- **ETCD clusters** - used to keep track of deployments and loggings.

- **kube-scheduler** - To schedule on dockers and which containers to use.
  - doesn't actually do it though
- **Controllers** -
  - Node controller
  - Replication-Controller
- To monitor status & remidiate situations

- **kube-apiserver** - Manages & controll all the services on the cluster.

- **kubelet** - Is an agent which runs on each node and act as agent. It runs on that pods and monitor it basically acting as a captain of the ship.

- **kube-proxy** - To make sure proper rules are applied on worker-nodes to allow them to communicate.
  - Always look for a new service once found create appropriate rules and allow communication among pods that is done using creating IPTable.

ContainerD

① ctr - CLI for debugging

② nerdctl - alternative to docker

* crictl - used mainly for debugging purposes.

from kubernetes community

1.1 ETCD -

its a database

- key : value store database.
- By default when installed runs on port 2379 ← its a place where etcd listens

- By default you get "etcdctl" ← client

- /etcdctl --version

- etcd runs as a pod in k8 which can be accessed using
  kubectl get pods -n kube-system
- etcd. service

ETCD Commands
- etcd backup
- ↳ cluster-health
- ↳ mk
- ↳ mkdir
- ↳ set

vs

- etcd snapshot save
- endpoint health
- get
- put

ETCD
- Stores information regarding cluster

ex.
- Nodes
- PODS
- Configs
- Secreats
- Accounts
- Roles
- Bindings
- others

- Two types of deployments
  - ① from scratch
  - ② from kubeadm tool

1.2 kubeAPI-server

- kubectl get pods -n kube-system
  gives all the pods used for kube system.

Location - /etc/kubernetes/manifest/kube-apiserver.yaml

- **kube API** lets end users, different parts of cluster and external components communicate with one another.

- Responsible for all orchestration with the servers
  that's why kube-apiserver.service consist of different certificates along with different services and their location.

- kubeAPI server is at the center for all comm^n
  - Authenticate User
  - Validate Request
  - Retrieve data
  - Update ETCD
  - Scheduler
  - kubelet

location cat /etc/kubernetes/manifest/kube-apiserver.yaml.

View api-server Options

- ps -aux | grep kube-apiserver

1.3 Kube Controller

Controller-manager –
Is a process which continuously manages a components & works towards maintaining desired status
– Watch status - auto heal
– Remediate situation.

① Node controller –
Monitor the state and take actions necessary to do so.

- There are number of controllers once installed they are located at
  – kube-controller-manager.service

We have in .service
--node monitoring-period = 5 sec
so once the node container is not responding it will go into grace period which is set to be 40 sec if container still doesn't reply then eviction timeout of 5m after that new container deployed.

This settings and much more can be changed at kube-controller-manager.service.

- By default all controllers are enabled.

  - kubeadm

- kubectl get pods -n kube-system

cat /etc/kubernetes/manifests/kube-controller-manager.yaml

1.4 kube scheduler

- To schedule which pod goes on which container.

How does it decides ?
- Based on resource requirment

for ex. we need mem of 10gb in that case it will choose based on criteria to which containers has 10gb mem available while considering other factors such as CPU usage.

- ① filter nodes
- ② Rank nodes

1.5 kubelet — Captain on the ship

- ① Register a node
  ② Create a Pods
  ③ Monitor Node & pods

- kubeadm does not deploy kubelets
  Has to be manually installed

1.6 kubeproxy -

objective of kubeproxy -
- ① Look for new pods
  ② Once new pod found create a set of rules on each node to forward traffic betn each node

- creates an IP table on each node to forward traffic

1.7 Pods

- kubectl run nginx --image nginx

- kubectl get pods

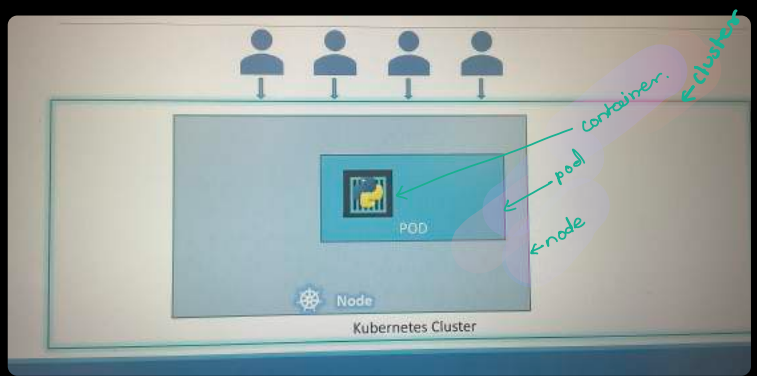
- YAML files works as input for kubernetes to run applications

- kubectl describe pod myapp-pod

Yaml to create an pod contains

- apiVersion: v1
- kind: Pod
- metadata:
- Spec:

} AKMS



Smallest is container
↓
Pod
↓
Node
↓
cluster

- Controllers

• Replication Controllers

① TO load Balancing and Scaling
② To maintain the set replication pods running

• Replica set is a new & recommanded way to set the replication set

rc-definition.yml

• kubectl create -f rc-definition.yml

• kubectl get replicationcontroller
• k get replicaseb

* New

replicaset-definition.yml

difference between replication set and replica set is

• replicaset does require selector in definition file.

* Selecters are mentioned here so that replica set can manage other pods mentioned in selecter manually which matches the name explained in selector

Selector can also be used in replication set but they are optional.

To replace/update the defination file changes.

`kubectl replace -f replicaset-defination.yml`

Option # 2

- `kubectl scale --replicas=6 -f replicaset-defination.yml`

- `kubectl get replicasets`
- `kubectl delete replicaset myapp-replicaset`

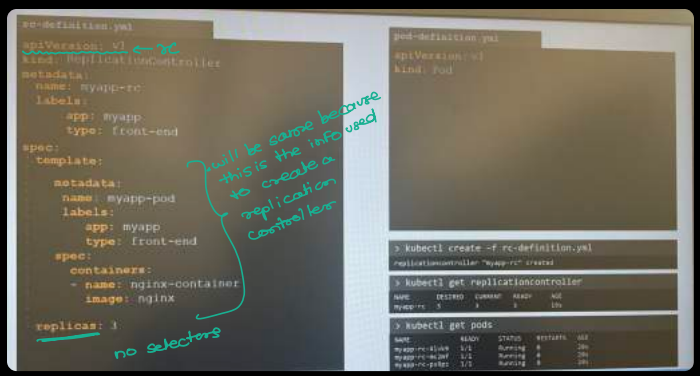
- `kubectl replace -f replicaset-def?.yml`
- `kubectl edit replicaset new-replica-set`
  (new-replica-set ↑ name)

Scaling

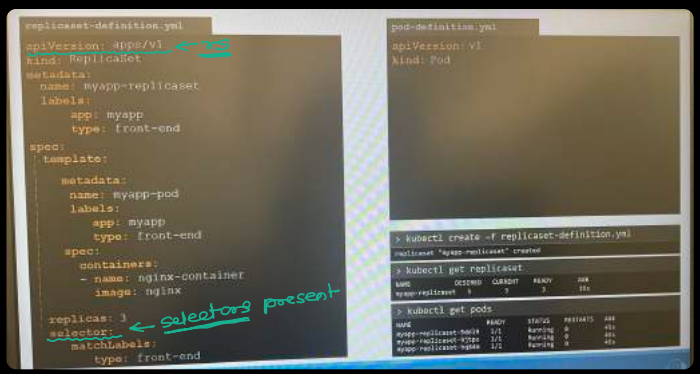
- `kubectl scale rs new-replica-set --replicas=5`

→ Replicaset defination file requires a selectors because it can also manage previously created pods.

- rc-defination.yml



- replicaset-def?.yml



Replica set

Deployments

Defination file for both will be same except the kind will be diff

• Deployments

- Deployments are used to manage the Replicasets.

- Rolling update is one of the main feature in deployment

- kubectl get all
- k get deployments

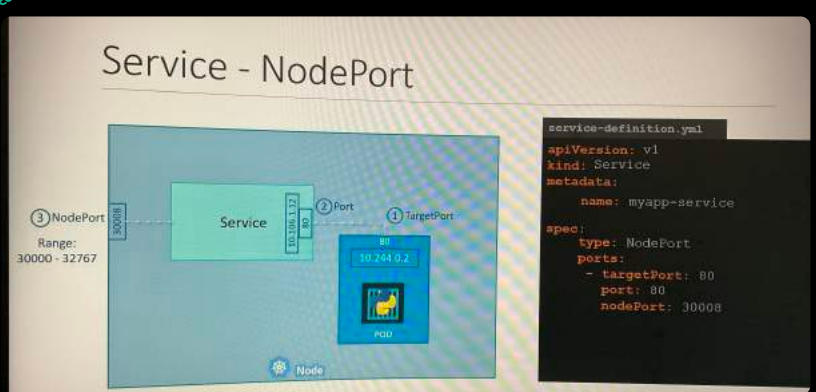
Services → enables connectivity betn pods
(This is How Networking in K8 is done)

- Service types
  - ① Node port
  - ② clusterIP
  - ③ Load Balancer

① Nodeport –
range – 30,000 – 32,767

It's called node port because it creates a virtual port for the node to communicate with
- definition file should have labels here that's how service knows which pods to communicate with

use case –
you want your particular pod/pods to be available to access from outside of your network → use nodeport
steps → create a definition file
- set port no. & values
- apply



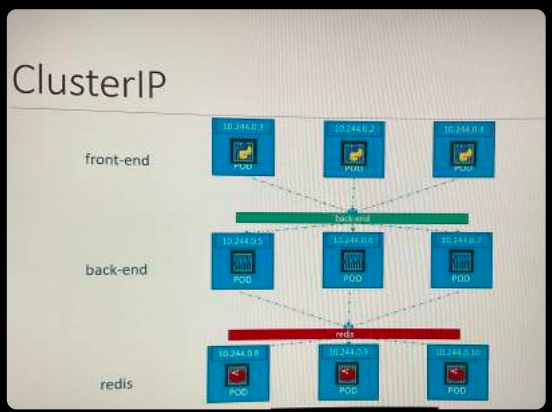
Works on
1:1
1:many
So either that nodeport is setup on particular node we can communicate with all the pods

Service is highly flexible the same config. steps can be followed for all

② ClusterIP –

Works as a clusterIP for better communication

- central interface to communicate
- Exposes pods within the cluster



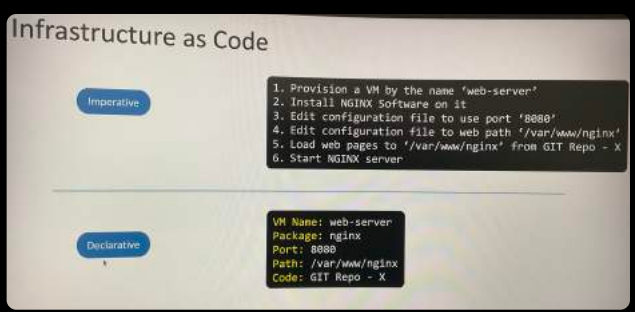
- k get service

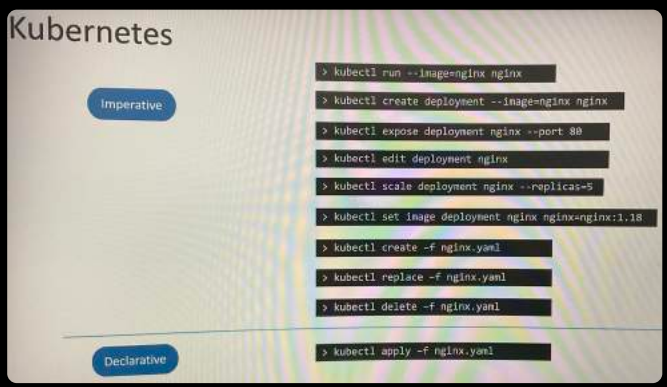
③ Load Balancer –

To make it accessible via single URL.

- kubectl get service

- ClusterIP is the default service created by kubernetes.





use kubectl apply

- Namespaces

- Basically a workspace in terms of terraform

  - we can add namespace value in defn file to make sure pod is operating in right ns.

- To create namespace simply a defination file

or

- kubectl create namespace nameofns

To see resources of dev namespaces

kubectl get pods --namespaces=dev

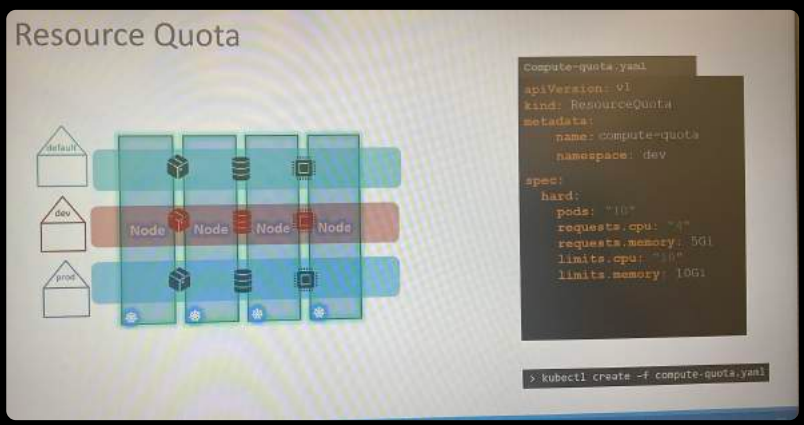
- To switch namespace

→ k config set-context $(k config current-context) --namespace=dev

- k get pods --all-namespaces

• Resource Quota – namespace

To limit resources in namespace we can create resource quota.



- k run pods -----
- k create deployments -----
- k create namespace dev

• Schedulers

* • IF pod-defination file has `nodeName` explicitly mentioned that means those nodes are manually scheduled or else if there is not there means these pods are open for auto scheduling

* if no schedulers there pods are in pending state since there def^{n} file does not have nodeName mentioned.

* nameNode can only be assign at creation time.

* To change the nameNode we have to do Binding (creating Binding obj.)

  - To check if scheduler is there
    `kubectl get pods --namespace kube-system`

  - If there is no scheduler running the pods will be in the pending state.

To see the available nodes on the cluster

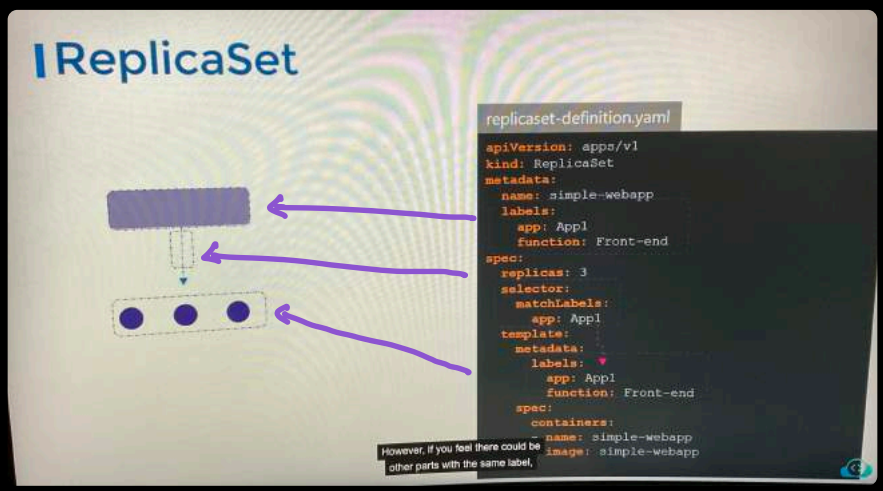
• Labels and Selectors

Labels – To filter or group

- pod-definition.yaml

```
labels:
    app: App1
    function: front-end
```

kubectl get pods --selector app=App1



ex.

kubectl get pod --selector env=prod

kubectl get pod --selector bu=finance

kubectl get all --selector env=prod,tier=frontend

• Tains & Tolerations

Tains and Tolerations are set to create a restrictions on the pods to allow or deny pods access to particular nodes.

Basically sets a restrictions on the pod.

1. set tain (restrictions) on the node
2. set allow to particular type pods so only that type of pods can be allowed on that node.

- Taints are set on nodes
- pods are set on tolerations

- kubectl taints nodes node-name key=value:

What happens to the pod if they are not tolerant ← taint-effect

① pods won't be scheduled

NoSchedule | PreferNoschedule | NoExecute

Ex. kubectl taint nodes node1 app=blue:NOschedule

- taints and pods does not restrict the pods to go to particular node. Instead it only makes sure that → particular node only accepts the Tolerated / particular pods.

If goal is to make sure certain pod only go to certain node with filling out limitations of taints and tolerations it's called as node affinity.

- Master node is automatically tainted by default to prevent any pods to run on Master node and that's a best practice.

To see this

`kubectl describe node kubemaster | grep taint`

`k get nodes`

- `kubectl describe node node01 | grep -i taints`

- `kubectl taint node node1 app=mycrm:NoSchedule`

Reminder –

if yaml file is given

descriptive : –

`kubect apply -f pod.yaml`

Imperative :

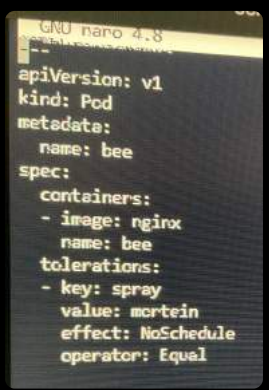
`kubectl run tom --imag=nginx`

To taint node we do this

- kubectl taint node1 spray=mortein:NoSchedule

- for tainting pod we add the tolerations inside the yaml file

ex.



- To untaint – to remove taint from node
get the taint from node

1. – kubectl describe node controlplane
copy the whole tain from there
and

2. kubectl taint node controlplane paste –

Node selectors

- To set particular pod on particular node we can take advantage of node selectors and affinity. but it does not guarantee that this pods won't be placed on other nodes

- To make sure that particular pod go to particular node only and nowhere else we have to use taints and toleration along with node affinity.

Example use-case

- To make sure the high load pods are put inside high size node.
- To basically manage the resources and avoid high load resources to put into smaller node.

- kubectl label nodes node1 size=large.

You cannot say put pod inside a node where node is not small or put pod in Large or med. To do this we have node affinity

Two ways
1. Node selectors
2. Node affinity

Node Affinity - To have advance expressions
Ex. OR, NOT

- Node affinity :-

Node selector is limited like it does not provide and OR option → to tackle that node affinity which is more Advanced is used.

pod-definition.yml

```yaml
apiVersion:
kind:
metadata:
  name: myapp-pod
spec:
  containers:
    - name: data-processor
      image: data-processor
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: size
                operator: In
                values:
                  - Large
```

Node Affinity Types

Available:

`requiredDuringSchedulingIgnoredDuringExecution`

`preferredDuringSchedulingIgnoredDuringExecution`

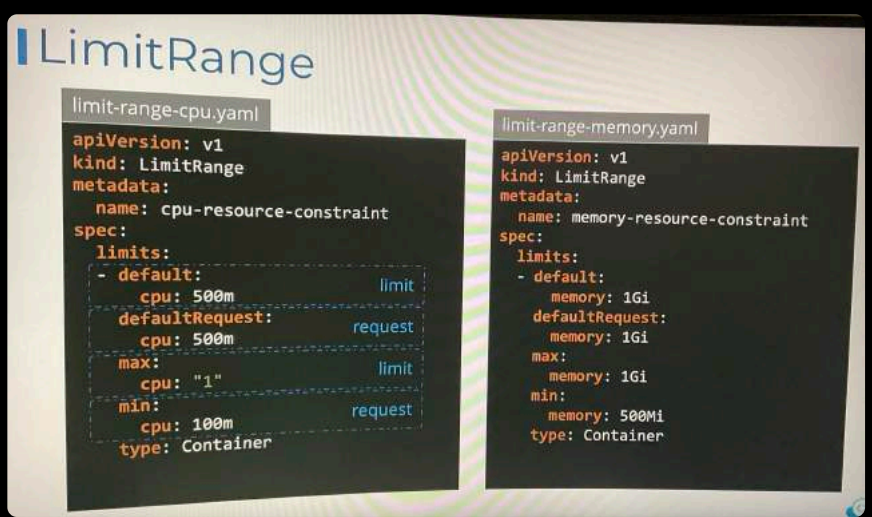
Planned:

`requiredDuringSchedulingRequiredDuringExecution`

`preferredDuringSchedulingRequiredDuringExecution`

• Resource Request

- We can add "resources" section in pod defination file to set the resources.
  - We can also set limits
- If pod constantly try to go above limited resources, Pod will be terminated with out of memory (OOM)



- Resource Quota is namespace level object

Editing a Pod

- Only certain things can be directly edited in pod
  - spec.toleration

To edit there are two options

① kubectl edit pod podname
then once that changes saved to temp since it gives error when trying to change main file
→ Once done create a new pod by using that file

2. Create an Yaml file of existing pod

- kubectl get Pod webapp -o yaml > pod2.yaml
  vi pod2.yaml – To make changes
- then delete old pod and create a new pod

• DemonSets

- Is one copy always available on each node.
- It is a set which is used to run pod of stack of pods on every single node – ex.

for logging nodes or for monitoring we want to make sure every node has this pods with services installed on them running everytime.

Daemonsets are used for deploying background services on the nodes for ex. system operations service, collecting logs, monitoring frameworks like prometheus and storage volumes

- kubectl get daemonsets --all-namespaces

- kubectl describe daemonset kube-proxy --namespace = kube-system

• Static Pods

- When we have pods which are created by kubelet on its own are known as static pods.

↙ Captain of the ship. todo this copy yaml files to /etc/kubernetes/manifest

To create such pods copy your yaml files to

/etc/kubernetes/manifest

location is saved in kubelet.service
2. We can also put configfile in kubelet.service which consist the config for static pod.

* only pods can be created this way not ds, rs

k get pods --all-namespaces

- So with -controlplane at the end listed are static pods.

* fun-exercise

— Deleting static node.

1. Look for which node has that pod
   k describe pod -o wide → node found
2. Get IP of that node since static pods can only be deleted when we `rm yaml file`
   `k describe node node01 -o wide`
3. SSh to that node then find pod location default → /etc/kubernetes/manifest if not check `/var/lib/kubelet/config.yaml` get loc^n from there
4. Once found rm pod.yaml.

— x —

Custom Schedulers

We can have a custom schedulers set.

- We might want to add extra filtering layer to pods before they can be scheduled
- or we want to replace the default scheduler we can do all this

To see currently running scheduler

- `kubectl get events -o wide.`

- kubectl logs my-custom-scheduler --name-space =kube-system

- kube-system ^node is where all the system services are located for kubernetes.

- To see active scheduler and other services

  - k get pods --namespace = kube-system.
  - k get namespaces

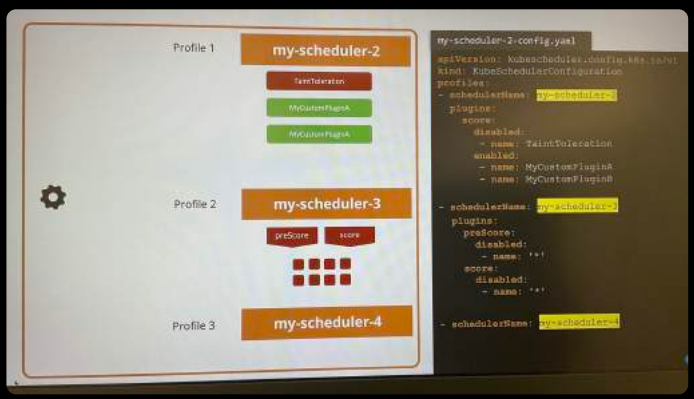
- Scheduling Phase -

1. Scheduling Queue
2. Filtering
3. Scoring
4. Binding

- We can also set priority to pods to add them in the start of Scheduling Queue.

- Scheduler profiles

They are used to merge all the custom schedulers at one location to minimize confusion & reduce resource usage and manage all in single binary.



Logging and Monitoring

- In pods services kubelet has a service called cAdvisor which sends all the metrics to different metrics services to monitor.

- kubectl top node
- kubectl top pods

- Application logs

- kubectl logs -f podname

↑

for live logs

- If container contains multiple pods then to get a log for particular pod we have to explicitly mention the name of the pod.
- ex. k logs -f event-simulator-pod pod-name
  ↑ container name

4.0 # • Application Lifecycle management

• Rollouts

- kubectl rollout status deployment/ myapp-deployment

• To see history

• kubectl rollout history deployment/ myapp-deployment

• By default Rolling update

- Where one by one update is done on the system.

Two update strategies

—① Recreate —

first it is scaled down to 0 then new version is deployed.

② RollingUpdate

- Scaled down 1 by 1 & simultaneously scaled up.

• kubectl rollout undo deployment/ myapp-deployment

- To see the current deployment strategy

- kubectl describe deployments

In kubernetes to run commands on containers which we just created we can use command: ["sleep"] As an entrypoint

4.1 Application Command

- Environmental variables
  - Another way to env is set it directly inside defination file.

1. ConfigMaps -

If we have multiple environment variable in pod-def^n file it is better to save them in seperate file which is called as configMap

- configMap consist of configuration data

To view configMap.

- kubectl get configmaps
- kubectl describe configmaps

To view the environmental variables in the pod

- kubectl describe pod.

* As stated before kubectl edit may not allow us to make the required changes

In that case we just save the temp file and delete old pod and create new

or

- `kubectl replace --force -f paste location`

2. • Secrets

- To store sensetive information

imperative -

```
kubectl create secret generic \
app-secret --from-literal=DB_Host=mysql
           --from-literal=DB_User=root
```

Declarative -

```
kubectl create -f
```

While saving secreats make sure you save them in encoded format

↳ To encode `echo -n 'mysql' | base64`

- `kubectl describe secrets`

- `kubectl get secret app-secret -o yaml`

↳ To get values.

To decode

- echo -n 'encodedtext' | base64 --decode

- Secrets are not encrypted they are only encoded.

- Secrets are not encrypted in ETCD

- Anyone able to create pods/ deployments in the same namespace can access the secrets.

- Consider external secrets providers like AWS, GCP.

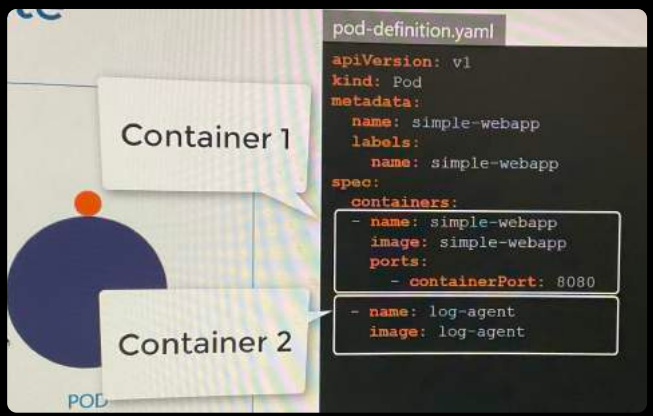
- Anyone with access to etcd can able to see encoded information too

3. Inside defination file

- You can set env inside defination file

ex.
```
env:
- name: APP_COLOR
  valuefrom:
      secretkey.
```

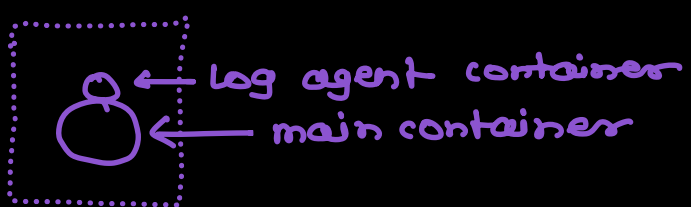
• Multicontainer Pods



- Containers are smallest inside a pod

- To create multi-container setup for the pod change the manifest file → Add two different container names into it

We use multicontainer setups in usually along with 1 pod as main pod and other being log agent



• multi-container

· Cluster Maintaince

If we wanna use the node to upgrade or anything. you can use command

- `kubectl drain node-1`

  ↳ pods are gracefully terminated and turned on different node.

  and that node-1 will be unschedulable until we disable it.

- `kubectl uncordon node-1`

  TO allow scheduling again

- `kubectl cordon node-1`

  Just to make node unschedulable

  - no pods termination like drain and

- To empty the node and mark it unsched-

  · `k drain node01 --ignore-daemonset`

  ↑

  if we use this command on a pod which is not a part of replicaset it will not work.

- Backup and Restore

two options - querying etcd or Baking up file.

If we did some changes imperative way to document thoes changes query the kube-apiserver to save thoes changes

ex. - kubectl get all --all-namespaces -o yaml > all-deploy-service.yaml

tools like velero ← for Backup.

- to save a snapshot in ETCD client

etcdctl snapshot save snapshot.db

Different Authentication Methods for kube-apiserver

① Static file
② Tokens
③ Certificates
④ Identity services

* Security Basics

- ① Symmetric key - Single key
- ② Asymmetric - public & private

5. Security

Three types of certificates
1. client
2. root → installed on root ex. Symantec
3. Server Certificates

Namin convension

Public key
- Server.crt
- Server.pem
- client.pem

Private key
- Server.key
- Server-key.pem
- client.key

*.crt *.pem

private key has key mentioned in their names.

*.key *-key.pem

- Different Services talks with each other using TLS secure connection
- Every service in k8 has a certificate and private key to communicate among different services.

- To sign this certificate either we can use self sign (not recommanded) and signed by CA (certificate Authorities)

To generate certificates

Tools
↳ - EsyRSA
- OPENSSL
- CFSSL

View certificates in already existing System.

- cat /etc/kubernetes/manifests/kube-apiserver.yaml

It will show all ther certificates once file shows all the certificates run the

- openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout

- Creating certificates with openssl

① `openssl genrsa -out ca.key 2048`

② openssl req -new -key ca.key
-sub "/CN=K-CA" -out ca.csr

③ openssl x509 -req -in ca.csr -signkey
ca.key -out ca.crt

④ To view current crt info

openssl x509 -in file.crt -text -noout

- certificate API is handled by `controller Manager`
  - k get csr

- Adding TLS of new user

- New user joins a team and asking for access to cluster →

① Get CSR and key from user

② encode that csr into base64

```
cat newuser.csr | base64 -w 0
```

③ Once encoded put the base64 encoded csr to defination file

④ View the status of csr → It will be in the pending state now

- k get csr
- k certificate approve <csrname>

- k describe CSR name -o yaml
  -o wide

Inspect Service log

- Journalctl -u etcd.service -1

for kubeadm

kubectl logs etcd-master

When kube-apiserver or kubeadm are down we can still access the logs using docker cmnds

docker ps -a → To list all pods

To view

- docker logs 87fc

To resolve any certificate related issue which caused any service to stop working

ex. kubectl is stopped working

- check docker process logs

- docker ps -a |grep kube-api

for CRI compatible resources use

↳ Container runtime interface.

`crictl ps -a`

`:2379` → default port for etcd

`critctl logs containerid` from `ps`

or

`docker logs containerid`

All certificates related to etcd are saved at `/etc/kubernetes/pki/etcd`

Certificate API –

To manage certificate signing process.

- `kubectl get csr`

1. To create a key

- `openssl genrsa -out jane.key 2048`

2. To generate sign in request using that key

- `openssl req -new -key jane.key -sub "/CN=jane" -out jane.csr`

To generate Base 64

- Cat filename.csr | base64 -w 0

To see the status

kubectl get csr

kubectl certificate approve akshay

deny

kubeconfig

- Provides authentication and authorization information for accessing a k8 cluster.

- Assume that I want to access pod info from my cluster (non-admin) in that case I have to pass information like this " k get pods
  -- server my-kube:6443
  -- client-key admin.key
  -- client-certificate admin.crt
  -- certificate-authority ca.crt
  We cannot put all this info again n again, that's why we use kubeconfig
- by default config file is there in HOME directory that's why we don't have to put the info so far.

kube config defn file has three sections
1. clusters
2. contexts
3. users

- k config view

kubeconfig default file path

- /root/.kube/config

To know current context —

- kubectl config --kubeconfig=/root/my-kube-config current-context

To change current context

- kubectl config --kubeconfig=/root/my-kube-config use-context research

API

To see API versions -

curl https://kube-master:6443/version

- curl https://kube-master:6443/api/v1/pods

Authorization Mechanism

1. Node Based
2. ABAC - Attribute base
3. RBAC - Role based
4. Webhook -

We create a attributes for everything so each policy or authorization will be separate attribute — which will be hard to manage

- We create Roles and that roles can be associated with the particular group.

We use webhook and API

- Always allow
- always deny

Where do we specify them?

- they are set on kube-api server

- /etc/kubernetes/manifest/kube-api.yaml

- By default they are set to always allow
can also specify multiple modes

ex — --authorization-mode=Node,RBAC,
webhook

To see the default roles
— k get roles

To see the access of different roles

- k describe role kube-proxy -n kube-system

-n = namespace

- k describe rolebinding kube-proxy -n
kube system

- check access
- k auth can-i delete nodes
- k auth can-i create pods --as dev-user

How to create user and give permissions

-① create a role → if dev — developer

```
• k create role developer \
-- verbs=list,create,delete \
-- resource pod
```

② create rolebindings for that user so that we can bind that <u>sam</u> user to that role

```
• k create rolebindings dev-user-binding \
--namespace=default --role=developer \
--user=sam
```

OR

We can just create manifest files

• cluster roles

- Set directly on the cluster so all the nodes inside that cluster follows.

• k get clusterroles

• k get clusterrolebindings

• Service Accounts

Two types of Accounts in kubernetes

① User Account

② Service Account

- Service Accounts are used by services for monitoring and logging purposes.

  Ex. prometheus uses service account to get monitoring information

  - every namespace has default service account.
  - We can use that tokens to external soft services to like prometheus.
  - In new update you will have to generate token manually

② Service Account

To create

```
kubectl create serviceaccount name
```

To view

```
• kubectl get serviceaccount
```

This creates a token which can be used by services to access the account

To view the token

```
kubectl describe secret token-name
```

This will show the token id

• Image Security

Containers are seperated using namespaces but shares the same kernal

In docker host where all containers are seperated by containers

- If we run `ps aux` command from docker host. It is also going to show the processes running in container with different process id's.

Security context

- Security content provides way to set various aspect of security such as user and group IDs.

- By default kubernetes is set to **all allow**

- We can add network policy to the pod which will only allow the traffic which is explicitly allowed.

- **To implement the network policy we use labels and selectors.**

• Storage

Storage in Docker — fs → /var/lib/docker

Layered Architecture for docker

- L1- Base Ubuntu layer
  changes in apt packages
  changes in pip packages
  Source code
  L5: Update Entrypoint with "flask" cmd.

  L6: Container layer - made when we create container base of this above layers

Volumes —
— docker volume create data_volume.
to mount

docker run -v data_volume:/var/lib/mysql

all volumes are stored in
/var/lib/docker

- Storage drivers are the drivers who takes care of layered Architecterse in the docker.

PV - persistant volumes

PVC - claimed by user or pod
way to dynamically request a specific amount and type of storage without needing to know the underlying structure.

Ex. Asking for 50 GB storage is called PVC.

- To Bind PV & PVC ther is Binding criteria
- if no volumes available PVC will be under pending state.

- k get persistentvolumeclaim
- k delete -||-

- `pvreclaimpolicy` is set to Retain by default
  - means it will stay until it is deleted by the admin.

PV

- A storage in the clusters

PVC

- A request for that storag

• Networking

• Switching :- Switch is used for local communications.

cmd • ip link

cmd • ip addr add 192.168.1.10/24 dev eth0

cmd • route

Router -

Comm^{n} bet^{n} two networks

• route → To display routing table

• To add routing bet^{n} two networks

• ip route add 192.168.2.0/24 via 192.168.1.1

ip add of diff network where we want to connect

Router's ip address which works as Gateway to connect to diff network

. default gateway for all the traffic

• ip route add default via 192.168.1.1

• ip forwarding enable

— cat /proc/sys/net/ipv4/ip_forward

• DNS

• All DNS entry are set inside /etc/host file for local but as entry grows it becomes hard to manage such entry that's where DNS server comes into picture

. local host file for entry table

- DNS server keeps track
- Location to set DNS server ip address
  - cat /etc/resolv.conf

- Which file to check when looking for DNS is set in /etc/nsswitch.conf ← We can edit this to look for DNS server first

- nslookup sam.com
- dig sam.com

- Network Namespaces

- They are used to create a seperation bet-n different network's routing table, ARP tables.
  - so container has no idea about host's Ip address routing table or any other network config.

- To create network namespaces
  - ip nets add red
  - ip nets
- To run a ip link command inside red
  - ip nets exec red ip link

or

- ip -n red link

- To see ARP table
  - arp

- To add connection betn two network namespaces

- Docker Networking

- Below is None network which is limited to that container only — no comm
  - docker run --network none nginx

- network option #2
  - docker run --network host nginx

- network option #3

• CNI - Container Networking Interface

- set of standards set to develop a networking solution
- CNI has plugins which uses scripts to handle networking problems.

- Docker doesn't support CNI directly we can do it with work around.

Ports

# Name	# Port
• kube-api	6443
• ETCD	2380
• kube-scheduler	10251
• kubelet	102550
• kube-controller-manager	10252

- To get internal IP address of nodes
  - k get nodes -o wide

- To get Mac address of that post
  - ip a | grep -B2 (ip address)

(Make sure you ssh to that node before running)

- To get default gateway
  - ip route show default
  - ifconfig

- How to see ports pod listening on ?
  - netstat -nplt

- CNI in kubernetes

CNI location is stored in
kubelet.service

CNI location - /etc/cni/

• Docker Networking

· docker info – To see list of available networks

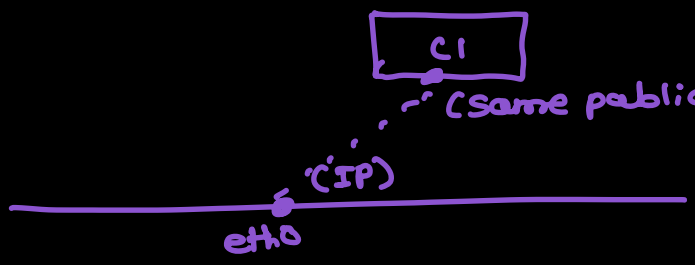
Types of networks on docker
- Bridge
- Host
- ipvlan
- macvlan
- null
- overlay

By default – Bridge network

① Bridge Network

· With Bridge containers can communicate with each other but inter-Bridge communication needs a gateway setup.

② Host Network



- With host Network type Container uses the same Ip address as docker veth

Advantage

- Dye to using same public Ip latency reduces and performance increase.

Disadvantage

- Since public Ip it's open to internet that means more valunerable for attacks

We can create multiple containers and all can use same Ip address just different ports.

We use service alongwith pods for networking

k get services

k create services

so that service manifest file will have a static ip address assigned to it

Service usage -

① Directing traffic to the correct pod

② Load Balancing

* Every pod in k8 see's the same Ip address

* for labels it is mandatory that both the labels (if you have two labels) They should match exactly.

one matching and other not then that won't be considered as part of that group.

- DNS in kubernetes

• Design and Install a k8

- Understand the purpose of Installing cluster

If Education
- Minikube
- Single node – AWS

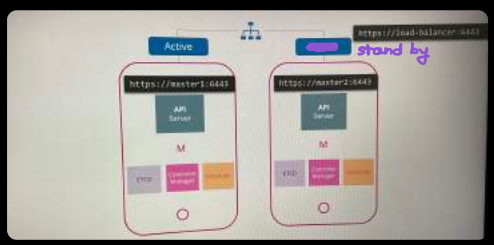
- Development and Testing
  - Multi-node cluster with a single Master and multiple workers
  - Setup using kubeadm tool or quick provision on GCP or AWS

→ What happens when Master node is not working?

As long as the worker nodes are working functioning of pods is active However, creation or update cannot be done. We cannot Access cluster through API.

• High Availibility

- To have high availability its better to use two Master nodes alongwith Load Balancer.



- To prevent any conflicts betⁿ two masters and their services it is better to put one Master in Active mode and other onto standby.

- We can decide which Master to put put Active by leader selection process

- kube-controller-managers --leader-elect true

ETCD HA

- When ETCD is installed with same stack as other services its called as stacked topology.

Disadvantage –

When even one master is down etcd becomes unavailable

that's why we can install etcd seperately called as **External etcd topology**.

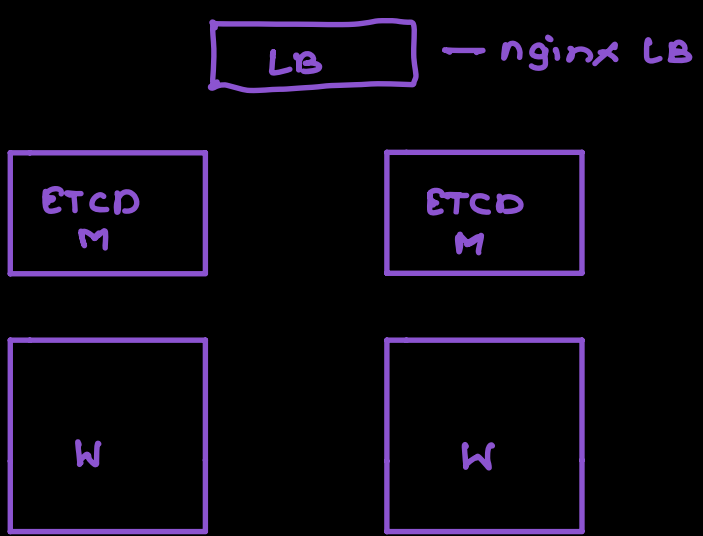
Disadvantage –

- Harder to setup
- requires more numbers of etcd servers.

Where the etcd server is ?

You can check that by

- cat /etc/systemd/system/kube-apiserver.service.



Installing kubernetes

- Each node requires container - runtime installed to run containers on the pod for kubernetes
- Docker uses cri-dockerd

- There are different types of containerd available in the market
  - containerd
  - CRI-O

- Troubleshooting in controlplane

if installed by kubeadm
— look for pod in kube-system namespace.

- if installed as service

service kube-apiserver status

kube-controller
kube-scheduler
kubelet
kube-proxy

To check logs here →

- sudo journalctl -u kube-apiserver

check logs of the both

*** kube-system pods are static pods so their manifest files can be found in

/etc/kubernetes/manifest

Make changes here.

- If node crashed
  - ① check nodes
    - k get nodes
  - ② k describe node name
  - ③ top
  - ④ df-h
  - ⑤ service kubelet status
  - ⑥ sudo journalctl -u kubelet
- ⑦ check certificates

- `openssl x509 -in /var/lib/kubelet/worker-1.crt -text`

ex. so node01 was not working properly
To solve this issue
- k get nodes
  describe node
* - sudo journalctl -u kubelet - to get logs
check reason IF it says stopped
- `ssh node01`
  - `service kubelet status`
  - `service kubelet start`

- Since We are dealing with systemd managed system we have to use `journalctl`

- check logs with

  `sudo journalctl -u kube-system`

  ssh to that node all the config related to kubelet will be at

  # # # `/var/lib/kubelet/config.yaml`

  One more in `/etc/kubernetes/kubelet.conf`