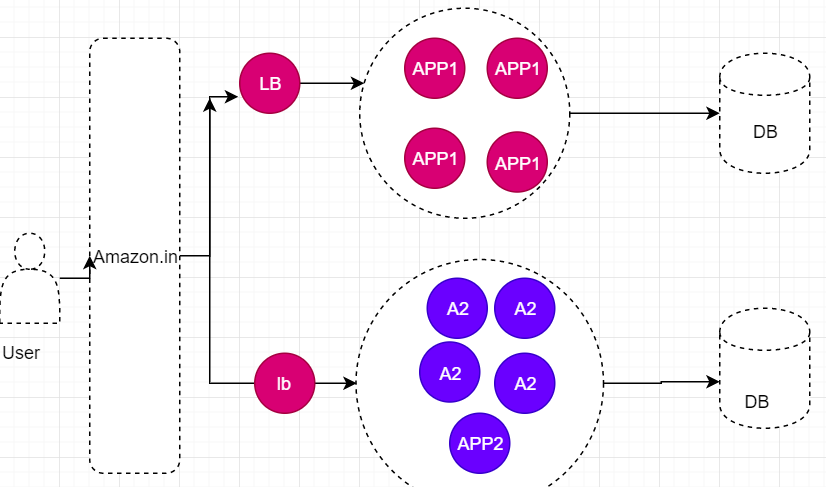
* Code base is failed, doing changes to big code base take lot of time, and to test and to move to prod , ( bcoz of big code base making changes to app more efforts and time)
* For new comers understanding the system take lot of time.
* We cant scale only a specific module or functionality.
* We have to always stick to one technology standard
* If one app fails our complete app go down.

These are the problems with monolithic architecture.

**Micro service architecture:**

* Dividing the monolithic application in to smaller application ( smaller app called micro service
* Every micro services should be small enough such that it can be managed by two developpers its also called two pizza team.
* Increases productivity [ (i.e developpers does not need to focous on complete app they need to focous on their micro service)Lets say amazon has 150 micro services and 300 developpers are required there , we will only care about my micro service ]
* Provides cost efective scaling.(i.e we can scale a specific micro service rather scaling complete monolithic application)
* Every micro service must have it own data base.





In git every micro service will have git repository

Every micro service has CI CD process

Micro services will talk to each other using restfull API’s

We can do deployements of any microservices independently.

**Disadvantages :**

We need to deploy more micro serices

**IQ) Differet deployment techniques ( how you do zero down time deployments)**

* **Rolling updates:**

**Ex :**  we have a micro service with 30 replicas we want to deploy a new version instead of replacing all 30 in one shot ( causes down time) replace them in batch wise i.e replace 30 % at a time. Repeat this until all old containers are replaced with new containers.

* **Blue green deployements:**

Make an replica of that micro service and use that , if we have any isues we can rollback to previous micro service. Once it is good we can delete old one.

**Deploying containers in production**

* Lot of activities comes in to picture if we manage everything on our own
* So we have to depend on third party docker archestration frame work.
* To run containers in prod we have to choose one of the following container archestration tools

1. Kubernates -it’s a google framme work.
2. Docker swam -it is a docker native frame work
3. Mesos – it’s a frame work from apache
4. Cloud – Amazon ECS( elastic container service)

V) amazon EKS (amazon kubernative services)

VI) Azure AKS (azure kubernative services)

Vii) GKS (google kubernative services)

viii) Rancher labs

IX) open shift

07-11-19 (thu)

---------------- Holiday--------

08-11-19 (fri) :

**Setting up kubernates cluster:**

There are several ways to setup kubernates cluster

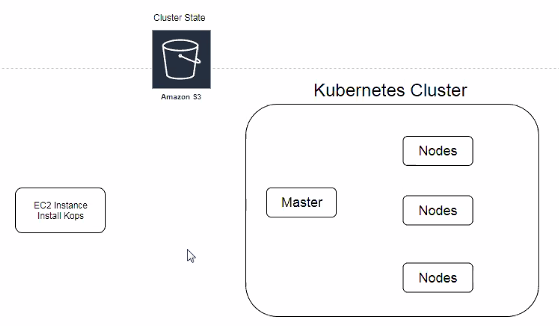
* We can run kubernates anywhere i.e – on our own physical servers
* Virtual machines
* Even on a cloud
* We could use kubeabm to setup kubernates cluster
* Kubespray
* We could use **kops** to setup kubernetes cluster on AWS

**Setting up cluster using Kops:**

Refer to github repository (<https://github.com/javahometech/kubernetes>)

Etcd : it is distributed key value store

There is a special networking



Networking : Kubernetes has its own framework which allows pods ( containers) to communicate each other from any host.

Scheduling : it means finding nodes to run a container. When you deploy pods into kubernates , kubernates has to find the appropriate nodes to create read replicas.this logic is built-in.

Step 1 :

Launch Ec2 instance

Step 2:

Crate IAM role

Step3 :

Install Kops on EC2

Step4 :

Kops is used to build our cluster , then we need kubectl to interact with Cluster

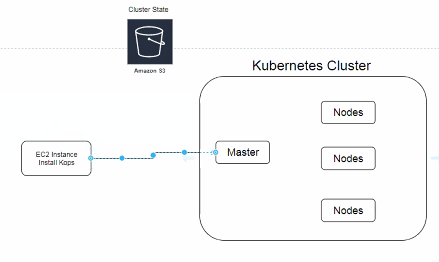
Step5:

Create s3 bucket in AWS

Step6 :

Create private hosted zone in Aws route53

Step 7 : create ssh key pair



From ec2 we want to ssh in to cluster.

Step 9:

**Create kubernetes cluster definition :**

**Here is the Git hub documents refer to this from beginning step**

# [Kubernetes on AWS using Kops

**1. Launch Linux EC2 instance in AWS**

**2. Create and attach IAM role to EC2 Instance.**

Kops need permissions to access

S3

EC2

VPC

Route53

Autoscaling

etc..

**3. Install Kops on EC2**

curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops

**4. Install kubectl**

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

**5. Create S3 bucket in AWS**

S3 bucket is used by kubernetes to persist cluster state, lets create s3 bucket using aws cli **Note:** Make sure you choose bucket name that is uniqe accross all aws accounts

aws s3 mb s3://javahome.in.k8s --region ap-south-1

**6. Create private hosted zone in AWS Route53**

1. Head over to aws Route53 and create hostedzone
2. Choose name for example (javahome.in)
3. Choose type as privated hosted zone for VPC
4. Select default vpc in the region you are setting up your cluster
5. Hit create

**7 Configure environment variables.**

Open .bashrc file

vi ~/.bashrc

Add following content into .bashrc, you can choose any arbitary name for cluster and make sure buck name matches the one you created in previous step.

export KOPS\_CLUSTER\_NAME=javahome.in

export KOPS\_STATE\_STORE=s3://javahome.in.k8s

Then running command to reflect variables added to .bashrc

source ~/.bashrc

**8. Create ssh key pair**

This keypair is used for ssh into kubernetes cluster

ssh-keygen

**9. Create a Kubernetes cluster definition.**

kops create cluster \

--state=${KOPS\_STATE\_STORE} \

--node-count=2 \

--master-size=t2.micro \

--node-size=t2.micro \

--zones=ap-south-1a,ap-south-1b \

--name=${KOPS\_CLUSTER\_NAME} \

--dns private \

--master-count 1

**10. Create kubernetes cluster**

kops update cluster --yes

Above command may take some time to create the required infrastructure resources on AWS. Execute the validate command to check its status and wait until the cluster becomes ready

kops validate cluster

For the above above command, you might see validation failed error initially when you create cluster and it is expected behaviour, you have to wait for some more time and check again.

**11. To connect to the master**

ssh admin@api.javahome.in

**Destroy the kubernetes cluster**

kops delete cluster --yes **]**

**Refer to the below link also**

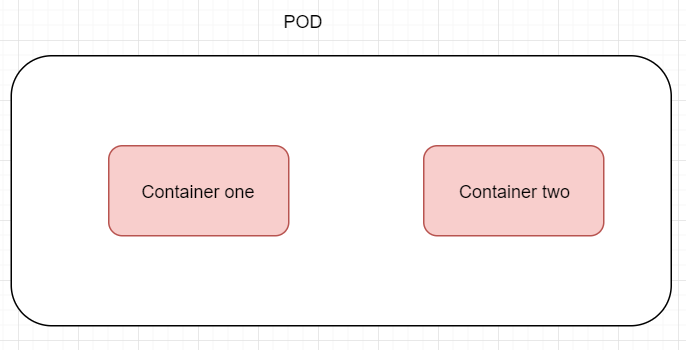
(<https://github.com/kubernetes/kops/blob/master/docs/terraform.md>)

**11-11-19**

Kubernetes pod :-

* Pod is atomic unit of deployment
* To deploy any app in to kubernates we atleast need a pod
* Inside a pod we can have one or more containers

If you have a scenario where couple of containers needs to be deployed together and to be destroyed together then keep it in a single pod.



Kubernates pod will have its own ip adress.

Its also called as cluster IP ,any where with in that cluster it is acessable.

Containers on a pod can talk to each other using local host , If container1 wants to talk to container2 by local host.

**Deployng our 1st pod in kubernates cluster:**

We can rqst kubernates cluster to create a pod

* Using a command
* Write a Yaml document where we describe all the details about our POD.
* **Kubectl create -l https://github.com/javahometech/kubernetes.git**
* **Kubectl get pods**

**By**  defaults pods are not exposed to the internet but I want to test this within the cluster

**Ssh** [**admin@api.javahome.in**](mailto:admin@api.javahome.in)

**Refer to document [**<https://github.com/javahometech/kubernetes/tree/master/pods>]

**Login or ssh in to master from kops machine**

**Curl** [**http://100.96.2.3:8080**](http://100.96.2.3:8080) **– to send http reqst**

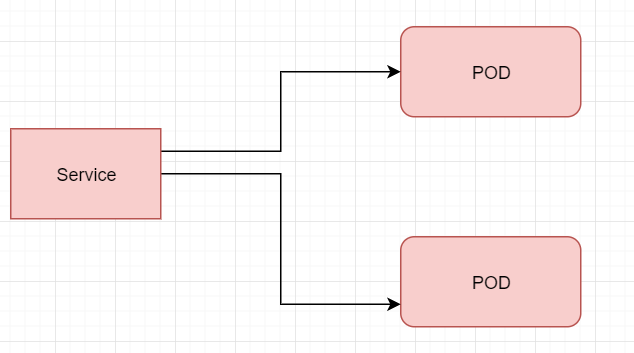
**I want to expose this pot to internet**

**Kubectl expose pods/nodeapp –type=”NodePort” --port8080**

**Kubectl get service**

After exposing a POD kubernetes creates a service object by allocating nodeport on every node ina cluster,

You hit any node(master or node),with node port the traffic is routed to service,and service routes traffic to POD.



Here service acts like single point of contact for specific app (micro services)