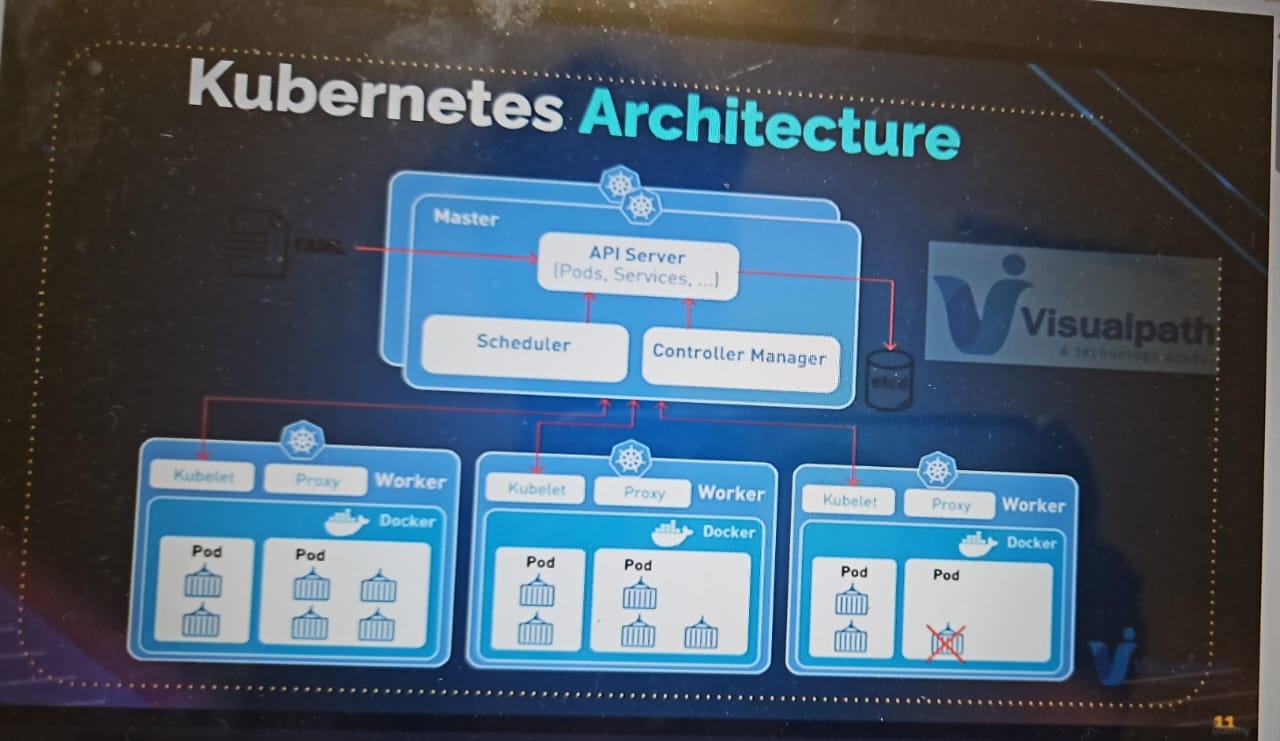
Kubernetes

Most popular containerization tool

Orchestration tools—docker swarm, Kubernetes, mesosphere marathon, AWS ECS & EKS, azure container service, google container service, CoreOS fleet, OpenShift

Created by GOOGLE to manage their containers AKA Borg



Master(control-plane): responsible for controlling kubernetes cluster,orchestrating workloads.

Kube API Server acts as front-end for the kubernetes control plane. Handling RESTful requests.

Scheduler assigns workloads to nodes based on resource availability and constraints.

Controller manager runs various controllers to maintain cluster’s state, such as node and job controllers.

Worker Nodes: is where actual application workload runs

hosts and manages containers using kubernetes componenets

Kubelet ensures containers are running on a pods

Kube-proxy manages network rules for commn b/w pods and services.

Container runtime(docker engine) runs the containers

Minikube:

Tool used to setup single node cluster(if master n worker runs on same machine) on K8’s

Tool lets you run kubernetes locally, tuns on a singal node kubernetes cluster on your personal computer(win, macos, linux pcs) so that you can try that out kubernetes.

Setup with Minikube

Ec2 instanve t2.med 20gb

Update server

Docker install

Install minkube, kubectl

Start minikube

Virtualbox(VM manager), container, hyperkit, hyper-v. KVM,parallels, podman

[ cd /c

git clone https://github.com/devopshydclub/vprofile-project.git

cd vprofile-project/

git checkout master

git pull

clear

git checkout kubernetes-setup

ls

cd minikube/

ls

cat Minikube-commands.txt

]

Open powershell as admin

Setup chocolatey

Install minikube with chocolatey

choco install minikube kubernetes-cli -y

Open powershell again and run

minikube start --no-vtx-check

kubectl get nodes

Create a sample deployment and expose it on port 8080:

kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0

kubectl expose deployment hello-minikube --type=NodePort --port=8080

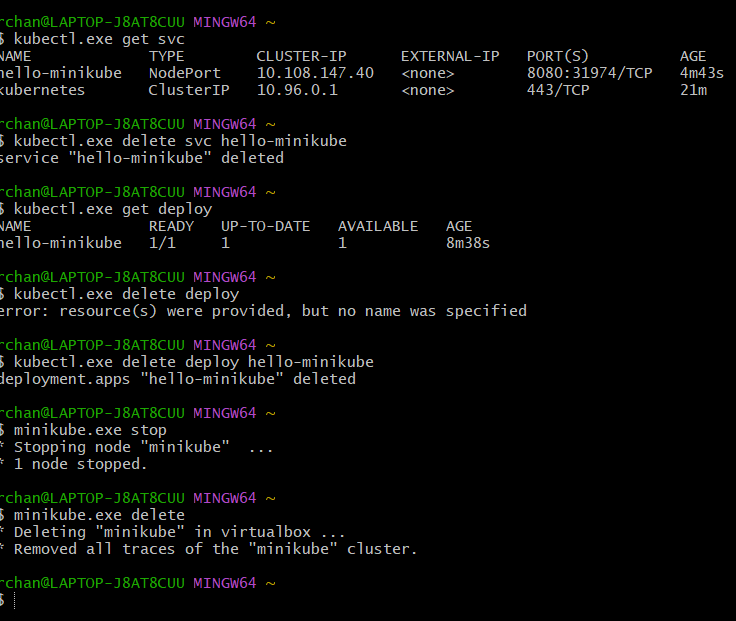
It may take a moment, but your deployment will soon show up when you run:

kubectl get services hello-minikube

The easiest way to access this service is to let minikube launch a web browser for you:

minikube service hello-minikube

To delete minikube VM:



Open virtualbox and you see no minikube VM

Kubectl:

Command line tool used to interact with kubernetes clusters,it communicates with kubernetes API server to manage n inspect cluster resources.

Setup kubectl

curl -LO [https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

K8s objects:

Pod

Service

Replica set

Deployment

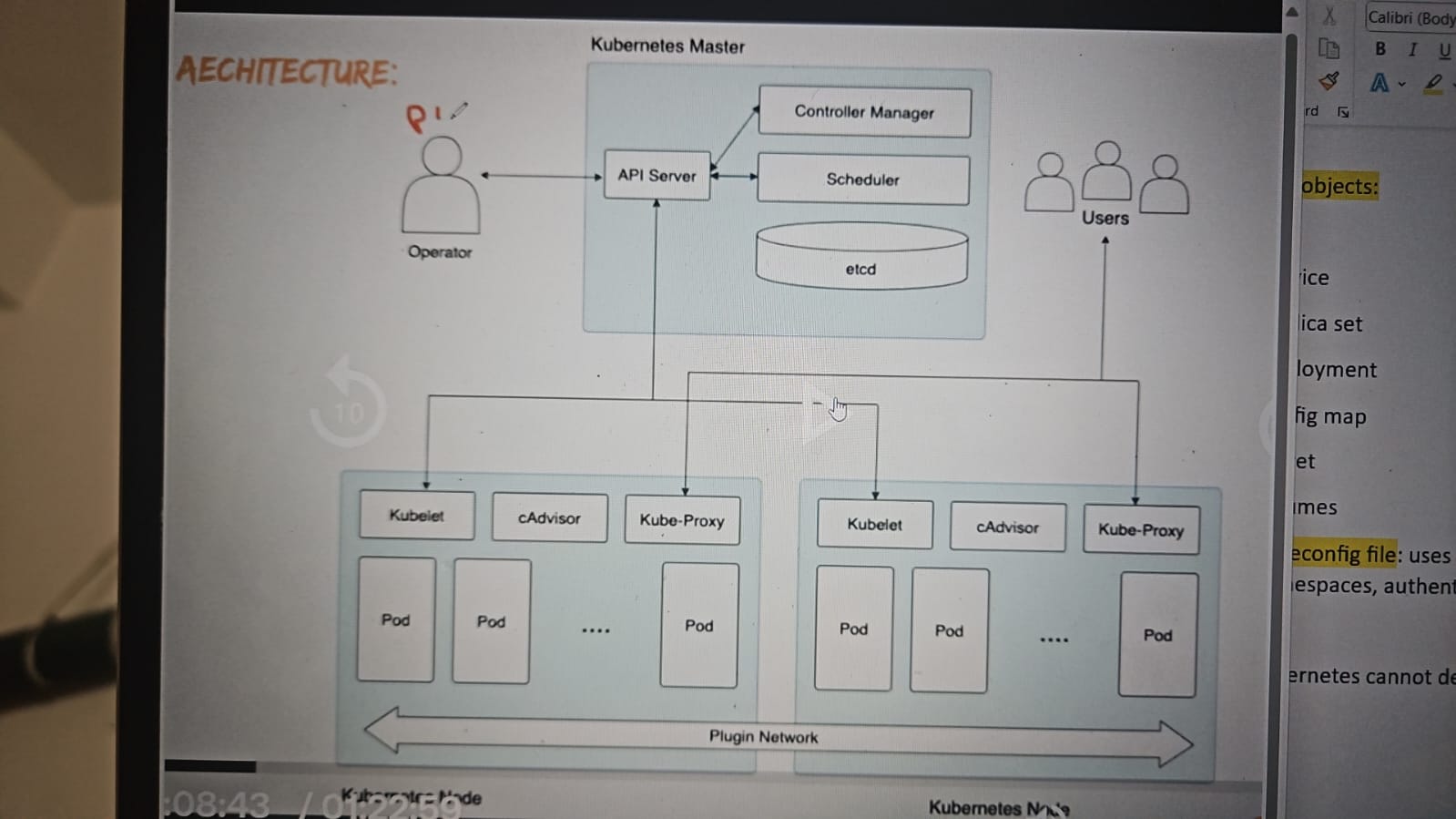
Config map

Secret

Volumes

Kubeconfig file: uses to organize information about clusters, users, namespaces, authentication mechanisms

Kubernetes cannot deal directly with containers, containers are inside the pod



REQUESTS are sent to api server and stored in etcd(database for cluster) in key-value format.

api server creates pods

cluster is grp of servers

scheduler checks either there are any pending tasks or not, decides in which worker node pod is gng to be created

kubelet passes information about no of pods to scheduler via api server

Controller manager: 3types—

pod controller – technically we call it as replication ctrller,

monitors desired no of pods are running or not

if a pod deletes then RC will created new pods

Node controller will check the nodes for every 5seconds through API server.

if node ctroller does not get any response from any node for 40sec, then it marks that as unreachable.

If worker server deletes, it creates new

service controller if any service is delete, automatically another one is deleted

2 types of kubernetes cluster:

Self managed K8’s cluster – minikube(single node cluster), kubeadmin(multi node cluster), KOPs

Cloud managed K8’s cluster – AWS EKS, Azure AKS, GCP GKS, IBM IKE

pods: grp of containers

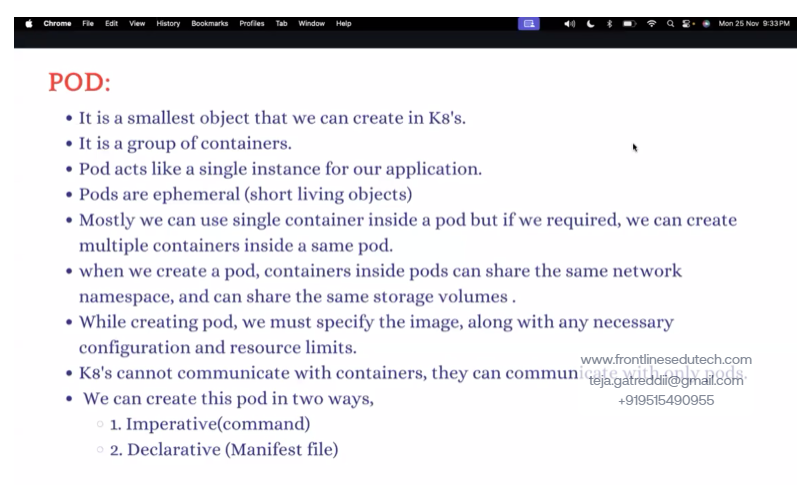
basic execution unit of kuberenetes application, smallest and simplest unit in kubernetes object model that you create or deploy

kubernetes manages the pods rather than the containers directly.

one-container-per-pod model is most commonly used use-case.

multiple container-pod – 1main container, other as a sidecar or init container

each pod is meant to run a singl einstance of a given application



imperative way – using command

kubectl run pod-2 –image=nginx

kubectl get pods/pod/po

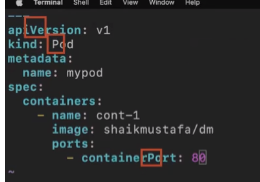
to see list of resources in kubectl

kubectl api-resources

declarative way – manifest file

vim flm.yml

apiVersion: v1 (V is capital), we get to know its v1 or not using kubectl api-resources



->to run pod

kubectl create -f flm.yml

->to check pods

kubectl get po

->to check particular pod

kubectl get po mypod

->if you want full info abt that pod in wide format

kubectl get po -o mypod wide

->if you want get it in yaml format

kubectl get po -o mypod yaml

->to know full info abt pod

kubectl describe pod mypod

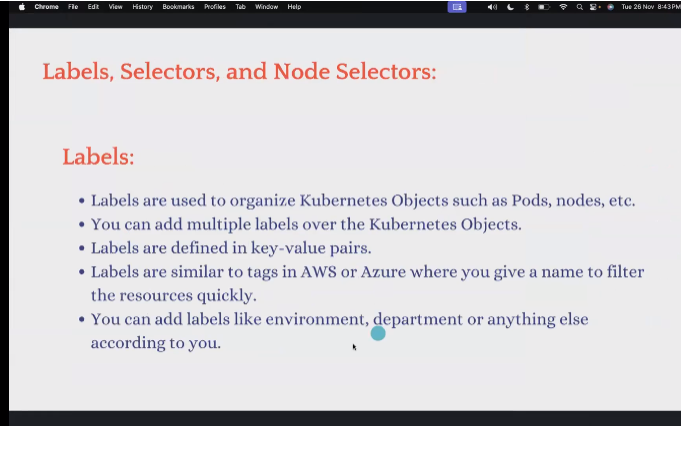
->to delete particular pod

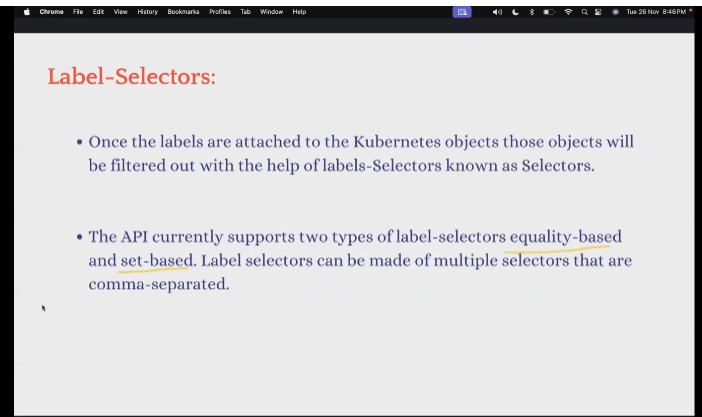
kubectl delete pod pod-1

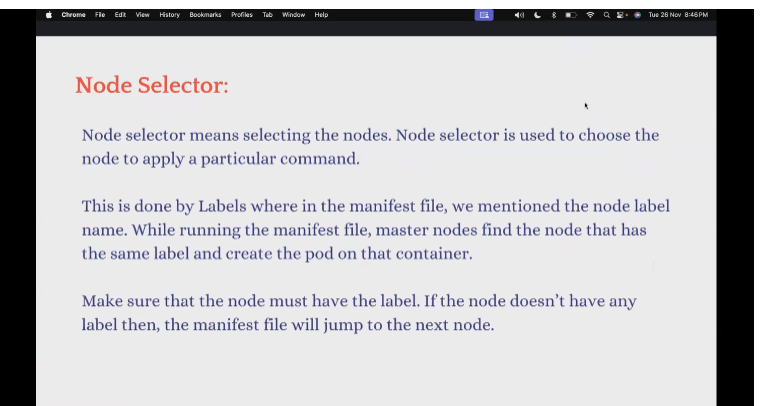
->to delete all pods

kubectl delete pod --all

Labels







Kops: kuberenetes operations is a command-line tool that simplifies the management of kubernetes clusters.

Setup with kops

Domain for kubernetes DNS records

Eg: groophy.in from GoDaddy

Create a linux VM and setup

Eg: Kops, kubectl, ssh keys, awscli

Login to AWS accnt and setup

S3 bucket, IAM user for AWScli, route 53 hosted zone

* Ec2 instance(kops, amazon linux, t2.micro,28gb storage)
* IAM user(kopsuser) attach it to instance(ec2🡪 actions>>security>>attach iam user)
* Install awscli using access key creds from iam user

ssh to instance(ubuntu)->apt update

snap install aws-cli –classic

aws configure(using access key creds from iam user)

ssh to instance(linux ami) --

curl -Lo kops 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

sudo mv kops /usr/local/bin/kops

kubectl(from browser)

chmod +x kubectl

mv kubectl /usr/local/bin

create s3 bucket

aws s3 ls (to check list of buckets in server)

aws s3 mb s3://chandana.flm.k8s

s3 versioning enabled

Deploying the pod

export KOPS\_STATE\_STORE=s3://chandana.flm.k8s (if you want to store data especially in particular bucket)

kops create cluster --name chandana.k8s.local --zones us-east-1a,us-east-1b --master-size t2.medium --master-count 1 --node-size t2.micro --node-count 2

because of this instances, security grps, ebs volumes, vpc, subnets, internet gateways, nat gateways, route tables, auto scaling grps, load balancer are created in console

Cluster configuration has been created.

Suggestions:

\* list clusters with: kops get cluster

\* edit this cluster with: kops edit cluster chandana.k8s.local

\* edit your node instance group: kops edit ig --name=chandana.k8s.local nodes-us-east-1a

\* edit your control-plane instance group: kops edit ig --name=chandana.k8s.local control-plane-us-east-1a

Finally configure your cluster with: kops update cluster --name chandana.k8s.local --yes –admin

kops update cluster --name chandana.k8s.local --yes --admin

Cluster is starting. It should be ready in a few minutes.

Suggestions:

\* validate cluster: kops validate cluster --wait 10m

\* list nodes: kubectl get nodes --show-labels

\* ssh to a control-plane node: ssh -i ~/.ssh/id\_rsa ubuntu@

\* the ubuntu user is specific to Ubuntu. If not using Ubuntu please use the appropriate user based on your OS.

\* read about installing addons at: <https://kops.sigs.k8s.io/addons>.

mkdir manifest

cd manifest

vim pod.yml

apiVersion: v1

kind: Pod

metadata:

name: pod-1

labels:

app:

spec:

containers:

* name: pod-1

image: devopsdocker/dm

ports:

* ContainerPort: 80

kubectl get nodes (shows all servers inside kubernetes cluster, if this is up then only pod is created)

kubectl create -f pod.yml (to create pod)

kubectl get pods (to check list of pods)

kubectl describe pod pod-1 (to verify if this pod has particular content or not)

kubectl get po --show-labels (to verify if this pod has label or not)

1 pod can have multiple labels

Kubectl label pod pod-1 Env=Dev (To add label to existing pod)

Kubectl get po -l app=swiggy (to list pod with label = swiggy)

Kubectl (to get list of pods which does not contain labels)

🡪 it is equality based selector

Filters objects based on exact key-value matches.

If label has app = swiggy

🡪 set based selectors

Filters objects based on multiple values or conditions.

selector:

matchExpressions:

- key: environment

operator: In

values: ["dev", "qa"]

Access the pod:

services is a method to expose pods your cluster(cluster—pod—container--application)

types:

cluster-IP if you want to access app within server not exposed externally, provides stable ip addr, DNS name.

to delete the pods

kubectl delete pod --all

vim pod.yml

kubectl create -f pod.yml

kubectl pod po -o wide

to expose pod vim service.yml

apiVersion: v1

kind: Service

metadata:

name: devops

spec:

type: ClusterIP

selector:

app: swiggy

ports:

* Port: 80

targetPort: 80

kubectl create -f service.yml

kubectl get svc

open any instance and connect terminal directly in the browser

Node port

Provides a way to expose a service on a static port on each node in the cluster.

Can access internal, external(change SGs nodes-chandana.k8s.local for all traffic and del all other SGs) through node ports only

Port Range is 30000 to 32767

Vim service.yml

apiVersion: v1

kind: Service

metadata:

name: chandana

spec:

type: NodePort

selector:

app: swiggy

ports:

* Port: 80

targetPort: 80

nodePort: 30002 (if you want to customize the port range)

port 80 frontend internal port, cannot be accessed from outside network

target port 80 is backend port

nodeport 30002 for outside network

if i access this node on port 30002 its gonna frward the req to any pod that this label app: swiggy

kubectl create -f service.yml

kubectl get svc

copy public ip addr of any instance(nodes-us-east 1b) and access it in browser

Load balancer

Can access app internally, externally by creating DNS names in cloud

Vim service.yml

apiVersion: v1

kind: Service

metadata:

name: chandana

spec:

type: LoadBalancer

selector:

app: swiggy

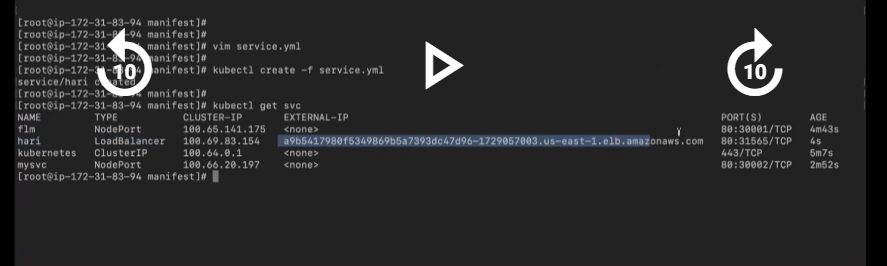
ports:

* Port: 80

targetPort: 80

kubectl create -f service.yml

kubectl get svc



You will see new loadbalancer in aws console

Kubectl delete svc hari (to delete the service)

You cannot manually delete instances/services which created using kubernetes, so dele it using kubernetes

Kops get cluster

Delete cluster

kops delete cluster --name chandana.flm.k8s --yes ( to delete cluster)

If you get error run export cmd, then re run del cmd

We can rename dns name by purchasing from godaddy

Namespaces:

Namespaces provides a mechanism for isloating grps of resources within a single cluster. Names should be unique within namespace but not across namespaces

Note: for prod cluster, consider not using the default namespace, instead make other namespaces and use those.

Initial Namespaces:

Default kubernetes includes namespace so that you can start using new cluster without 1st creating namespace

Kube-node-lease this holds lease objects associated with each node, Node leases allow the kubelet to send heartbeats so that ctrl plane can detect node failure

Kube-public readable by all clients, is mostly reserved for cluster usage, in case some resources should be visible n readable publicly throughout the whole cluster

Kube-system created by kubernetes systems

Kubectl get ns lists all available in your kubernetes cluster

Kubectl get all retrieves information abt all the resources in default namespaces, including pods, deployments, services n more

Kubectl get all –all-namespaces This extends the previous cmd to show resources from all namespaces instead of just the default one

Kubectl get svc -n kube-system lists all services running in the kube-system namespace, which contains kubernetes API server, coreDNS, kube-proxy

Kubectl create ns kubekart create new namespace kubekart, namespaces help organize within a kubernetes cluster

Kubectl run nginx1 –image=nginx -n kubekart runs anew pod named nginx1 using official img nginx inside kubekart namespace

vim pod1.yaml

cat pod1.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx12

namespace: kubekart

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

kubectl apply -f pod1.yaml

kubectl get pod -n kubekart

kubectl delete ns kubekart

REPLICA SET maintains replica of your pods.

When pod running web app, and pod goes down and users wont be able to a cess app, in such case manually need to login , del, recreate it and fix the pblm

But if we use replicaset here

Mentioning more replicas scheduler will distribute pods across multiple worker nodes, it will recreate new pods

[ReplicaSet | Kubernetes](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/)

vim replset.yaml

kubectl create -f replset.yaml

kubectl get pod

kubectl delete pod nginx1 nginx12 web2

clear

kubectl get pod

clear

kubectl get rs

kubectl get pod

kubectl delete pod frontend-qmxml frontend-s4kbp

kubectl get pod

vim replset.yaml

kubectl apply -f replset.yaml

clear

kubectl get pod

kubectl scale --replicas=1 rs/frontend

kubectl get pod

kubectl edit rs frontend

kubectl get pod

kubectl delete rs frontend

kubectl get pod

DEPLOYMENT: [Deployments | Kubernetes](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)

Provides declarative updates for pods and replicasets

Define desired state in a deployment, and the deployment ctrller changes the actual state to the desired state at a ctrlled rate.

Create a deployment

apiVersion: apps/v1

kind:deployment

metadata:

name: chandana

spec:

replicas: 2

selector:

matchLabels:

app:swiggy

template:

metadata:

labels:

app: swiggy

spec:

containers:

-- name: cont-1

image: nginx

Ports:

-- containerPort: 80

--- (3 hypens can allow you to write another file in single file)

apiVersion: apps/V1

kind: Service

metadata:

name: chandana1

spec:

type: LoadBalancer

selector:

app: uber

ports:

-- port: 80

Targetport: 80

Commands and entrypoint:

FROM ubuntu

CMD[“echo hi”]

🡪 docker run printer executes echo hi

FROM ubuntu

ENTRYPOINT[“echo”]

🡪 docker run printer hi need to pass argument hi , else container failes

FROM ubuntu

ENTRYPOINT[echo]

CMD[“hi”]

🡪docker run printer

🡪 docker run printer hello we can override the comment

VOLUMES

CONFIG MAPS collection of variables, used to store non-confidentail data in key-value pairs

Pods consume configmaps as environment variables, command-line args or as config files in a volume. No secrecy or encryption.

Secrets: store and manage sensitive informations such as passwords.

The values for all keys in the data field have to be base64-encoded strings. If the conversion to base64 string is not desirable, you can choose to specify the stringData field instead, which accepts arbitrary strings as values

# Encode text

echo -n "admin" | base64

echo -n "mysecretpass" | base64

# Create Secret

vim mysecret.yaml

apiVersion: v1

kind: Secret

metadata:

name: mysecret

data:

username: YWRtaW4=

password: bXlzZWNyZXRwYXNz

type: Opaque

kubectl create -f mysecret.yaml

# Create Pod to read secret keys

vim readsecret.yaml

apiVersion: v1

kind: Pod

metadata:

name: secret-env-pod

spec:

containers:

- name: mycontainer

image: redis

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: mysecret

key: username

optional: false # same as default; "mysecret" must exist

# and include a key named "username"

- name: SECRET\_PASSWORD

valueFrom:

secretKeyRef:

name: mysecret

key: password

optional: false # same as default; "mysecret" must exist

# and include a key named "password"

restartPolicy: Never

kubectl create -f readsecret.yaml

kubectl get pod

# Login to Pod echo print variables

kubectl exec --stdin --tty secret-env-pod -- /bin/bash

root@secret-env-pod:/data# echo $username

root@secret-env-pod:/data# echo $SECRET\_USERNAME

admin

root@secret-env-pod:/data# echo $SECRET\_PASSWORD

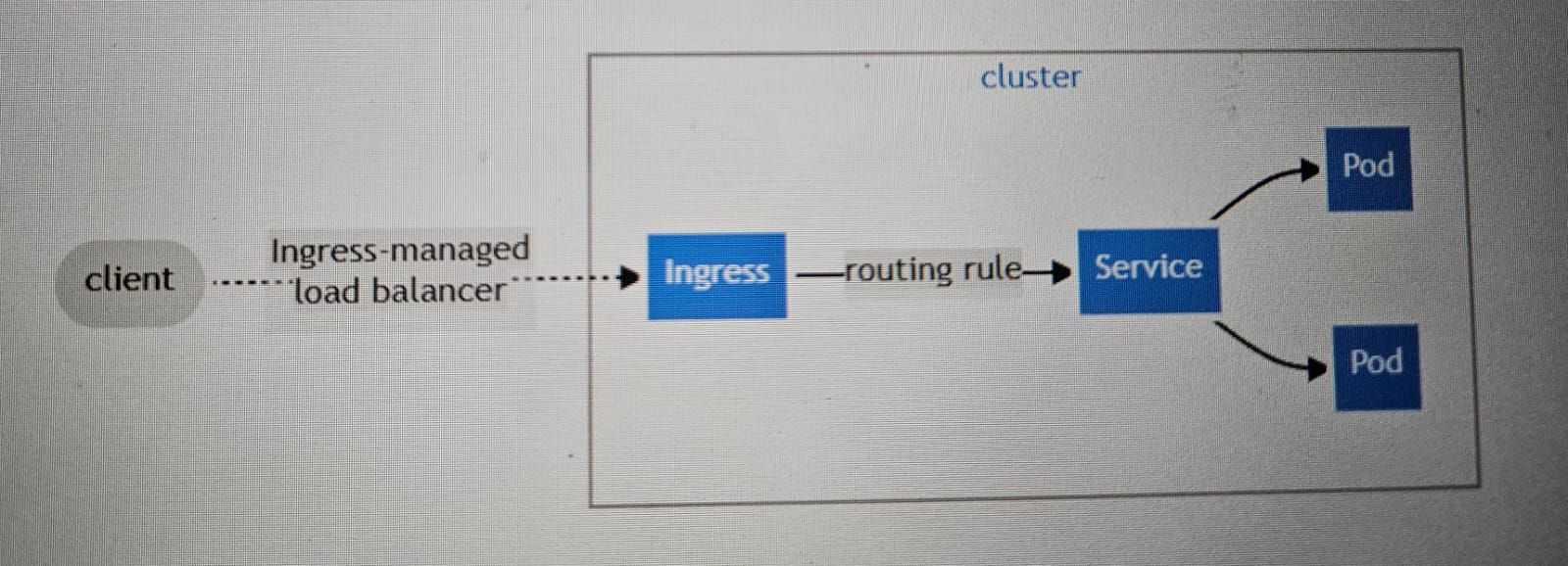
mysecretpass

Ingress::

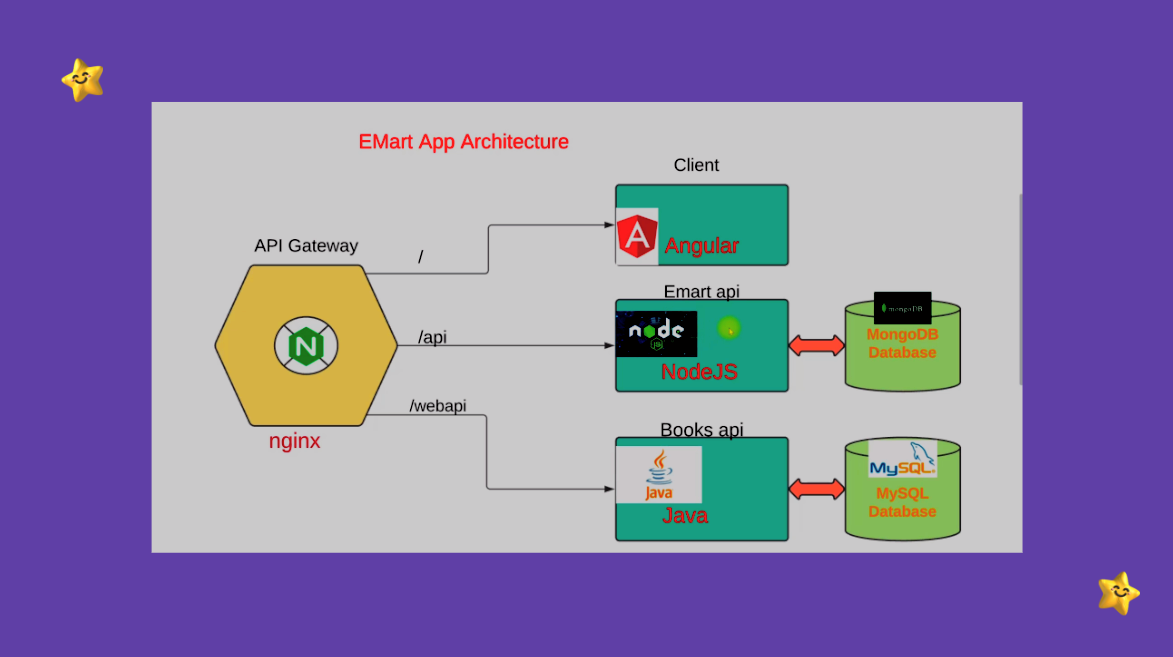
API object that manages external access to the services in a cluster,typically HTTP.

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

Traffic routing is controlled by rules defined on the ingress resource.



Ingress controllers:



**AWS Load Balancer Controller**

In AWS we use network load balancer(NLB) to expose NGINX ingress controller behind the service of Type=LoadBalancer

Same process (kops1.1) create aws cli, create kops.kubectl,cluster

kops create cluster --name=chandana.k8s.local --zones us-east-1a --master-size=t2.medium --node-size=t2.micro --node-volume-size=20 --master-volume-size=20

kops create cluster –> initiates kubernetes cluster creation.

--zones –> clusters will be deployed in diff AWS availability zones

--master-size –> sets the control plane

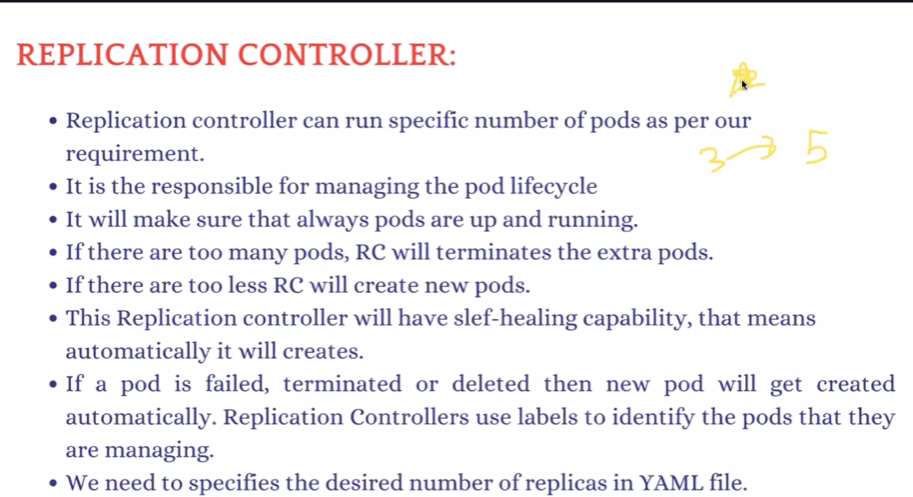
--node –> sets the worker node

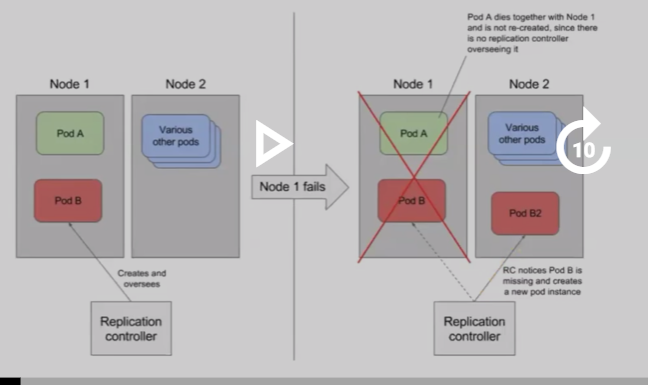
--node-volume-size 🡪 allocates storage for each worker,masternode

Auto scaling in Kubernetes is a powerful feature that helps manage workloads efficiently by adjusting resources based on demand.

Pods are deleted and created automatically based on requirement.

Replication controller:





RC 🡪pod🡪container

RC specification are pods like metadata(name),specs(how many pods)

pods specifications are containers(name, image, ports)