Docker

To run any application / software we need to install O.S first.

Installation of O.S known as provisioning.

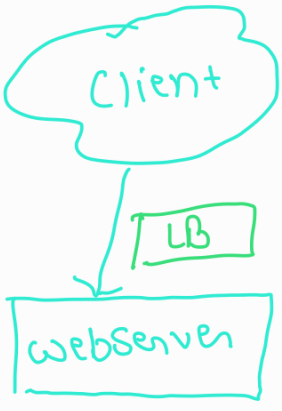
→ Four ways of provisioning

(i) Bare metal — 60 min

(ii) cloud computing — 2-5 min (Paid Service)

(iii) Virtualization → Oracle VM, virtualbox etc. 30-40min

(iv) Docker — commonly known as containerization.



* If load increased, then we have to launch one more new webserver. For this we have load-balancer.

Now how fast a new webserver can be launched matters a lot.

→

→ By containerization we can setup full fledge webserver in 1 sec.

Useful commands

docker ps → List all the running container in docker.

docker ps -a → List all the container. (running + Stopped)

To install O.S, we require image
eg → .iso file, windows Dvd etc

Hey - in docker also to launch O.S/container we will require image.

docker images → list all the available images locally.

docker pull centos:7 → download the centos image from docker public repository.

→ hub.docker.com

docker run -it centos:7

↳ interactive terminal

↳ new os/container will be launched from Centos image.

docker run -it ubuntu:14.04

docker ps -a ↳ All the history of container.

docker run -it --name Kishanos centos:7

↳ we can give the name from ourselves.

docker Ps

docker Stop Kishanos

docker Start Kishanos

docker attach Kishanos

Old Commands	New commands

(Management commands)

docker ps

docker container ps

docker start

docker container start.

docker container --help

→ looks clean.

docker container ls -a -q

→ it will give id of all container.

docker rm -f $(docker container ls -a -q)

display numeric id

→ docker container run -it --name OS1 centos:7

For coming out — (i) exit {Stop and exit}

(ii) Ctrl+P+q { directly exit without

stopping container ?

* If you have a requirement to run a program then terminate the O.S

docker run -it --name z1 centos:7 date

→ will run the date command then it will stop the container automatically.

* Create a custom image as per our requirement.

→ Two ways

(i) commit

(ii) Dockerfile

Commit

docker run -it --name myos1 centos:7

ifconfig - not found — yum whatprovides ifconfig

wget -not found

yum install net-tools

yum install wget

exit

→ docker commit myos1 mynewos:v1

→ docker images

→ docker run -it --name os2 mynewos:v1

↳ ifconfig

↳ wget

→ docker run -it --rm myimage:v1 ifconfig

→ docker inspect <container name>

↳ detailed info about container

Dockerfile

mkdir /ws

vi Dockerfile

↳ Pre defined file name for

custom image.

→ FROM centos:7
RUN yum install python36 -y

:wq

→ docker build -t testpython:v1 /ws

* will create a image having python already installed.

New Requirement → As soon as we start our container, python should start.

Vi Dockerfile

→ FROM centos
RUN yum install python36 -y
CMD Python3

→ docker build -t testpython:v2 /ws/

→ docker run -it --name=c2 testpython:v2

docker run -it os2 /usr/bin/python3.02

↳ directly python terminal will start.

=

(i) we can download image easily

centos:7 ↳ 200MB

ubuntu ↳ 100MB

(ii) docker run -it centos:7

—]

Kubernetes

→ In docker we saw that we run our application / microservices inside docker container because docker is faster in provisioning the O.S

→ Now if container crashes, then our application will be down.

→ To make our Application again running we have to manually again restart our container.

→ As a human being it's impossible for us to keep monitoring all the

containers all the time and keep restarting it.

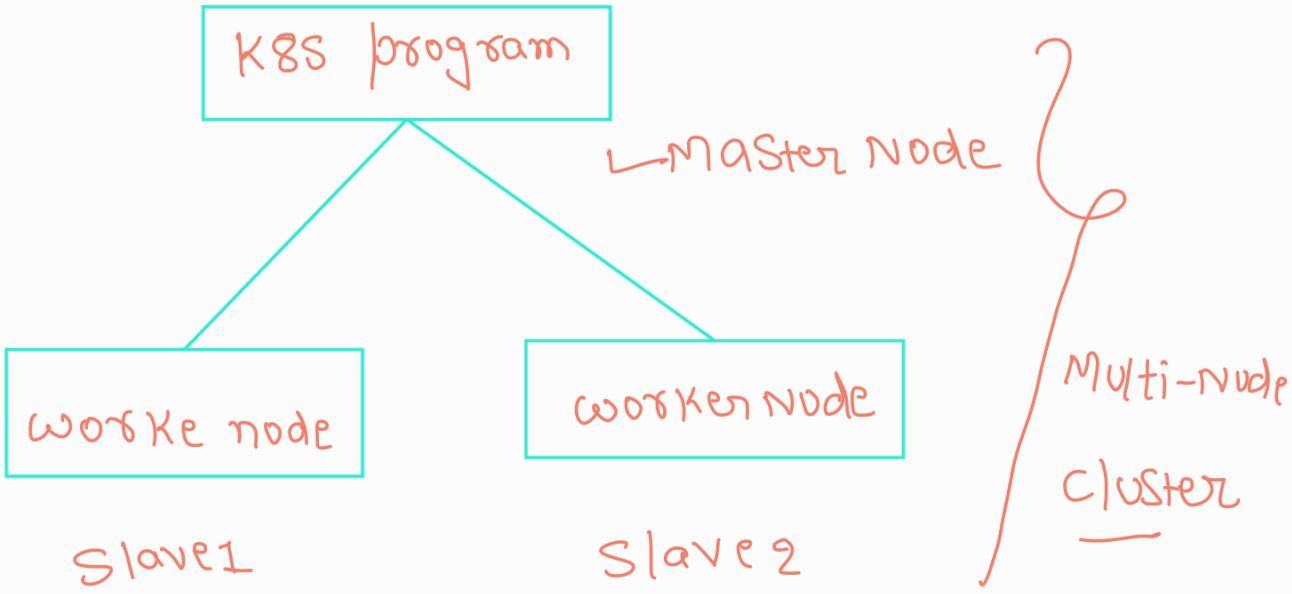
→ So, for doing this work we hire a manager who will manage containers on our behalf.
↳ Just a program / software

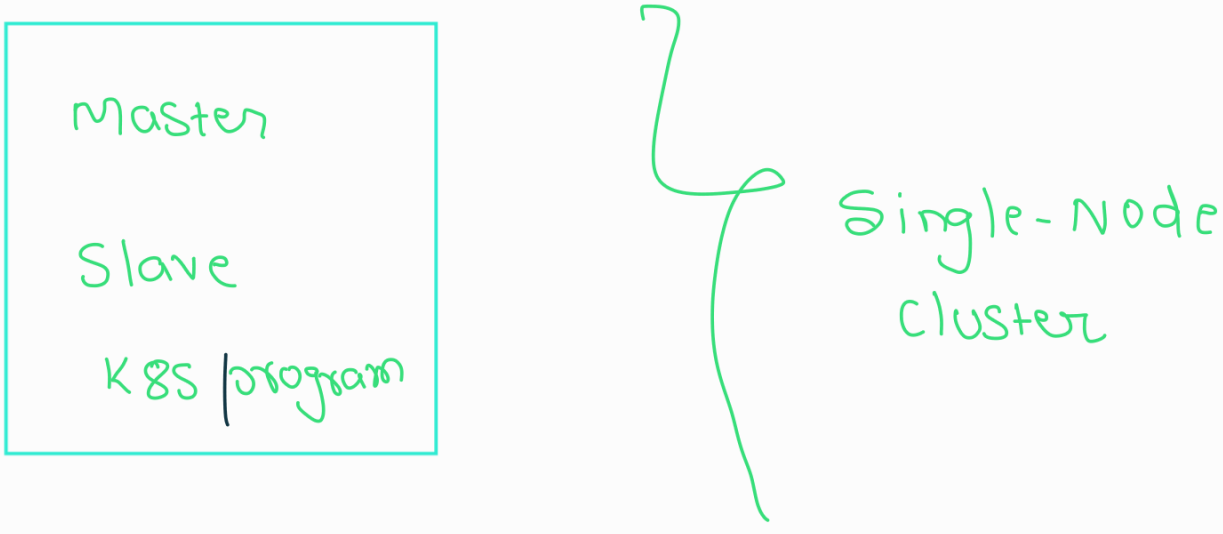
→ So, we tell our program to keep monitoring our container, if it will go down either restart it or create new with same connectivity permission.

→ Kubernetes is one of the manager program.

→ Docker Swarm is other tool which provides this management feature but K8s is more mature.

↳ K8s can manage multiple containers.

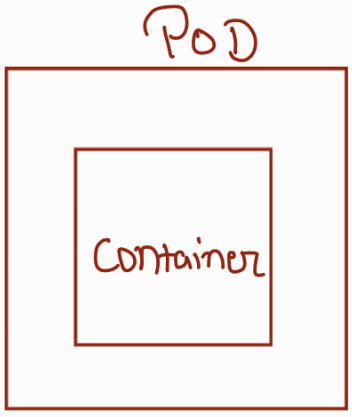




↳ Kubernetes is a controller program whose duty is to measure/monitor container.

Smallest unit of docker — container

Smallest unit of Kubernetes — POD



→ Kubernetes always manages pod not container.

→ For deploying Application we require infrastructure, and this infrastructure we get from docker.

→ If we started a webserver container and given access to client, but somehow that container get removed or crashed then client will not be able to access webserver.

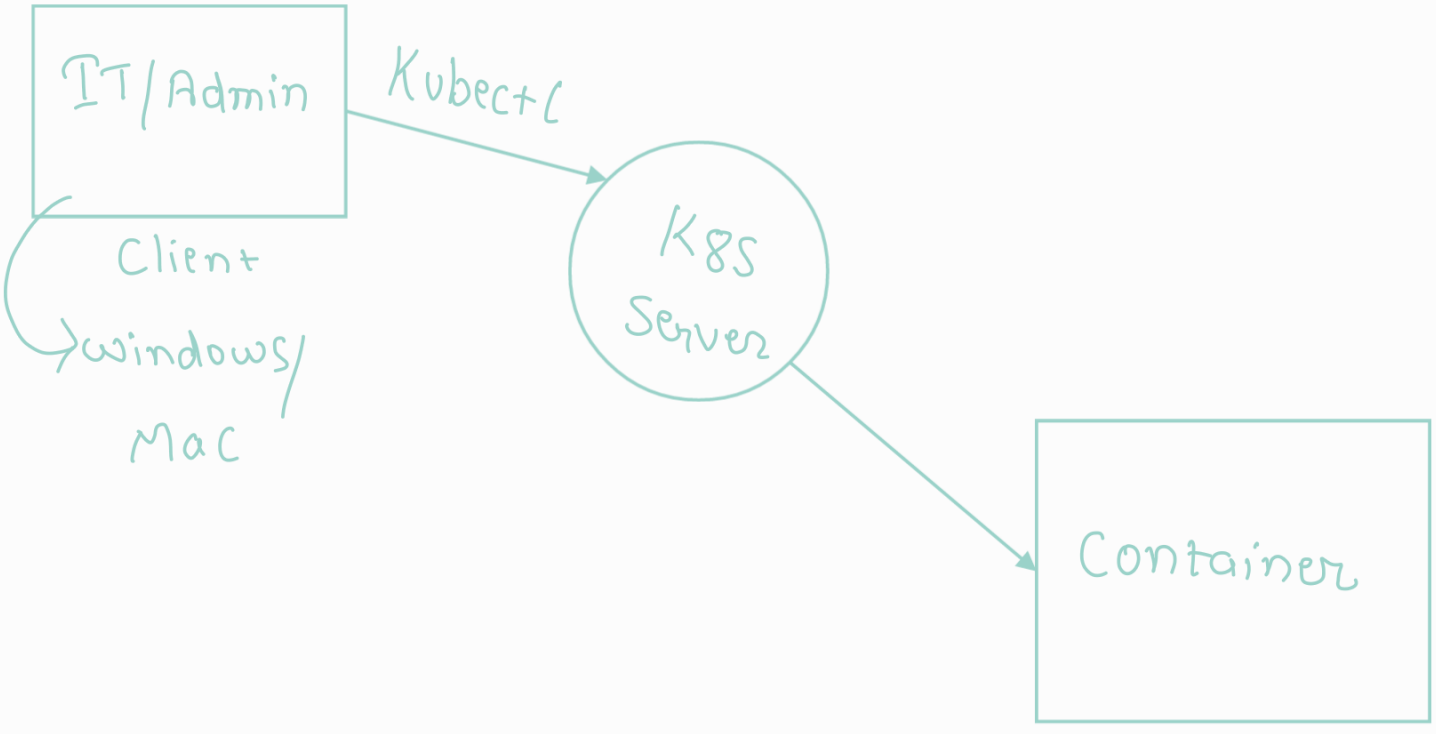
→ So we want K8s to keep monitoring our container.

↳ known as container

Management tool.

* We will use minikube to provision Kubernetes cluster in our O.S

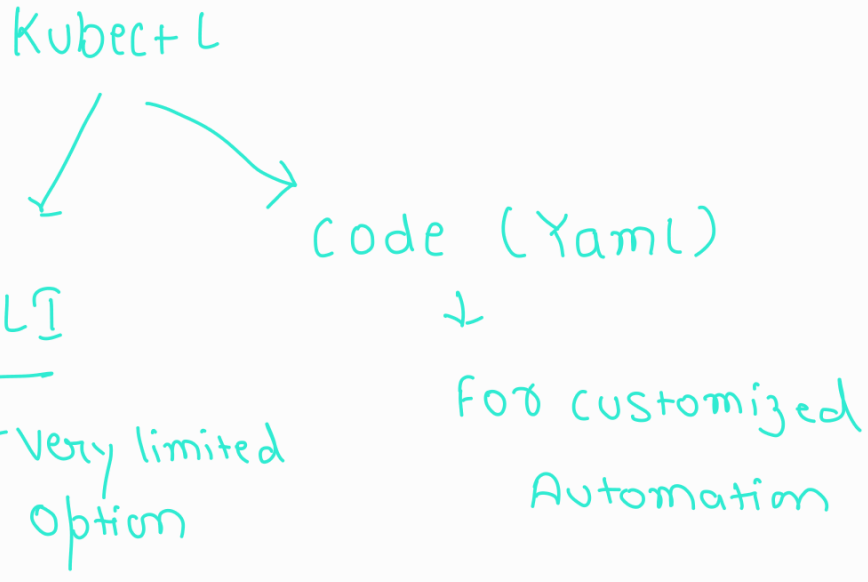
* Once minikube will be setup then we can use "Kubectl" command to interact with Kubernetes which will further interact to container.



→ Kubectl get pods ≃ docker ps

↳

Will show the list of running pods, same as "docker ps"
Show the list of running container.



Pod.yaml

```
apiVersion: v1
  kind : Pod
 metadata:
      name: "mypod"
       labels:
           app: web
  Spec:
```

```
Containers:
    name: "myc1"
    image: "roykishan8/webserver"
```

→ Kubectl apply -f pod.yaml

Deployment → we always launch our pod with the help of deployment.

- Deployment is just act as a manager for pods which will keep monitoring the state of pod everytime.

↳ To monitor desired number of replicas of pod deployment uses ReplicaSet.

To implement load Balancer in Kubernetes we have to use Service.

Service

- **ClusterIp**
  - You can only access this Service while inside the cluster.
  - → Kubernetes resources uses clusterIP Service to communicate to each other internally.
  - (targetPort)
  - Port
- **NodePort**
  - Exposes the Service on each Node ip. (NodePort)
  - targetPort
  - Port
- **External (Load Balancer)**
  - Exposes the Service externally using a cloud provider's load Balancer.

Service.yaml

apiVersion: v1

Kind: Service

metadata:

name: mylb1

Spec:

type: NodePort

Selector:

app: web

Ports:

- targetPort: 80

Port: 8080

NodePort: 30000 { 30000 - 32768 }