# Container management tools

It **controls** the containers like docker.

# Container management type

Kubernetes, D container, Rocket, **Docker swarm,** apache marathon

# Kuberneters

* It helps to automates **container deployment, load balncing, scaling, capability , highly availability**
* It schedule runs, **crons jobs, and** manges isolated conatiners which are running on virtual /physical/cloud
* Written in Golang by google
* Now handled by CNCF

## Online plateform

* Play with K8S
* Play with Kyberneters classroom
* Kubernetes playground

## Cloud based k8s services

* GKE, AKS, Amazon EKS(Elastic KS)

## Kubernetes installation tool

* Minicube
* Kubeadm

## Problems without cmt

* Containers **can not communicate**
* Autoscalling and **load balancing not possible**
* Every time more careful about the containers
* We can overcome this with few solution mentioned below in the doc

# Features of K8S

* Orchestration(clusting of any no. of the containers)
* Autoscaling

1. vertically scaling 🡪 2GB to 8 GB, no new hiring only use existing employee for the work
2. horizental scaling(**preferable**) 🡪 create one more containers, hiring new employee

* Autohealing
* Load Balancing
* Plateform independent
* Fault Tolerance (**node/pod failure**)
* Rollback
* Health monitroing
* Batch execution

## Kubernetes benefits

* GUI is available
* Inbuilt tool present for monitoring
* Autoscaling is available
* Data volumes only shared with containers pods.
  + Pod 🡪 smallest and basic unit; like containers in docker.
* Kubernetes not directly handle the containers, only deal with pod.
* Logging and monitoring by inbuilt tool

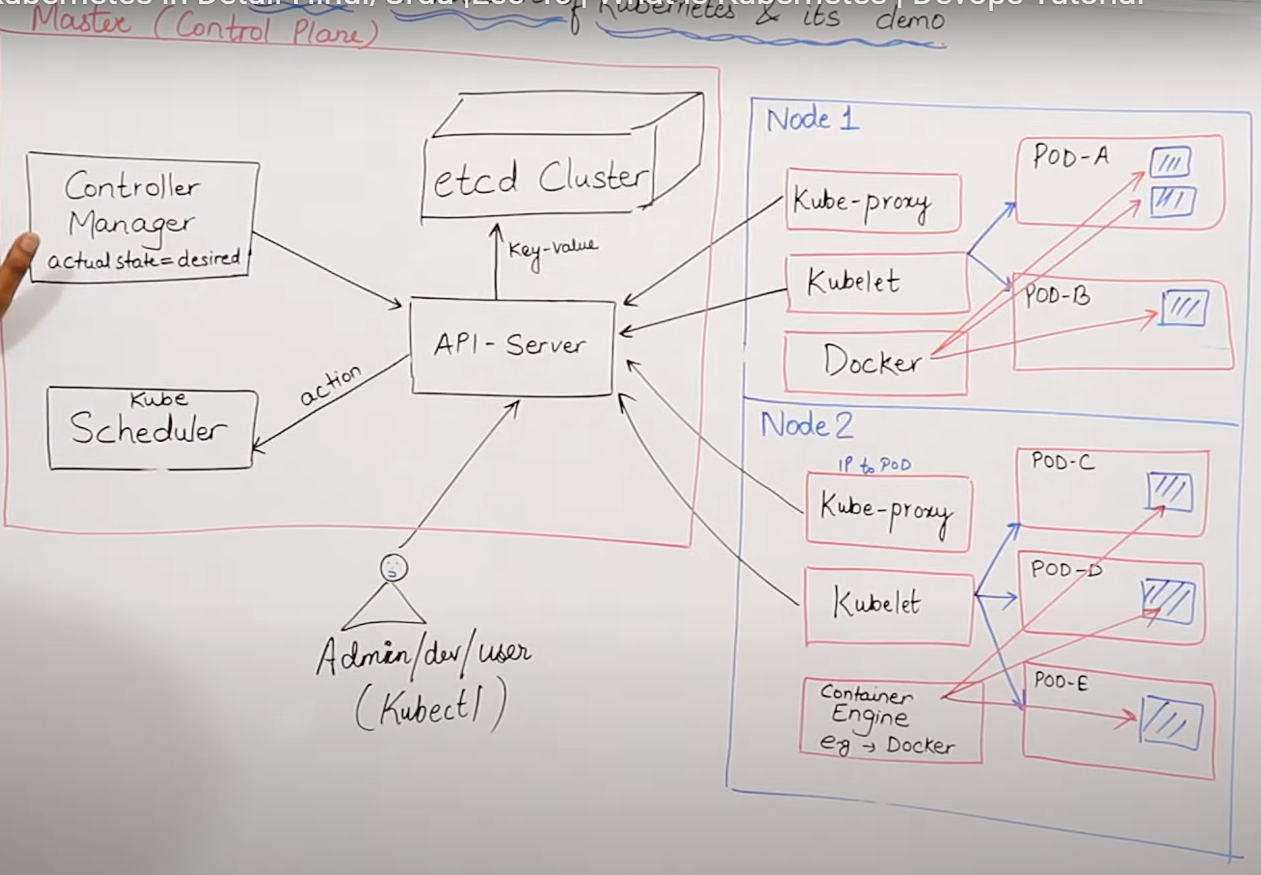
## Docker swarm benefits

* GUI is not available
* Not autoscaling is available
* Can shared data volumes with any other containers
* Logging and monitoring by 3rd party tool like splunk.

## Master slave architexture

1. Master(server) – kubernetes api
2. Multiple master also posible
3. Pod is part of node
4. **One node can have multiple pod**
5. **One pod can have multiple container**
6. Node(client server) **– k8s communicate with the pod**

# K8S architecture



## Master control panel

* API server : like receptionist
* Controller manager : like controller in api, guranttee of required work
* Etcd cluster(**not part of k8s**) : database like zookeeper, ohai in chef , key-value
* Kube schuduler
* Kubectl
  1. JSON/YAML file
  2. Admin/dev/user
* container engine like docker

## Worker

* Node
* Kube-proxy : assign ip to pod
* Kublet(khabri) it control pod by sending request to api server
* container engine like docker
* **Container**

1. It does not have any IP only pod have IP
2. container tightly coupled
3. **once fail it will fail all container in it**
4. one pod one container is fine approach
5. failed pod can not be repaired
6. always create new pod and will get new IP

* Pod

1. atomic unit like cell in human
2. **K8S does not have container on node but on pod**
3. K8S only know pod not container
4. Can have more thn one container
5. Once failed it is useless

# Master component

## Manifest

* Yml or json

## Kube api server

* It is kind of the front end server
* It can handle automatically manifiest request

## Etcd

* Store metadata, Fully replicated, Secure, Fast , high available store(key-client)

## Kube schuduler

* It handle pod creation and management reqeust

## Controller manager

* Try to make actual and desired state equal
* Component
* Node controller : detect the node and check how are newly created, respoding

Two package:

* Kube controller if non cloud
* Cloud controller if cloud
* Route controller : route the network
* Service controller : load balancer
* Volume controller

# Slave component

## Kubelet

* It contact with api server for the request and response
* Listens to kubernetes,
* 1025 pod number it uses.
* Success and fail report to the master

## Container engine

* Pulls images
* Start and stop containers
* Exposing containers on the ports in manifest
* With the kublets

## Kube proxy

* Assign ip to each pod(dynamic on new creation)

## Pods

* Smallest unit in kubernetes
* It wrappe the container
* K8s controls only pods not the pods
* Tightly coupled container
* No by default auto healing and auto scaling of pods
* No recovery of the pods

## Higher level pods

These features give extra strength to the k8s:

* Replication set : auto scaling and auto healing
* Deployment versioning and rollback
* Service static ip and networking
* Volume non epherimal storage means outside the containers, on fails of pod it does not lost.

# Setup kubernetes master and node on AWS

One master 2 node

## Commands for master

* Sudo su
* Apt-get update

### Install https packages

* apt-get install apt-transport-https
* Httpd is used for intra cluster communication
* Sudo apt install docker.io -y

### Check docker

* apt install docker.io -y
* docker --version
* systemctl start docker
* systemctl enable docker

### Setup GPG key

* It is required for intra cluster communication
* It will be added to source key on this node
* K8 sends signed info to out host
* Key will be on both and is going to accept communication
* Exit : ctl + x 🡪 ctl + y 🡪 enter
* --------------------------------------------------------------------
* sudo curl -s https://packages.cloud.google.com/apt... | sudo apt-key add
* nano /etc/apt/sources.list.d/kubernetes.list
* deb http://apt.kubernetes.io/ kubernetes-xenial main
* apt-get update
* apt-get install -y kubelet kubeadm kubectl kubernetes-cni

## Bootstrapping the master

* kubeadm init
* COPY THE COMMAND TO RUN IN NODES & SAVE IN NOTEPAD
* mkdir -p $HOME/.kube
* cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
* chown $(id -u):$(id -g) $HOME/.kube/config

### Deploy flanned

* kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
* kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml>

## Configure worker node (in master)

* COPY LONG CODE PROVIDED MY MASTER IN NODE NOW LIKE CODE GIVEN BELOW
* kubectl get nodes

e.g- kubeadm join 172.31.6.165:6443 --token kl9fhu.co2n90v3rxtqllrs --discovery-token-ca-cert-hash sha256:b0f8003d23dbf445e0132a53d7aa1922bdef8d553d9eca06e65c928322b3e7c0

# Deployment and Rollback

Replication controller and replica set is not able to do updates and rollback apps in the cluster.

RC and RS always create new pod not rollback failed pod.

* Deployments object act as supervisor.
* Deployment will never talk directly with pods.
* DO work is roll out, roll back, roll update.
* This provides self healing mechanism to address machine failure and maintance.
* It will maintain repllica set as version
* We can scale up and down
* Pause the deployment to multiple fixes to its pod template specification, and then resume as per requirement.
* If there is issue in the deployment it will automatically rollback to previous version
* Kubectl rollout undo deploy/mydeployments --to-revision=2
* Kubectl get deploy

## Inspect deployment

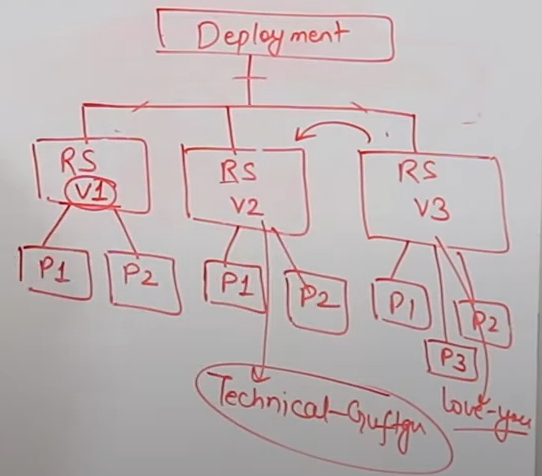
* Kubectl get deploy
* Name : mydeployment
* READY : how many replica are available
* Update to date : display number of replica
* Available : how many are replica are
* AGE : time

Propertional scaling?

Deployment 🡪 RS 🡪 POD

On rollback number of pods will remain same.

Like in v3 we have 3 pods on rollback to v2 pods will remain 3 not 2.



## Installation

COMMANDS USED IN THIS VIDEO

sudo su

#command to install docker is

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

#install Kubectl now with the given link

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

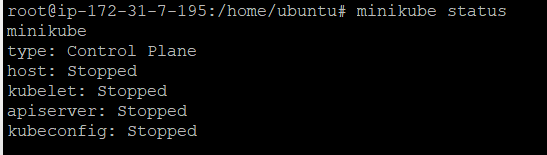
#install Minikube with the given link

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

apt install conntrack

minikube start --vm-driver=none

minikube status



kubectl version

kubectl get nodes

## YAML file

kind: Deployment

apiVersion: apps/v1

metadata:

name: mydeployments

spec:

replicas: 2 #create two pods

selector: # tell controller which pods to watch /belongs to

matchLabels:

name: deployment

template:

metadata:

name: testpod

labels:

name: deployment

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5; done"]

# liveprob:

# K8s name space:

kubectl get pods : default namespace

* If no resource then we will get no resource found

kubectl get pods **-n abc**

* It command will check pods in abc namespace.

kubectl get namespaces:

* Four namespaces by default…

kubectl config set-context $(kube-ctl config current-context) –namespace=dev

* set namespace as dev

kubectl get pods

* check in default namesapce

kubectl get pods -n dev

* fetch all pods form namespace

-------

* We can allocate memory to namespace.
* All resource within that namespace will only be formed within that memory
* We can give name to our object. But it is difficult to manage so we will give namespace.
* A scope for every names.
* A machanism to attach policy to subsection of the cluster.
* Most resources such pods, services, replication controllers and other are in same namespace
* Low level resources such as node and presistent volumen are not in any namespace( they are at cluster level).

apiversion: v1

kind:namespace

metadata:

name: dev

labels:

name: dev

Config and secrets

It is a object

To check deployment was created or not

* Kubectl get deploy

To check how deployment creates RS & pods

* Kubectl describe deploy mydeployments
* kubectl get rs

To scale up or scale down

* Kubectl scale –replicas =1 deploy mydeployments

To check wht is running inside container

* Kubectl logs -f <podname>

History and version

* Kubectl rollout history deployment mydeployments

Undo: one step back

* Kubectl rollout undo deploy/mydeployments