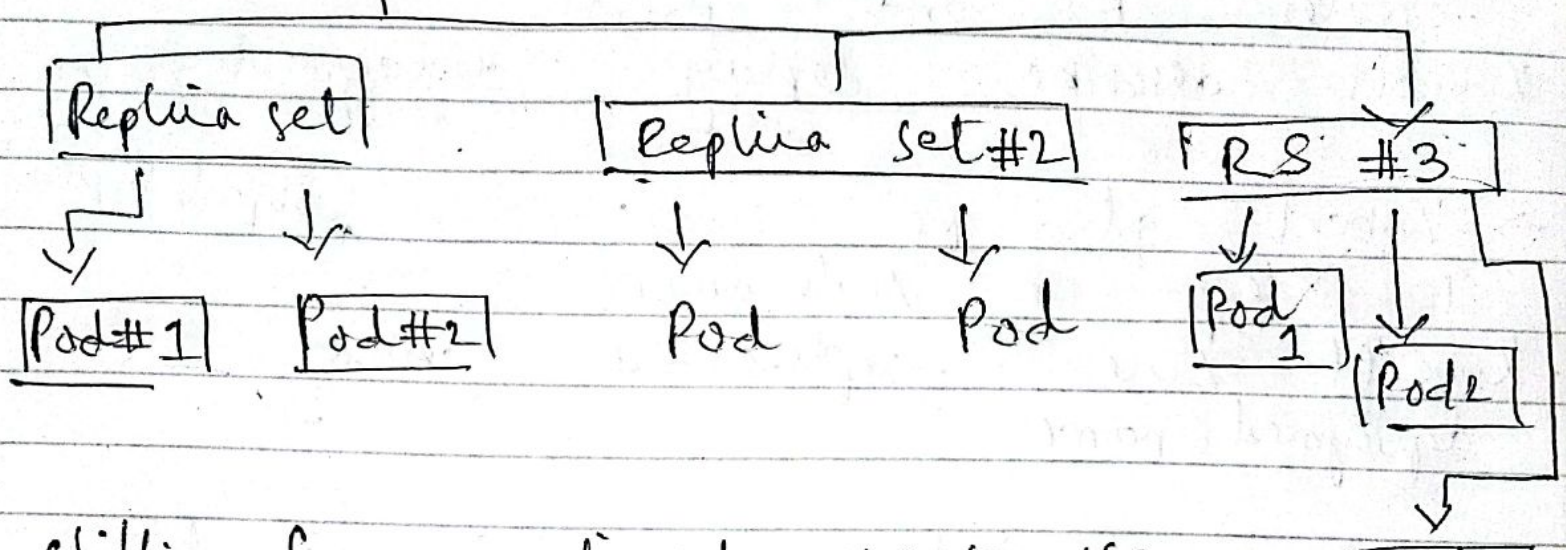
= tz-medium

lecture # 50: Deployment and Roll back.

Deployment (controls replica set).



shifting from replicaset version #3 to 2 or 2 to 1 is known as roll back.

→ A deployment provides declarative updates for pods and replicasets.

⇒ deployment is more powerful than replicaset.

Scanned with CamScanner

-> command to get back to original/previous

Kubectl rollout undo deploy/mydeployment
--to-revision = 2 (if version not given then just roll back to 1 version behind like 4 se 3 pe)

Cmd -> kubectl get deploy

To check deployment was created or not??

-> kubectl get deploy

-> command to check how deployment creates RS and pods
Kubectl describe deploy deployment_name

-> kubect get rs
To scaleup or scale down
. kubectl scale --replicas = 1 deploy (object type)
deployment_name

-> To check what is running inside container
kubectl logs -f <podname>

-> kubectl ~~rollbo~~ rollout history deployment
deployment_name

Some important commands:-

-> kubectl describe deploy my_deployment
(object, deployment name)

-> Command to scale up or down replicas in deployment:-

Kubectl scale -replicas = 1 deploy my-deploy.

-> agar deployment ki file ko update karenge ~~to~~ aur phir apply karenge (kubectl apply -f file.yml) then phir ek new replicaset create hoga.

-> Rollout mai version previous achieve hota hai but no. of pods current wale deployment ki rehte hain. (V1 has 2 pods and current V3 has 5 pods)
when you roll out from V3 to V1 you will have 5 pods not 2 but sirf kaam pichle wala ajata hai

-> jab deployment create hoti hai tou sath mai replicaset bhi create hoti hai (you can confirm this using kubectl ~~get dep~~ describe deployment deployment_name).
Inside ~~dep~~ details you can confirm that replicaset is created.

Scanned with CamScanner

→ kubectl get pods.

mydeployment_79d695ab2_abc79812d

↓ deployment name

↓ id of replicaset

↓ id of pod

deployment will create replicaset and replicaset will create pods.

13/6/22

KuberNetes Networking

Kubernetes networking addresses four concerns:-

① containers within a pod use networking to communicate via loopback.

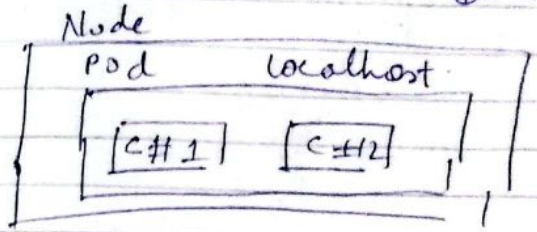
② Cluster networking provides communication between different pods.

③ The service resources lets you expose an application running in pods to be reachable from outside your cluster.

④ You can also use services to publish services only for consumption inside your cluster.

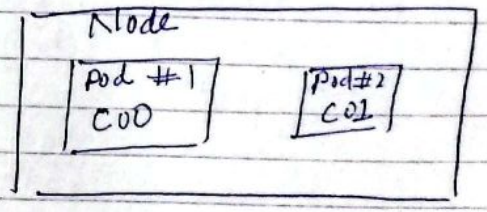
cluster :- group of nodes.

→ multiple containers within a pod can communicate with each other via ~~tixat~~ localhost.



→ ek pod k multiple containers same node pe hi honge.

Scenario #2.



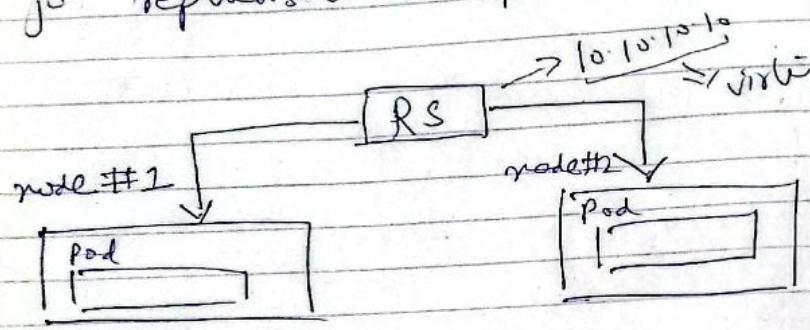
→ pod to pod communication on same worker node happens through pod IP.

There is a problem b/c we studied in a replica set whenever the pod is recreated, it will

172.17.0.4
172.17.0.5

VIP = Virtual IP

get the new IP address. To kya har defa hum pod ki new ip yaad rakhenge ??
So the solution is object type in Kubernetes known as "services" jo replicaset k upar lagta hai.



→ services will provide virtual ip to replicaset. This virtual ip will be mapped with pods of replicaset

→ Actually pod ki apni original ip k upar ek ip map hoegi
mapped virtual ip

Pod ki ip

→ so jab 10.10.10.10:80 pe request bhejenge then ye request ~~replicaset~~ ~~series~~ services pe jaegi then to replicaset phir replica set decide karega k kis running pod pe request ko forward karna hai.
(request within cluster)

→ Front-end pod ko backend pod se baat karne k lye VIP useful hai q k pod recreate hone par pod ki IP change hoti hai. Ab jo bhi ~~pod~~ ~~banenge unse~~ VIP k through communicate kiye jaa sakta hai

→ sare pods ki ip mapped hogi replicaset ki virtual ip k sath

→ When using RC, pods are terminated and created during scaling or replication ~~controller~~ operations.

→ services object is a logical bridge b/w pods and end users, which provides virtual IP and allows client to reliably connect to the containers running in the pod using the virtual ip.

→ pod has unique ip address, those ip's are not exposed outside the cluster.

→ services help to expose the Virtual IP mapped to the pods and allows application to receive traffic 10.1.1.1

→ Basically hum pods pe label lagaenge name = kamil then services pe bhi label lagaenge name=kamil phir jis pod pe name=kamil ka label hoga us pod ki IP mapped hogi

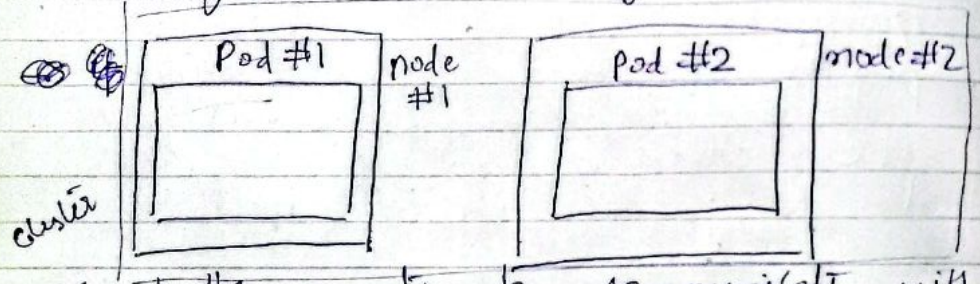
Scanned with CamScanner

services ki VIP K sath.

→ services will create an endpoint to acces pod/application

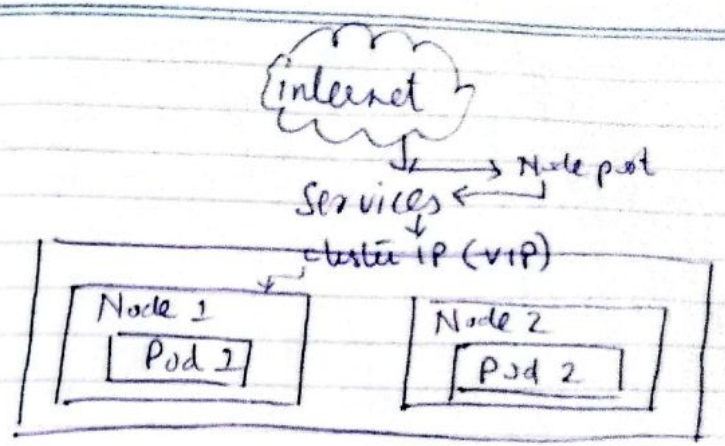
→ By default services can run only b/w ports 30,000 - 32767.

→ The set of pods labeled by a service is usually determined by a selector.



if pod #1 wants to communicate with pod #2 of node #2 then it will also used services → clusterip method pod #1 will use VIP to talk to pod #2

Note:- Virtual ip works within a cluster but how can we access the application outside the cluster, the answer is Node port.



Node port ranges 30,000, 32,767

→ Node port cluster IP se uper K a concept hai. we assign node a perific port.

public DNS Node 1 : port_NO_node 1

~~(It will take us to pod running in node #1).~~

agar pod Node #1 pe hega to wahe se serve hega otherwise from node 2

Flow of Communication:

→ Internet → Node port $\xrightarrow{\text{connected}}$ cluster IP (VIP) → pod ← container.

→ Node port makes a service accessible from outside the cluster.

→ Exposes the services on the same

port of each selected node in the cluster using NAT.

Internet
↓
node port
↓
virtual IP of cluster
↓

Node [Pod containers]

VOLUME CONCEPT :-

→ container K andar jo volume hai wo container K delete hone K sath delete hojata hai so in K8s, a volume is attached to a pod. container K delete hone se Pod ko farq nhi parega.

Pod ⇓

[volume [C00] [C01]]

Pod delete hogaya to volume gaya.
→ pod recreate hone par volume empty hoga

Hostpath :-

→ Use this when we want to access the content of a pod / container from host machine.
A hostpath volume mounts a file or directory from the host node's filesystem into your pod

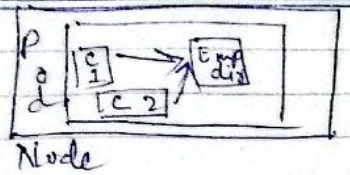
Lecture #52 :- 25-june-2022.

Persistent Volume And Liveness Probe

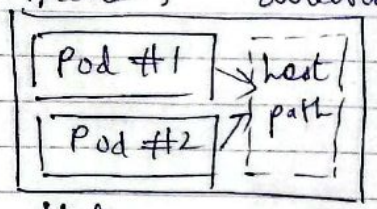
→ A persistent volume (PV) is a cluster-wide resource that you can use to store data in a way that it persists beyond the lifeline of pod. All nodes in the cluster that access this Persistent v

Two important storage concepts related to ~~host~~ storage

① Emptydir :- common storage b/w containers in a pod. If the container is deleted, we have no issues but if the pod is deleted the storage is ~~lost~~ lost.

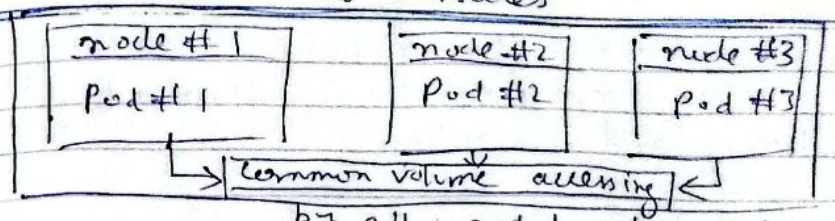


② Host path: common storage shared by pods within in a node. If pods get deleted the backup is available at node's directory like "EFS".

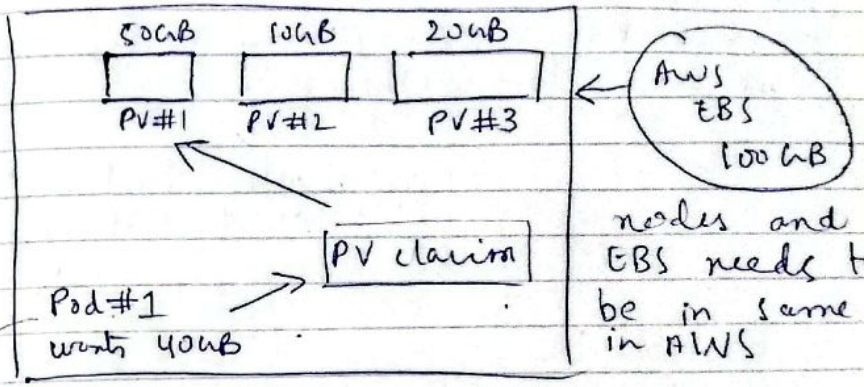


But the question that arises is agar "pod_x" delete hogaya aur next time wohi "pod_x" node 1 ki jaga node 2 pe deploy hogaya to host path ka concept fail hogaya b/c host path is created inside a node and not been accessible outside the node so the concept of persistent volume comes to the picture where a common volume is created is a cluster of nodes ~~which~~ and this volume is ~~accessible~~ accessible by various pods ~~in a~~ from multiple nodes

cluster of nodes



by all pod 1 pod 2 and pod 3



nodes and EBS needs to be in same AZ in AWS

→ while writing manifest we claim that how much storage we required for our pod, that storage we will claim from one of the persistent volume which is available

Liveness Probe / Healthcheck

→ It actually checks whether the application in a pods' container working properly or not ??

→ For running healthchecks, we would use cmds specific to the application, if the

cmd succeeds, it returns 0 and the kubelet considers the container to be alive and healthy.

Initial delay second : 5
period seconds : 20
time outseconds :- 30

for eg: starts first health check at 1:05 pm
↓ next check
1: 25 pm
↓
1: 45 pm

20 seconds k gap se health check hoga.

time outseconds: 30: what does it mean??

It means that ya to reply 0 ayega ya 1 from health check but agar 30 seconds tak koi reply nhi ayega then it means kill the process / container and starts again. and recreate a new container

Refer to the file in laptop

lecture #53

CONFIG MAP AND SECRETS

→ we have different environments like development, QA, Production

while performing application deployment on K8's cluster, sometimes we need to change the application configuration file depending on environment like QA, dev, stage or Prod.

Dev	QA	Prod	
env #1	env #2	Env#3	Env#4

→ Yani jese jese hamara K8's cluster different env se pass ho tou usay us environment k hisaab se configuration file milli jaye.

→ Configmaps are useful for storing and sharing non-sensitive, unencrypted configuration information. Use secrets otherwise.

→ Config map are not intended to act as a replacement for a property file

Config map can be accessed in following ways:-

① As environment variable ✓
② As volumes in Pod ✓

→ Config file stored in virtual memory jab tak object waha pe exist karta hai

→ The secret data on nodes is stored in tmpfs volume (tmps is a file system which keeps all files in virtual memory).

→ The API server stores secrets as plaintext in etcd

① Secrets can be created
① from a text file
② from a yaml file

To be continued with Lab.

Lecture # 54 (5-7-2022)

Name Spaces AND RESOURCE QUOTA

→ Normally so far whatever ~~we~~ resources we have created, have in default namespace.

→ Namespace is similar to branches concept in git.

→ K8s namespaces help different projects, teams or customers to share a K8s cluster and provides

→ We can use resource Quota on specifying how many resources each namespace can use.

→ By default, a K8s cluster will instantiate a default namespace when provisioning cluster.

→ Nodes and persistentVolume are not in any namespace. They are on cluster level. while resources like pods, services, replication controllers are in namespace

Scanned with CamScanner

43:00

10

How to Create Namespaces.

apiVersion: V1
Kind: Namespace
metadata:
  name: dev
  labels:
    name: dev.

→ Kubectl get pods.
It will search pods in defaulit namespace. But if you want k namespace change hojae aur jo bhi searching ho wo kisi aur namespace mai ho. Then the command is:-

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

→ A pod in K8s will run with no limits on CPU and memory

→ You can optionally specify how many CPU and memory (RAM) each container needs.

→ CPU is specified in units of cores and memory is specified in units of bytes.

1 CPU = 1000 mi
CPU = 500 mi = 0.5 CPU

→ Two types of constraints can be set for each resource type Request and Limits.

→ A request is the amount of that resources that the system will guarantee for the container and K8s will use this value to decides on which node to place the node.

→ A limit is the max amount of resources that K8s will allow the container to use.

① if request = not mention
  limit = mention
then
  Request = limit

② if request = mention
  limit = not mention
then
  Request ≠ limit
and limit will be set to default (which is allocated for that cluster/namespace).

Scanned with CamScanner

if limit and request both are not mentioned then limit = request

b/c both limit and request are unlimited.

→ A K8s cluster can be divided into namespaces. If a pod/container is created in a namespace that has a default CPU limit and the container does not specify its own CPU limit, then the container is assigned the default CPU limit.

→ You can limit compute, memory and storage

→ Every container that runs in the namespace must have its own CPU limit.

→ To be continued. ---

consider this scenario

Resource Quota

```
apiVersion : v1
Kind: ResourceQuota
metadata
    name: myquota
spec:
  hard:
    limits.cpu : "400m"
    limits.memory : "400Mi"
    requests.cpu : "200m"
    requests.memory : "200Mi"
```

we have a deployment file

```
Kind: Deployment
apiVersion: apps/v1
metadata:
    name: deployment
spec:
    replicas: 3 ——— pods = 3
    selector:
    template:
        metadata
        spec:
            containers:
              - name
                image:
                commands
                resources:
                  requests:
                    cpu: "200m"
```

Each pod will take "200m" so 3 pods will take "600m" which is not available as we have set limits.cpu = "400m" ~~for~~ in resource quota for this namespace.

Scanned with CamScanner

lecture # 55

Kubernetes Horizontal Pod Autoscaling :-

Default Range :- (when you create limit Range in Kind)

CPU :-

request = 0.5 CPU (minimum value)
limit = 1 CPU (maximum value)

Memory :-

request = 500 MB (minimum value)
limit = 1 GB (maximum value)

minimum value :- container ko is value se kam nhi mil sakta.

maximum value :- container ko is value se ziada nhi mil sakta. in terms of CPU and memory.

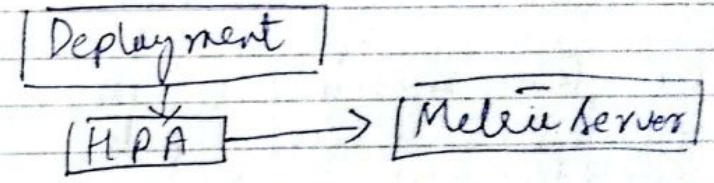
Provided that we have created "Kind: LimitRange" file for CPU and Memory. If we dont create LimitRange file we can set limit value > 1 CPU or 1 GB and request less than 0.5 CPU or 500 MB. The above min and max value will only work if you create "Kind: Limit Range"

For detailed study and concept you can read Kubernetes horizontal pod autoscaling lab #8 in laptop

---

Horizontal Pod Autoscaler

Metric Server :- pods ki metric collect karta hai. Metrics means CPU and memory utilization and check/balance storage utilization and sends data to HPA (horizontal pod autoscaler). HPA will then inform deployment/replicaset to create pods based on the metric values of the pods.



→ Pods autoscaling /scale up/scale down is done using HPA.
→ Metric Server is kind of API
→ HPA is one of the object type in K8s.
→ K8s mai bydefault autoscaling nhi hoti. for this we use HPA and Metric server

→ The controller fetches the metrics from the resource metrics API for each pod targetted by the Horizontal Pod Autoscaler.

pod 1	pod 2	pod 3	pod 4
30%	40%	20%	70%

160/4 = 40%

Now is 40% ki base pe autoscaling hogi.

Steps:-

① Lets create a deployment (Pod will be created).

② Attach deployment with autoscaling (we have a command for this)

③ Go to container (increase load) you'll see others pods also created.

(we will set min and max no. of pods created in case of loads on pods).

lecture #56

Kubernetes Jobs, Init Container and Pod lifecycle

→ Difference b/w pods and jobs?
pods can be recreated themselves. jobs wont ~~pod can be terminated itself But for job you have to go and terminate yourself.~~

→ For scheduling task, we use jobs (pods mai schedule based kaam nhi hota).

Use Cases for Jobs:-

① Take Backup of DB
② Helm chart uses jobs
③ Running Batch process
④ Run a task at a schedule interval
⑤ Log rotation

→ Job is a kind of pod and kaam pura hone k baad job k ~~and~~ andar ka pod (container) delete/terminate hojata hai

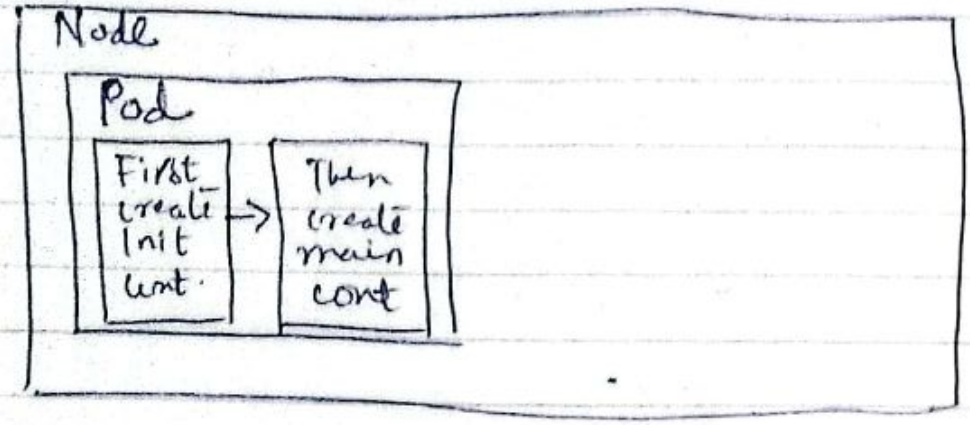
The Cron job pattern:-

If we have multiple nodes hosting the application for

Scanned with CamScanner

high availability, which nodes handles Crop?

INIT CONTAINER:



In case of init container, jab tak init container ka kaam successfully execute nhi hoga tab tak main container create nhi hoga.

→ Init containers are specialized container that run before app containers in a Pod.

→ If a init container of pod fails, K8s repeatedly restarts the pod until init container succeeds.

→ init containers do not support readiness probe.

Scanned with CamScanner

SOME IMPORTANT POINTS RELATED TO NETWORKING IN KUBERNETES :-

TYPES OF SERVICES :-

1. Cluster IP
2. ~~Ne~~ Node Port
3. Load Balancer

Cluster IP :-

→ frontend pod ko backend pod se communicate karna hai within a application. We know, pods ki ip change hoti rehti hai upon creation/~~dele~~ deletion so now frontend pod cannot directly communicate with backend pod due to this change of ip issue. So a Kubernetes service can help us group these pods together and provide a single interface to access the PODs in a group. For example services created for the backend pods will help group all the backend PODs together and provide a single interface for other pods to access this services. Ab backend k sare pods apni services k interface k peche ajaenge

→ Service for backend (Interface) → { All backend pods.

→ Ab frontend ko backend se communicate karna hoga to direct backend se communicate nhi karega; instead backend k interface per request bhejega.

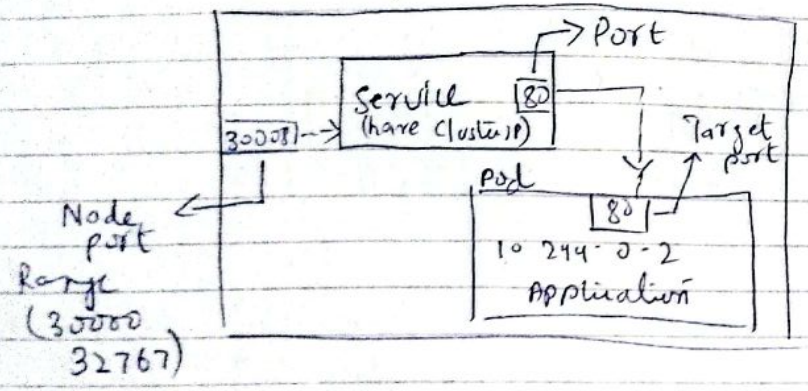
→ Each service gets an IP and name assigned to it inside the cluster and that is the name that should be used by other PODs to access the service. This type of service is known as cluster ip.

→ Hum pods pe label lagaenge name = Kamil then services pe bhi label lagenge name = Kamil then jis pod pe name = Kamil ka label hoga us pod ki ip mapped hogi services

ki VIP K sath.

-> Services can help us by mapping a port on the Node to a port on the POD.

NODE PORT :-



service-definition.yml

```
apiversion: v1
Kind: service
metadata:
    name: myapp-services
spec:
    type: Node port
    ports:
        target Port: 80 ----->(Pod Port)
        port : 80 ------->(service port)
        node port: 30008 --->(node port)
```

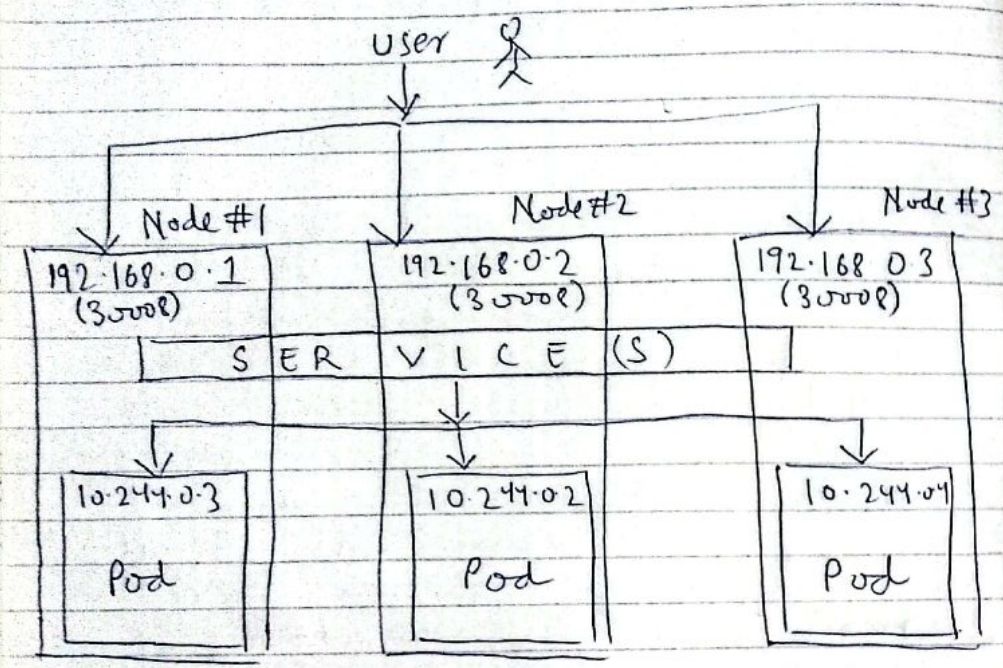
If we dont provide target port it is assumed to be the same as port. (i.e service port)

-> We have simply specified the target port but we didn't mention the target port on which POD. There could be 1000's of PODs with web running on port 80. We use labels and selectors to link PODs with services. We know pods were created with a label, we just need to bring that label into services definition file. Under ~~labels~~ ~~we~~ selector, we gonna provide a list of labels to identify the POD.

-> The service acts as built in load balancer to distribute load across different PODs.

-> Node port ko kese pata chalta hai K kis service pe attack (connect) karna hai? like there could be multiple services. The answer is service definition file, where we mentioned node port and port. The node port 30008 connects to

service running on port 80.



→ Say if user hits http://192.168.0.1:30008 the request will go to Node #1 (server)
→ then to service (cluster IP) → service will then look for pod(s) it should make a request to → if that pod is not available in node #1 and is available ~~in~~ at node #3, then this request will ~~hit~~ route to node #3, where the requested pod is currently available.

~~User → http://192.168.0.1:30008 (send request to node #1) → service~~

~~User~~
~~↓~~
~~Node #1 ip: node port~~
~~↓~~
~~service (having ~~cluster~~ cluster ip)~~
~~↓~~
~~will search~~

User on the internet
↓ make request
http://node #1 ip: node port
↓
service (having cluster ip)
↓
service will look where the pod is availabe in the cluster, whether the requested pod is available at node #1, node #2, or node #3. wherever the pod(s) is available ~~is~~ in ~~available~~ cluster, service will route request to that node for the pod (where application is running).

Scanned with CamScanner

→ services definition k through service ko pata hota hai k kin pod(s) se connect hona hai. (using labels in selector)

→ service definition file k through service ko pata hota hai k node-port kis service se attach hai. i.e (node port and port).

→ Endpoints = Pods

→ In nodeport the service makes an internal POD accesible on a Port on the node.

CKA

28th August, 2022:-

→ Components like scheduler, kube-controller and kubelet cannot directly interact with etcd. They inform API server and then Api server communicate with etcd.

Namespace :-

DNS:-

db-service.dev.svc.cluster.local

- db-service → service name
- dev → name space
- svc → service
- cluster.local → domain.

→ In declerative approach we dont give step by step instruction. focuses what to do instead how to do ??

→ We cannot move running pod from one node to another.

→ Taints are set on nodes and tolerance are set on pods.

Scanned with CamScanner

Kubectl taint nodes node-name
key = value : taint-effect

Noschedule | Prefer Noschedule
| No Execute

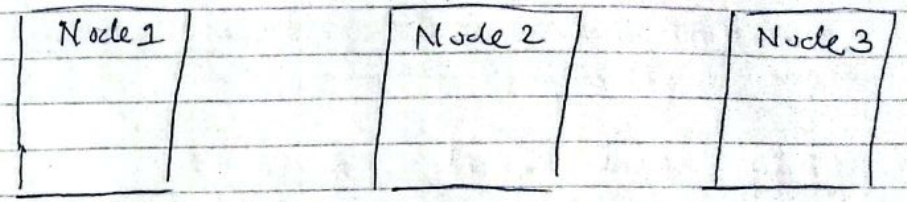
what happens to Pods that do not tolerate this taint.

→ Taints and toleration is used when we want to reserved any particular node for specific pods (application) so we will taint the node with application name and add toleration to the pod (application)

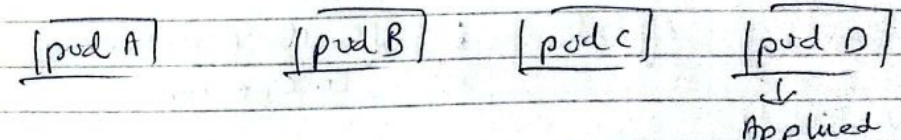
→ Taint tells the node to accept the pod with a specific toleration

→ If the requirement is to restrict pod to certain ~~node~~ ~~pod~~ it is achieved using concept known as "node Affinity"

Consider a scenario



Applied Taint to node 1



Applied on pod D the toleration

→ Node 1 will only accept pod "D" but node "D" can also be placed on node 2 and node 3 b/c there is no restriction (taint) set on node 2 and node 3.

→ But agar hum chahte hain k pod D sirf node 1 pe hi deploy ho node 2 ya node 3 pe kabhi deploy na ho then we use node affinity.

Scanned with CamScanner

20

→ ~~Restriction on~~ on node (using Taint ~~and~~ to

→ Restriction on nodes achieved using taints and toleration.

→ Restriction on pods achieved using node affinity.

→ study taint effect ~~on~~ in K8s. and command to taint and untaint.

→ Taints (node) effects :-

① Noshedule

② Prefer noshedule

③ No execute

By default, pods have no toleration

means that it will apply on all Pods that are going to be (scated).

→ means that it will apply on all pods that are already ~~part of~~ created (means this taint effect also apply on existing pods unlike Noshedule policy).

Note :- on next time, continue with taint and toleration concept. (lecture) practice test

16.1 CKA completed (3-9-2022)

Daemon Sets :-

Daemon set are like replica sets, as it helps us to deploy multiple instances of pod. But it runs one copy of pod on each node. It helps to maintain one copy of pod is always present in all nodes in the cluster. And every node has a pod. (DS dont have replicas in their definition file)

Static Pods :- (created by kubelet)

static pods are managed directly by the kubelet daemon on a specific node without the API server observing them. The pod running on a node are visible to the API server, but cannot be controlled from there. These pods have also have node name appended to it

→ Anything that is appended to the docker run command, will go into the "args" property of the pod definition file in the form of array

Dockerfile		K8s
ENTRYPOINT ["sleep"]	⇒	command: ["sleep2.0"]
CMD ["5"]	⇒	args: ["10"]

Note :- command overwrites entrypoint

Scanned with CamScanner

instruction whereas args field overwrite the CMD instruction in dockefile

→ Agar dockefile mai "Entry point" and "CMD" hai but pod definition file mai sirf "command" hai tou pod file ka "command" dockefile ki "Entry point" ko overwrite kardega and dockefile ka "CMD" use nhi hoga wo ignore hojaega.

Cluster Maintenance :-

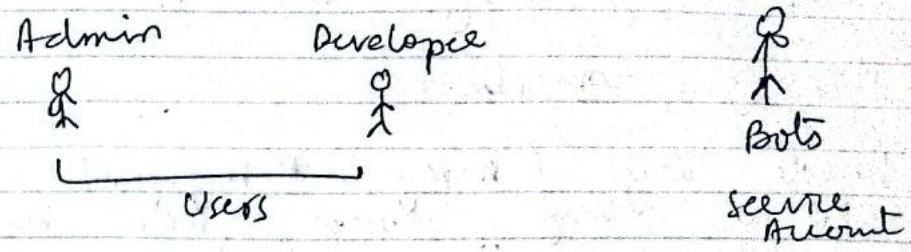
→ Drain Concept :- When node is on maintenance.

→ Uncordon Concept :- maintenance of node is completed and configure node to be schedulable again.

Kubeadm :- A tool used to build and upgrade K8s cluster component. Kubeadm performs the actions necessary to get a minimum viable cluster up and running quickly.

Security :- By default, all pods can access all other pods within a cluster.

Accounts :-



→ All User access is managed by the kube-apiserver.

TLS Basics :-

→ Symmetric Encryption :- Same key is used to encrypt and decrypt the data. The Problem was that key has to be sent over the n/w which is not secured way

→ CA signed CSR using its own private key i.e ca key.

TYPES OF CERTIFICATES :-

① Server certificate (Generate by web-server)

② Client Certificate (Generate by client like when we want to ssh into machine)

③ Root Certificate (Generate by CA after ~~w~~ validating the CSR)

<u>Naming Conventions:-</u>

Private Key (*.key *-key.pem)
→ Private Key always have key included in their files)

Certificate (Public Key) (*.crt, *.pem)

→ CA has two certificates (ca.crt and ca.key)

→ Generate Keys → admin.key
CSR → admin.csr
Sign Certificates → admin.crt (self signed)

→ if kubectl stops working it means kubectl utility unable to connect with the kube-api-server (inspect the logs of kube-api-server container using docker ps -a | grep kube-apiserver.

→ study TLS in K8s
→ elid lab#2.

→ Certificates related stuff is managed by controller manager.

→ ~~csr~~ User ko cluster access karne k liye sirf csr nhi banana hota balke csr object bh banta hai jis mai csr ki encoded key hoti hai.

→ config file (By default stores under $Home/.kube/config

→ Kisi bhi user ki jo cluster ko access karna chahta hai uski key and certificate is file mai define hota hai.
Inside this file we have:-
① Clusters ② Users.
③ Context (means what user can access what cluster) like admin@Production

Here admin = user and production = cluster

→ Kube proxy ≠ kubectl proxy

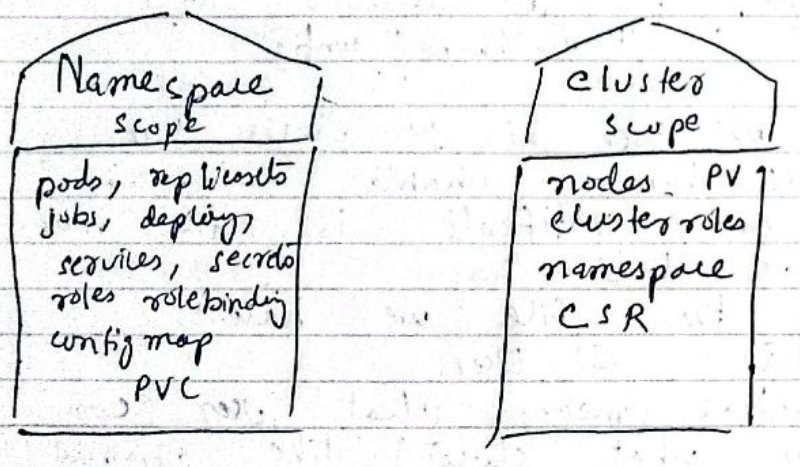
Kube proxy = establishes connectivity b/w pods and services across different nodes in the cluster.
Kubectl proxy = HTTP proxy service created by kubectl utility to access the kube api server

Scanned with CamScanner

→ Role binding is the object that is used to attach the user to the role
→ role itself is a separate object

→ Roles and Roles binding falls under the scope of namespaces.

Resource(s) Distribution :-



Account TYPES :-

① User Account :- Used by user to access the cluster / perform administrative task

② Service Account :- Used by app to access the cluster

→ we cannot edit the service account of existing pod. Must delete and ~~delete~~ recreate the pod.

→ when a service account is created, it first create the service ~~acc~~ account object, and then generates token for service account, it then creates secret object ~~to~~ and stores that ~~secret~~ token inside the secret object.

→ This secret object is then linked to the service account.

→ The token is needed to access the Kubernetes API

Service Account object ← linked to — secret object
↓
stored token inside
↓
needed to authenticate to Kubernetes

Network Policies :-

The entities that a pods can communicate with are identified through a combination of 3 identifiers.

① Other pods
② Namespace
③ IP block (Traffic to and from node)

Important lecture :-

STORAGE :-

This will create vol inside default volume directory in docker host i.e /var/lib/docker/volumes

→ docker volume create volume_name

docker run -v ~~data~~ volume-name : /var/lib/mysql image_name

This is volume bind (volume-name is inside default directory).

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

This is bind mounting.

~~mount path~~

Volume Mounts :- ——> inside container

-mountpath : /path/ in / container / pod

name: which volume (must be already created)

volumes :- ——> host

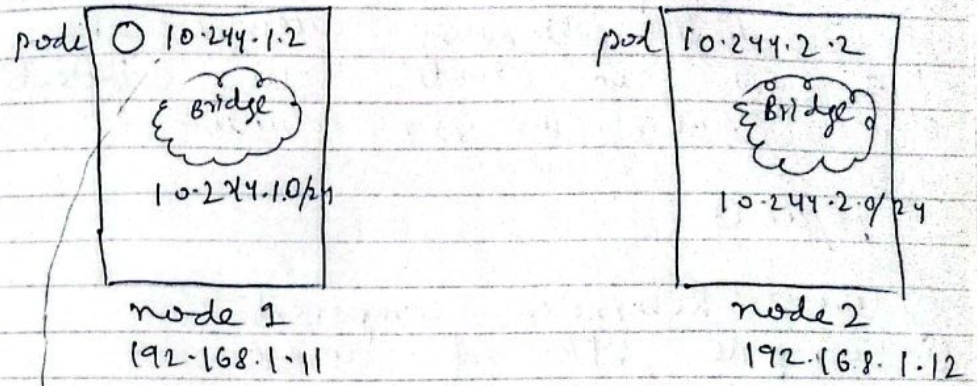
-name : must match with above name

hostpath :

path: /var/log/webapp → host

Networking :-

communication from pod to pod but on different nodes.



→ wants to ping Pod 10.244.2.2. on node#2 But How ??

node 1 $: ip route add 10.244.2.2 via 192.168.1.12

IP address of node #2 192.168.1.12 works as Gateway.

CNI is responsible for creating container network namespaces, identifying and attaching these namespaces to the right network

→ Study N/w ~~interfaces~~ namespaces and CNI in K8s.

Scanned with CamScanner

Service-name . namespace-name . svc . cluster . local

INGRESS :-

→ Ingress helps users access your application using a single externally accessible URL that you can route to different services within ~~in~~ your cluster.

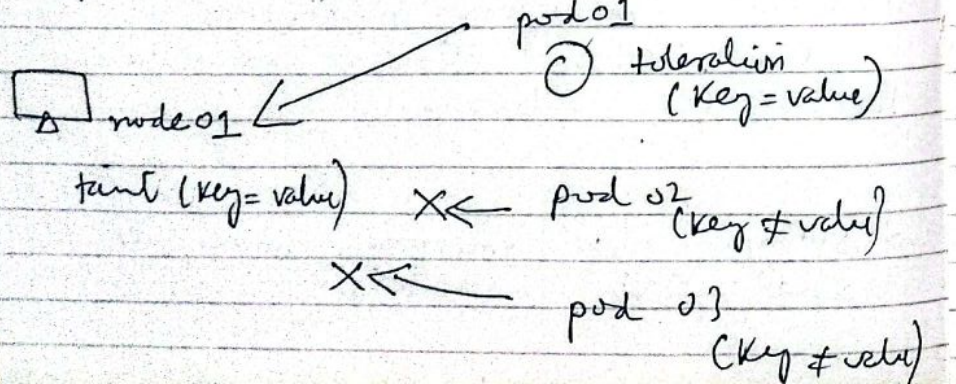
11/3/2023 :-

① Learn Kubernetes components.
② complete 14 of course.

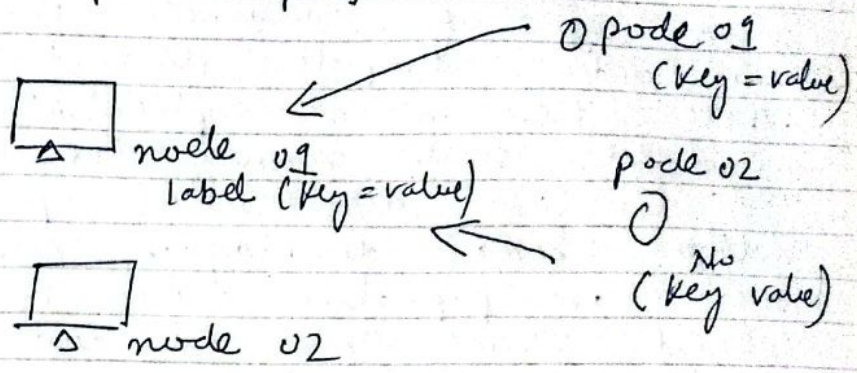
Approaches :-

Declarative Approach vs Procedural (Imperative) Approach

TAINT AND TOLERATION :-
(protect node)



Node Affinity :-
(protect pod)



Let see we the requirement where pod A can be placed on node A
pod B " " " on node B
pod C " " " " node C

Then use both taint and toleration and node affinity concept.

ConfigMap :-
By using configmap you can easily update the configuration of an application without having to rebuild container image. They store non-confidential configuration data such as environment variables, command line arguments or configuration files.

Scanned with CamScanner

IN ii Container :-

An init container in K8s is special type of container that runs and complete its task before the main application container starts running. They are used for task like setting up database schema, downloading config files, or populating data in shared volume. Once init cont. complete its task the main container can then start and use the results of these tasks.

pod-eviction-timeout = ye wo time hai K pod K ready hone Ka time. Is time tak K lye pod Ka wait kiya jata hai aur ye ~~o~~ kube-controller-manager pe set hota hai. Is time K baad node Ko dead consider Karliya jata hai. means node unavailable.

uncordon = node ~~un~~ available.

drain = node unavailable. ~~(cordoned)~~

→ check the difference between drain and cordon the node. (mentioned in CKA notes).

drain = All present (current) workload will be evicted.

cordon = future workload will be evicted.

Network providers (Calico and flannel)

In K8s, n/w providers are responsible for providing n/w connectivity b/w pods running on different worker node in K8s cluster. They provide their different features and trade offs, choosing the right one is very important, to ensure best performance, scalability and security.

The version shown in the "kubectl get ~~poo~~ nodes" command represents the version of the Kubernetes control plane components running on each node in the cluster. The components include API server, etcd controller manager and Kube-scheduler. Whereas in worker node, the version means the version of kubelet component that is running on each worker node.

node01 Ready <none> 84m v1.26.0

→ context = specific cluster, namespace and user that kubectl will use to interact with K8s API server.

Scanned with CamScanner

Service N/W:- 27

When you define a nodeport/clusterIP service in Kubernetes, it exposes the target port (80 in this case) of the service on a high port number (the node port) on each node in cluster. In '80:30081/TCP' configuration, it means that traffic arriving at port '30081' on any node in the cluster will be forwarded to port '80' of the pods associated with the service. In short, the target port ('80') is specific to the pods within the cluster, while the node port ('30081') is the port exposed on each node to allow external traffic to reach the service.

Ingress:-

An ingress is used when we have multiple services on cluster and we want the user request to be routed to the specific service based on path. Unlike nodeport or load balancer, ingress is not actually a type of service, instead it is an entry point that sits in front of multiple services in the cluster.

Ingress controller and ingress resources:-

Before we implement the ingress, we need to deploy a supported reverse proxy or load balancing solution like nginx, Haproxy, or Trafik. Then we need to specify a set of rules to configure the ingress. The solution we deploy is called an ingress controller and the set of rules that we configure are called as ingress resources. Ingress resources are created using definitions files. Ingress controller is responsible for implemented the rules defined in the ingress resources and managing the actual traffic routing and load balancing. The rules like forward /foo to foo service and forward /bar to bar service is managed by ingress resources.

Scanned with CamScanner

Networking, namespaces, DNS, Services:-

→ When we create service of type "cluster ip" and we mapped it to the list of pods using labels. we say this service is accessible from within the cluster only. What does it mean??

It means the service is accessible only from pods and services running within the cluster. worker node(s) and master node are host (machines) so the "cluster ip" service is not supposed to be accessible from these machines. If you want to access the service directly onto the ~~the~~ worker node or the control plane node, the solution/answer is service of type node port and loadbalancer.

Service of type Node port:-

Node IP: Node port → cluster IP: Port → Pod: Target port

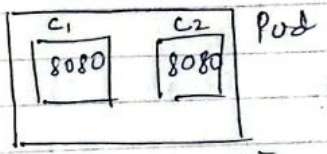
Service of type loadbalancer:-

External-IP-of-LB: Port → Node IP: Node Port → cluster IP: Port → Pod: Target port

If you make a service of type loadbalancer (a cluster IP and node port service will be implicitly spawned as well).

A cluster-IP service is a part of NodePort service. A node port service is a part of loadbalancer service.

Multiple Containers Using same port inside Kubernetes Pod



It is easy to achieve this in docker by using ~~8080~~ 8082:8080 host cont and 8083:8080 for the → containers

B/c in docker the containers dont share a single n/w namespace and thats why they can listen on same container ports, which are mapped to different ports on the host. This is not the case with Kubernetes. In a pod all containers inside that pod, share same n/w namespace.

Scanned with CamScanner

-> ~~Restriction on~~ on node (using Taint and to

-> Restriction on nodes achieved using taints and toleration.

-> Restriction on pods achieved using node affinity.

-> study taint effect ~~on~~ in K8s. and command i.e taint and untaint.

-> Taints (node) effects :-

① Noschedule
② Prefer noschedule
③ No execute

By default, pods have no toleration

means that it will apply on all Pods that are going to be created.

> means that it will apply on all pods that are already ~~part of~~ created (means this taint effect also apply on existing pods unlike noschedule policy).

Lt on next time, continue with taint and toleration concept. (lecture) lecture test.

16-9 CKA completed (3-9-2022)

Daemon Sets :-

Daemon set are like replica sets, as it helps us to deploy multiple instances of pod. But it runs one copy of pod on each node. It helps to maintain one copy of pod is always present in all nodes in the cluster. And every node has a pod. (DS dont have replicas in their definition file)

Static Pods :- (created by kubelet)

static pods are managed directly by the kubelet daemon on a specific node without the API server observing them. The pod running on a node are visible to the API server, but cannot be controlled from there. These pods have also have node name appended to it

-> Anything that is appended to the docker run command, will go into the "args" property of the pod definition file in the form of array

Dockerfile		K8s
ENTRYPOINT ["sleep"]	=>	command: ["sleep2.0"]
CMD ["5"]	=>	args: ["10"]

Note :- command overwrites entrypoint

Scanned with CamScanner