**TOMCAT**

Starting TOMCAT

Execute following script or startup.sh which is located under TOMCAT\_home/bin/

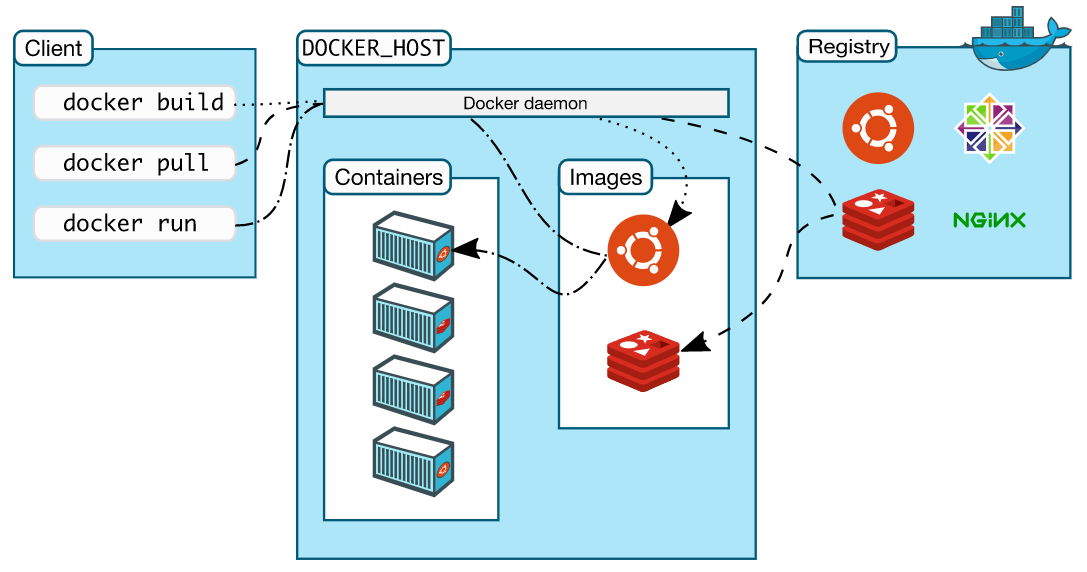
./startup.sh

To stop TOMCAT server

Execute ./shutdown.sh

**Date: 10-30-2019**

**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:** Is a server which will host docker images.

hub.docker.com- Docker has docker hub which is a registry available in internet.

Docker hub is hosted by Docker Inc. it is available for all.

**How to push images to docker hub?**

Step 1

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

Docker hub-id/image -name: tag

Login into docker hub

**cmd#** docker login

username:

password:

if we want to pull image from docker hub

#docker pull hub-id/image -name: tag

**Basic docker commands**

#docker images (to list images)

#docker rmi imageid or image name (remove image) to remove multiple images give space.

If an image is used by a container we cannot remove the image.

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

# docker rmi $(docker images -q) (to remove all images)

Containers

# docker run -it -p 8080:8080 sane/hari-app:1.0 (it means interactive terminal)

# docker ps (displays about running containers)

# docker ps -a (displays about run and stop containers)

# docker ps -aq (returns docker id’s)

# docker stop **container ID**

# docker start **container ID**

# docker restart **container ID**

# docker rm **container ID** (to remove container)  
Note: if container is running, we can’t remove but we can force remove.

# docker rm -f (force remove)

# docker rm -f $(docker ps -aq) to remove all containers

# docker run -d -p 1213:8080 –name=sanereddy/hari-myweb:1.0

Q. What is -P in docker run command?

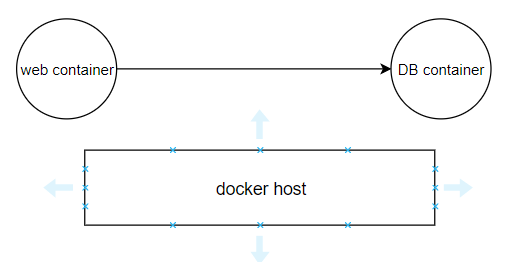
A. It publish all exposed port to random ports.

# docker run –rm -d -P –name=sanereddy sanereddy/hari -app:1.0 (remove container automatically when it stops)

**Docker Networking**

The objective of this discussion is how container talk to each other.

First, we will focus on how container will communicate when they are running on same host.



**Date: 01-11-2019**

We will see how container communicates when they are running on single host.

Earlier we had container links for connecting docker containers.

**Bridge network**

we have bridge network in two flavors.

1. **Default bridge:**
2. Default bridge is implicitly created.
3. By default, any container we create they join default bridge.

# docker network list (to list all network host)

# docker network inspect Network ID

1. Container joining default bridge can talk to each other by using IP address.

# docker exec -it container ID bash (to get into container terminal)

1. In default bridge communications through host names (container name) will not work.
2. **Custom bridge network**
3. We can create custom bridge network.
4. We can explicitly join containers to custom bridge network.
5. Communications through IP can container name.

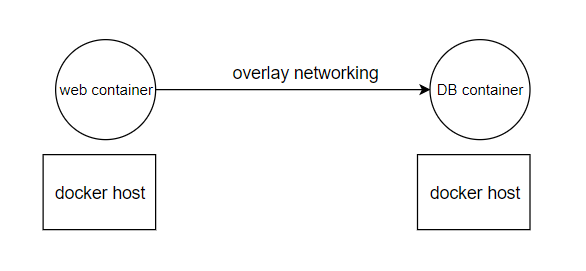
# docker network create network name (java home) --driver=bridge (to create custom bridge)

# docker run -d -network=java home -name=tomcattwo tomcat:8.0

**Overlay network**

Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other.

Containers are running two different host can talk to each other.



**Docker Volumes**

If docker container generates data on run time, that data is stored in with the container and that data is lost when container is deleted.

If we run databases inside a container, we lose data when container is removed. To solve this problem, we use volumes.

# docker exec -it container ID ash (alpine) (to get into container terminal)

# docker run -itd alpine

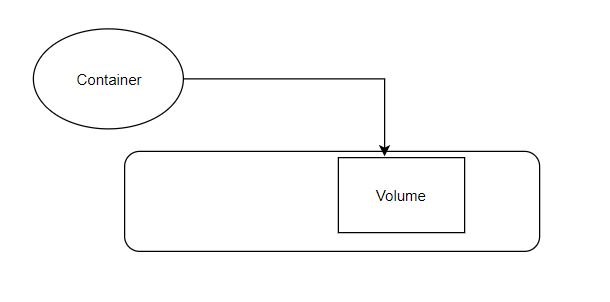
State full applications

Examples: Databases

Stateless applications

Any application which doesn’t generate data. Example web applications.

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



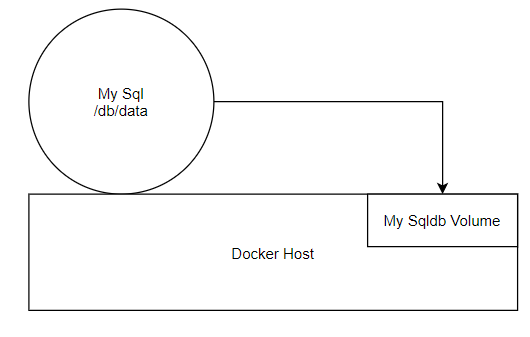
**Date: 04-11-2019**

# docker volume list (to display volumes)

# docker volume create mysqldb (to create volume)

# docker volume inspect mysqldb (to get data is stored)

# docker run -itd -v mysqlbd:/opt/data alpine (to mount the path of a volume)



# docker run -itd -v /home/ec2-user/chandureddy:/opt/data (we can create or change the folder)

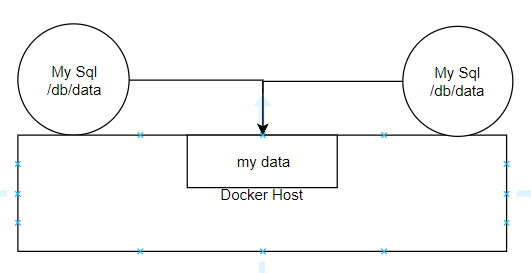
Assignment: <https://docs.docker.com/storage/volumes/>

Bind mount

Tmpfs mount

Q. how two containers running on a same host can share data?

A. By using volumes containers can share the data.



Q. Writing docker files or dockerizing applications.

A. in order to dockerizing any app we need to write docker.

To write docker file we must know docker instructions.

Q. Did you write docker files?

A. Yes.

Example

Create docker file and install python on it using alpine.

FROM alpine:3.10

LABEL AUTHOR=”Chandu reddy’’ (allows to add data information)

LABEL TIER=api

LABEL app=myapp

RUN apk add python3 (run is used to execute the commands in docker images at build time.)

WORKDIR /app (will create a folder if it doesn’t exist or uses if it exists. It makes the folder as present working directory.)

COPY app.py . (copies files from docker host to docker image)

EXPOSE 5000 (expose (container) port)

CMD [“python3’’, ‘’app.py’’] (is run time instruction i.e commands are executed commands when we use docker run.

Q. what is difference b/w run and CMD?

A. Run is built time instruction. CMD is run time instruction.

# docker run -d myapp:1.0 (built image)

ADD (instead of copy we use ADD)

For copy cmd source can be local host.

For add cmd source can be local host or remote URL

**Date: 05-11-2019**

Online docker application

Play with docker

Alt+enter – Maximize the session

Alt+enter – Minimize the session

CMD instructions can be overwritten at command level.

Entry point- It is a command similar to CMD. Entry point is a run time instruction. We cannot overwrite entry point instruction at command level.

If we try to overwrite entry point it appends entry point command inside the docker line with command at command line.

Using ENTRY POINT with CMD

Q. Difference b/w ENTRY POINT and CMD

A.

CMD and ENTRY POINT has two forms. Executable form and shell form.

Cat dockerfile

FROM alpine:latest

ENV APP\_ENV=prod

ENTRYPOINT [‘’echo’’,’’connecting to $APP\_ENV’’]

#exec form

CMD [‘’echo’’, ‘’hi $APP\_ENV’’]

# Shell form

CMD echo “hi $APP\_ENV’’

Exec form will not consider as a variable i.e., variable also printed as plain text instead of printing.

ENV Variables are helpful to pass arguments to our applications at run time.

# docker run -it -e APP ENV=prod javahome:1.0

ARG stands for arguments. At built time you want to change dynamically use ARG.

FROM alpine:latest

ARG PY\_VERSION=3

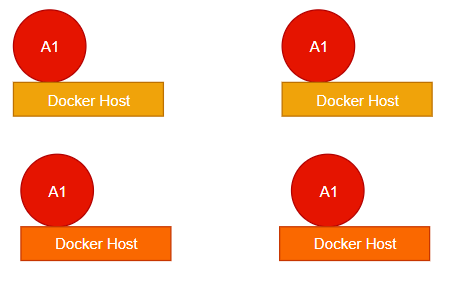
RUN apk add python${PY\_ VERSION}

Docker build –build-arg PY\_VERSION=3.5 -t javahome:1.0 .

Reference: <https://docs.docker.com/develop/develop-images/dockerfile_best-practices/>

**Date: 06-11-2019**

Running docker containers in production.



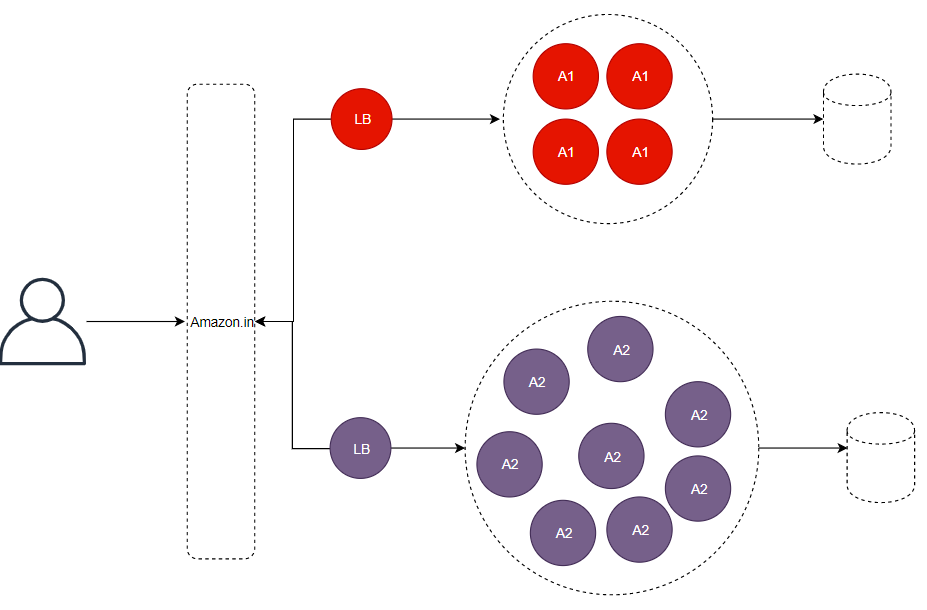
Monolithic: - A big application which is packed as single binaries or artifactory.

Problems with monolithic architecture.

1. Because of big code base making changes to the application takes more effort and time.
2. For new comers under the system takes lots of time.
3. We can’t scale only a specific module or a functionality.
4. We have to always stick to only technology standard.
5. If one application fails our complete application will be down.

**Micro services architecture**

1. Dividing monolithic application into smaller applications (Micro service).
2. Micro must be small enough such that two developer can manages.
3. It increases productivity i.e. developer doesn’t need to focus on complete application they need to focus on their micro service.
4. It provides cost effective scale. We scale a specific micro service rather scaling complete monolithic application.
5. Every micro service must have its own database.

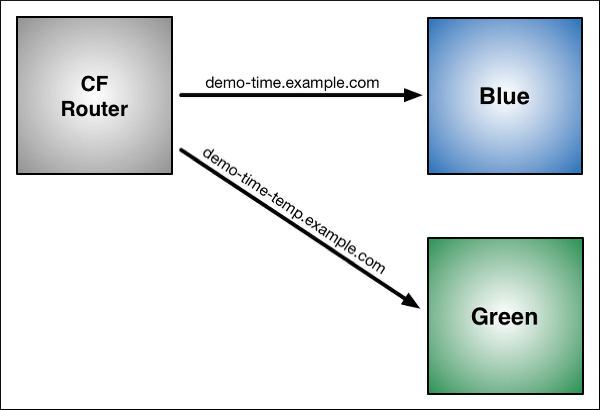


1. In GIT every micro service has its own repository.
2. Every micro service will have its own CI/CD process.
3. Micro services will talk each other using restful apps.
4. We can make deployment of micro services independently.

Q. Different deployment technics?

**Rolling updates:** for example, we have a micro service with 30 replicas we want to replace that. Instead of replacing all in shot (down time) replace them in batch wise i.e. replace 30% at a time. Repeat this all old containers replaces with new containers.

**Blue green deployments:** Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.



**Deploying containers in production**

Lot of activities comes into picture we manage everything on our own. So, we have to depends on third party docker orchestration frame work.

To run container in production we have to choose on the following container orchestrion tools.

1. Kubernetes (google frame work)
2. Docker swarm (docker native frame work)
3. Mesos (Apache)
4. Cloud – Amazon ECS (Elastic container service)
5. Amazon ECS (elastic container service)
6. Amazon EKS (elastic Kubernetes service)
7. Azure GKE (Google Kubernetes Engine)
8. Rancher labs
9. Open shift

**Date: 07-11-2019**

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

**Date: 08-11-2019**

**Setting up Kubernetes cluster**

There are several ways of setting Kubernetes cluster. We can run Kubernetes anywhere i.e. on our own physical servers, Virtual Machines and even on a cloud.

We could use **Kubeadm** to setting up Kubernetes cluster.

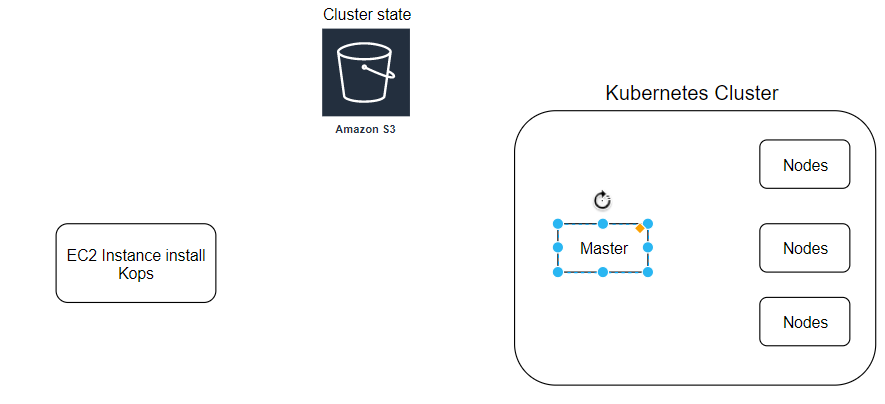
**Kubespray**

We could use **kops** to set up Kubernetes cluster on AWS.

**Setting up cluster using Kops:**

<https://github.com/javahometech/kubernetes>

etcd it is distributed key value store.



Networking: Kubernetes has its own frame work which allows PODS (containers) to communicate each other from any host.

Scheduling: it means finding nodes to run a container. When you deploy a PODS into Kubernetes. Kubernetes has to find the appropriate nodes to create 3 replicas.

Step 1:

Launch EC2 instance

Step 2:

Create IAM role

Kops need permissions to access

S3

EC2

VPC

Route53

Autoscaling

etc..

Step 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

KOP’s is used to build a cluster. (deploying services, delete cluster etc.,)

Step 4:

Install Kubectl

Kubectl is used to interact with cluster

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

Kops is used to build our cluster then we need Kubectl to interact with cluster.

Step 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

Created s3 bucket under my name **sanereddykops.in.k8s**

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

Step 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 private hosted zone for VPC
4. Select default vpc in the region you are setting up your cluster
5. Hit create

Step 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

Step 8:

Create ssh key pair

This keypair is used for ssh into kubernetes cluster

ssh-keygen

Step 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

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

Step 11:

**To connect to the master**

ssh admin@api.javahome.in

**Destroy the kubernetes cluster**

kops delete cluster --yes

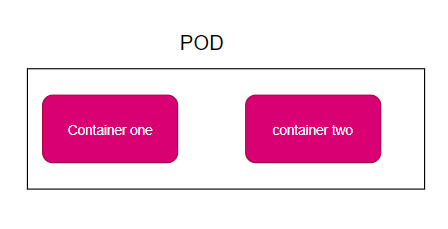
# Optional (Create terraform scripts through kops)

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

**Date: 11-11-2019**

POD is an atomic unit for deployment.

If you have a scenario where couples of containers need to be deployed together and to be destroyed together then keep it in a single POD.



Kubernetes PODs have IP address.

IP address of the POD can access anywhere from the cluster.

Containers on a same POD can talk to each other using local host.

Deploying our first POD inside a Kubernetes cluster.

We can request Kubernetes cluster to create a POD.

1. Using a command
2. Write YML document where we describe all details about our POD.
3. kubectl create -f https://raw.githubusercontent.com/javahometech/kubernetes/master/pods/pods.yml

### Command to get all pods

$ kubectl get pods

### Command to describe pod details

**Syntax** - kubectl describe pods/POD\_NAME

$ kubectl describe pods/nodeapp

By default, PODs are not exposed to the internet. KOPs are outside from cluster.

$ ssh [admin@api.javahome.in](mailto:admin@api.javahome.in)

Logging ssh into master

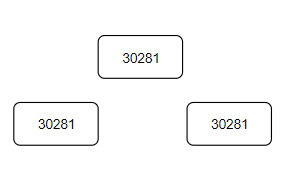
Curl <http://(pod> ip address):8080 (commands to send the http request)

**Exposing POD in the internet**

By default Pods run in a isolated environment i.e. they are reachable within kubernetes cluster, if you wanna reach your pod outside cluster, You have to expose it

$ kubectl expose pods/nodeapp --type="NodePort" --port 8080

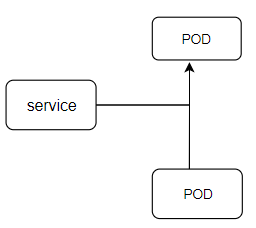
$ kubectl get service



After exposing a POD Kubernetes created a service object by allocating node POD on every node in the cluster.

You hit any node (master or node) with a node port the traffic is routed to a service. A service is routing a POD.

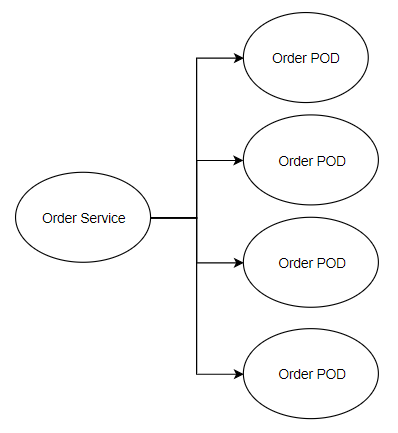
Servicse is acts like a single point of contact for a specific app (micro service)



**Date: 12-11-2019**

Kubernetes service:

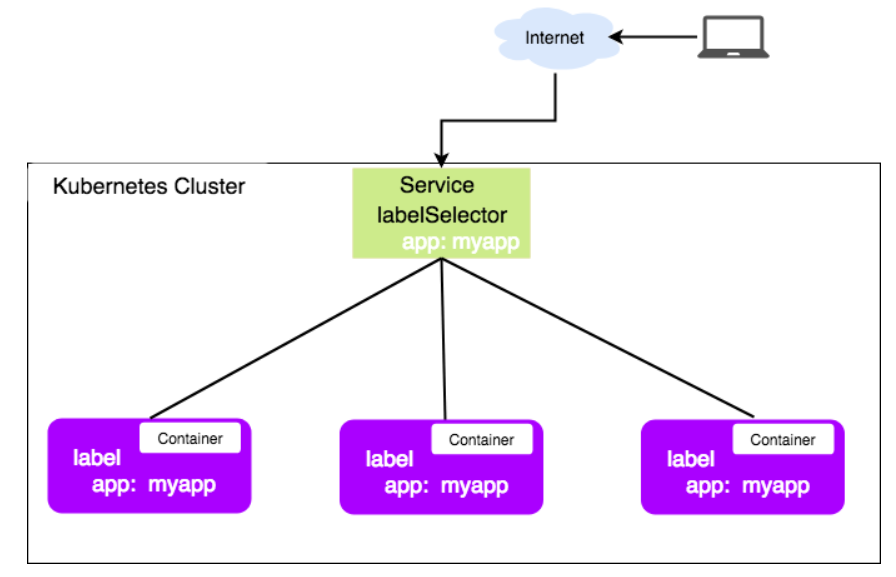
1. We use service to expose our PODs to the internet.
2. It also serves as a load balancer.



1. Service has also IP.
2. PODs are designed to be terminated.
3. Directly communicating with POD using IP is not recommended because changes are high that POD is re-created with different IP.
4. Always place a service object Infront of POD.

How service and its related POD are tied up.

* Kubernetes uses labels in order to map service objects with related PODs.
* Service can be single point of contact like load balancer for set of pods.
* By default, Pod is not expose to the internet or outside of the cluster, by using service Pod is exposed to internet.
* Any Pod create in future is added to service dynamically if pod label is matching with Service label selector.



# kubectl delete pod/nodeapp

# kubectl apply -f pods.yml

Service types:

* Node port
* Cluster IP

It will configure our service as a private.

Ex. A service for database must be provided.

* Load balancer

It creates cloud native load balancer this option works when Kubernetes cluster is on cloud.

* External Name

Maps the service to the contents of the external name field (e.g. foo.bar)

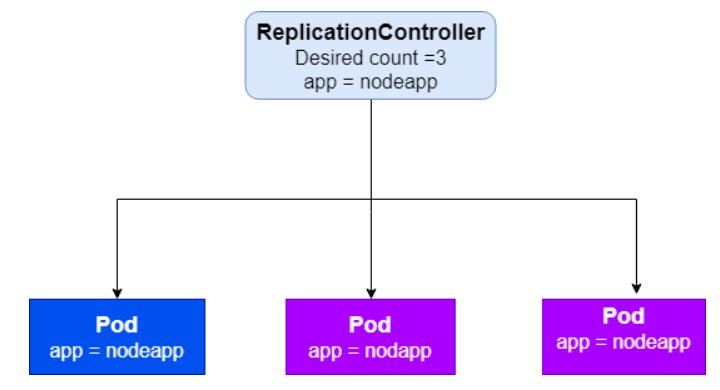
**Replication control:**

* Its job is maintaining specified numbers of replicas in the cluster.
* If number of POD is less than desired replicas it creates. If PODs are more than desired replicas it deletes them.

Reference: <https://github.com/javahometech/kubernetes/tree/master/ReplicationController>

* It replaces if any one of pod is deleted or failed.
* ReplicationController supervises multiple pods across multiple nodes
* ReplicationController is often abbreviated to **"rc"** or **"rcs"** in discussion, and as a shortcut in kubectl commands.

**EX:-** If pods are re-created on a node after disruptive maintenance such as a kernel upgrade. for this reason, use ReplicationController if if your application require only single-pod.



**Date: 13-11-2019**

**ReplicaSet:**

Replica set is an extension to replication control. Replica set supports set based selector.

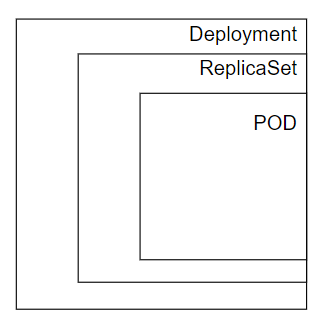
Q. How do you manages secrets in Kubernetes.

A. We have to use secret object.

# kubectl create -f secrets.yml

**Deployments:**

Deployment objects has rolling updates and rollbacks.



Deployment objects by default use rolling strategies.

**Liveness and Readiness Probes:**

The kubelet uses liveness probes to know when to restart a Container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a Container in such a state can help to make the application more available despite bugs.

Q. How to do you package and deploy your applications into Kubernetes.

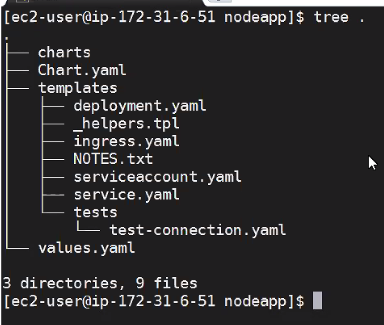
A. We use helm charts.

**Helm Charts**

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Reference: <https://github.com/javahometech/helm>

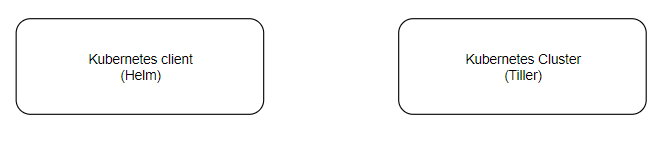
**Structure of Helm chart**



Helm charts is implemented by using golang.

**Date: 14-11-2019**

RBAC: roll based access control.

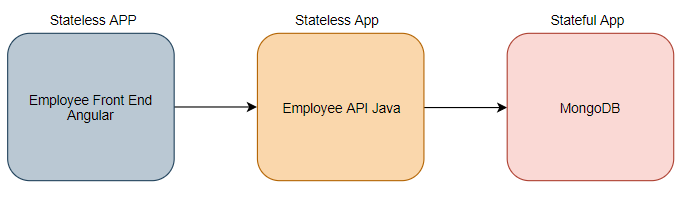


Helm is a utility run from client side.

Tiller is a path that run in Kubernetes cluster.

**Kubernetes architecture**:

Deploying 3 tier application into Kubernetes?



One microservice can talk to another microservices using service mail if they are in same **name space**.

**Date: 15-11-2019**