- Access nginx-svc by using name :-

Name of my service : nginx-svc

/# curl http://nginx-svc

O/p: <!DOCTYPE html>
<html>
<h1> Welcome to nginx! </h1>
</html>

So, its working.

/# nslookup nginx-svc → nslookup Command

↓ ↓

It gave IP-address : 10.0.242.134. what it does ?

But other things it couldn't find.

Q) what does nslookup do and what is purpose of /etc/resolv.conf

Ans: Nslookup (Name Server lookup) is a command for getting info. from the DNS Server.

It is a network admin tool for querying the Domain Name System (DNS) to obtain domain name (or) IP address mapping (or) any other specific DNS record.

It might be used to figure out what your name server is ?

If you give name — it will give you IP-address.

→ There is a concept called as Fully Qualified Domain Name → is a complete name of your service.

/# nslookup nginx-svc

Full name : → nginx-svc . default . svc.cluster.local

Address : 10.0.242.134.

name of your service

This speaks about namespace

↳ is a default domain name inside your K8s cluster

If you want → you can change that.

/# nslookup 10.0.242.134

O/p: It gives name of your service : nginx-svc.default.svc.cluster.local.
So, are these DNS entries.
So, you can access services by using its name (or) IP-address. Both are not going to change.

/# cat /etc/resolv.conf

↓
This consists of local info. about your domain names and name seavers.

O/p:- search default.svc.cluster.local svc.cluster.local cluster.local mea...pb.br.intenal.
clouddapp.net
1st domain.

Generally, it is trying to speak about domains which are known to you.

nameserver 10.0.0.10
options ndots:5

when you tried nslookup → it tries with 4 says it found with default...

IP address of your nameserver

Look into Environment Variables in alpine pod (Alpine was created post nginx service creation) :-

1st we created → nginx-rs
2nd → nginx-svc
3rd → alpine-pod.

/# printenv

O/p:- KUBERNETES_SERVICE_PORT=443

HOSTNAME=alpine
HOME=/root
NGINX_SVC_SERVICE_PORT=80 → this is the name of our service.
NGINX_SVC_PORT=tcp://10.0.242.134:80

* So, K8s will inject Environment Variables about your service into every Pod (container) that gets created after container creation.

That means
On K8s, after we create a Service → if we create a container (Pod) → it will have Environment variables related to service. But these are not dynamic. So for eg. if you remove the service these variables will not be removed. So relying on them is not a good idea, but K8s does this

* Look into Environment Variables in nginx pods (They were created prior to nginx-svc) :-

(Linux VM → which has kubectl)

Lets use other machine kubectl

$ kubectl get po

o/p alpine

nginx-rs-6gjfb

nginx-rs-zqxrd

Relying on Envt. variables is not a good idea. (It doesn't give dynamic info)

Relying on nslookup is a good idea (it gives dynamic info)

$ kubectl exec nginx-rs-6gjfb -- printenv

o/p:- HOSTNAME=nginx-rs-6gjfb

KUBERNETES_SERVICE_PORT=443

HOME=/root.

In this, case you seeing nginx-service? → No.

Bcoz, this (C)r was created before service's creation.

/# Exit.

* ClusterIP → gets (means Service?) an Internal IP address within k8s cluster. It can't be accessed to the outside world.

Google: kubernetes Services. (Publishing Services: Service Types

* Now, what I want to do is → I'm not interested in alpine pod.

So, I have some user who is outside → who wants to access nginx-service.

This person who is outside → how can be access nginx-service?

- External Communication using Kubernetes Service :-

* Some user external to k8s cluster wants to access nginx.

* For this what k8s tries to tell is → I have different types (Service Type)

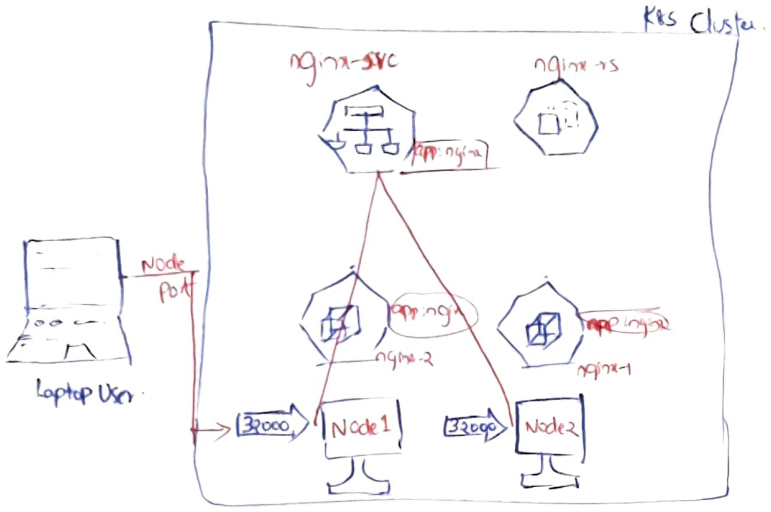
Right now, the only type which we know is → ClusterIP → which is an internal IP-address

→ k8s has the service publishing types:-

1) ClusterIP – for Internal comm?

2) Node Port → k8s will expose the application on a port on every node in k8s cluster

Now, when you access 32000 port ⟶ This basically connects to nginx-service



This is like Container Port-forwarding

But, in @ we had only one M/c.

Here we have multiple M/c's.

So, what we are trying to tell is

Your system needs to basically have a connection to one of the nodes and it would try to access the nodes on some port ⟶ and that would be accessing the app.

For this to work ⟶ your nodes need to have public-ip address. (if they want to be accessed from Internet.

But. (You don't want to expose the IP-addr of your node)

(This is useful within On-premises).

Till yesterday we were using Kubeadm ⟶ Remember all three nodes had public ips)

* Now
If you want to expose nginx-svc, you will be using node port.

Execise
↳ Try on kubeadm

* Load Balancer :-

This is generally used with managed K8s clusters.

Here, what we are trying to tell is. ⟶

Every cloud has some kind of Load Balancer (Layer 4 Load Balancer)

whenever you create service type as Loadbalancer ⟶

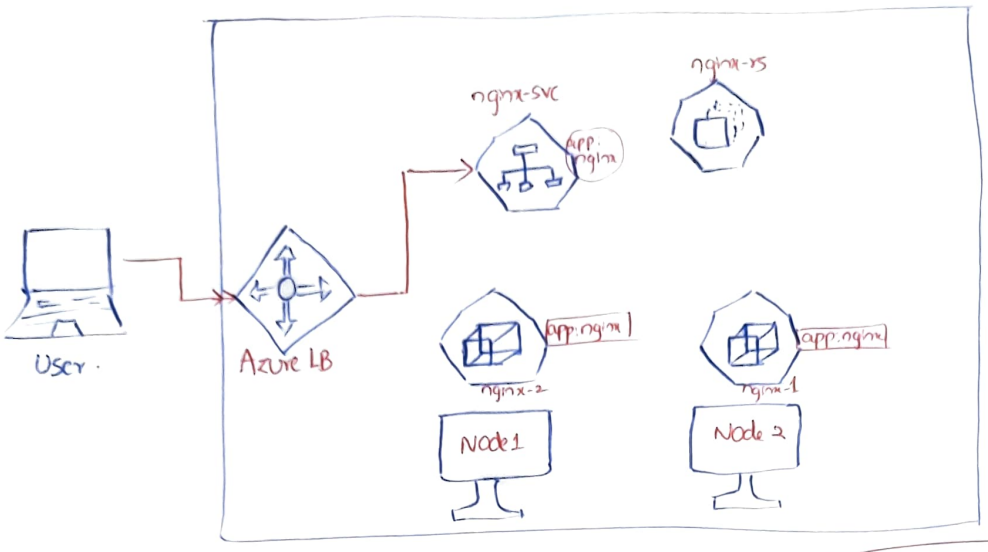
In case of Azure ⟶ it creates a Azure load balancer.

This job is done by Cloud-Controller Manager (CCM) (bcoz K8s by default will not have intelligence of azure)

Now you will be accessing load-balancer ⟶ which will forward the request to nginx-svc.
↓
also a load balancer

This is the 3rd type.

This is decent way. But there is a better way.



ClusterIP is private IP address

The type of service is :- Loadbalancer → This will not work in kubeadm as it doesn't have CCM. It doesn't know how to create a LB.

If you are using kubeadm, the only thing which you can resort to is nodeport (2nd type)

3rd type

Service-types

4th type: External :-

Here you get a CName record.

↓

CNAME record is something which you can put in DNS ~~records~~ servers.

When you basically put an external name → you get a cname → Add the entry to your DNS server which is maintained by your Orgn so that commun goes through.

(we use this rarely).

* Node Port and External name are used in on-premise K8S clusters a lot.

In the case of cloud → basically we tend to use Load balancers.

* Service types : 1) ClusterIP
2) nodeport
3) Load balancer
4) ExternalName.

'svc → mkdir lb-demo → nginx-lb-svc.yaml.

\svc\lb-demo :>

```
---
apiVersion: v1
Kind : Service
metadata:
   name: nginx-lb
spec:
  ports:
    - name: nginx-lb
      port: 80
      protocal: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer.
```

ports
↓
Service Ports.
↳ name
port → port that is exposed by this Service
↳ this will be loadbalancer's port.
target Port → is where your app^n is running in the ⓒ.

> Kubectl apply -f .\nginx-lb-svc.yaml.

> kubectl get svc -o wide -w

o/p - Name	Type	cluster-IP	External-IP	Port(s)	Selector
Kubernetes	clusterIP	10.0.0.1	<none>	443/TCP	<none>
nginx-lb	LoadBalancer	10.0.118.52	20.121.176.189	80:30087/TCP	app=nginx
nginx-svc	ClusterIP.	10.0.247134	<none>	80/TCP	app=nginx.

we got External IP over here.
It is actually IP-address of your LoadBalancer.
(Go to Azure → Load Balancer →)

Lets access using this IP-address

http://20.121.176.189

o/p: Welcome to nginx!

The port on which you'll be accessing app is 80.
If I have given 8080 → then it wud be
20.121.176.189:8080
which will map to 80 port of nginx.

* So, whatever applications you have written → let it be eg: nopcommerce, game of life, spc etc

Can we expose it to outside world now ? → Yes.

All you need to do is

→ If you are using Kubeadm → use nodePort.

If you are using Azure AKS → use type as LoadBalancer.

This is not a better way.

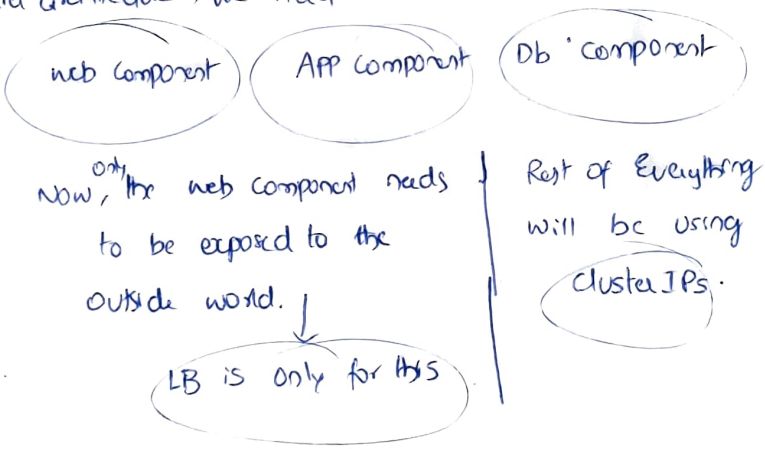
There is something more which we are going to learn.

In that we will try to access the app^n in a much better way.

For now, this is a Layer-4 L.B.

(Layer 7 is better than Layer 4).

* In n-tier architecture, we had



* Layer 4 & Layer 7 are OSI Layers.

→ I don't want to put my Pod in running state until its dependency is up.

① (I have app server & I have db server → I don't want to put my app pod in running state till db is up).

② In Pods, I'm running my apps. Apps might fail. → So when it fails, what shud we be doing ? How shud we be configuring our checks ?

Deleting the Cluster → $ az group delete --name myResourceGroup --yes --no-wait.

L-56-Apr-29-Part-2 | Health Checks/Probes for C's in Pods, Container Types in Pods

2 tabs

1) Powershell → Laptop 2) linux kubectl.

$ ssh ubuntu@52.25.224.255 (Azure VM)

↓

git kubectl

→ azure aks quickstart (lets quickly create k8s cluster).

$ az group create --name myResourceGroup --location eastus.

$ az aks create -g myResourceGroup -n myAKSCluster --enable-managed-identity --node-count 2 --enable-addons monitoring --enable-msi-auth-for-monitoring --generate-ssh-keys.

We have created nginx-rs & nginx-pods. Label → app:nginx, version → 1.23

↓

4 replicas.

I have created a service. (Label: app:nginx, version: 1.23)

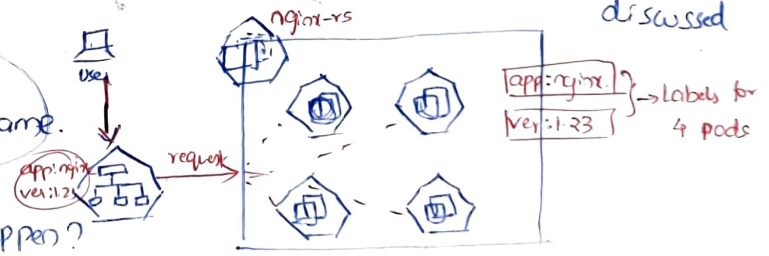
Now

* whoever wants to make a request. (Someone is raising the request).

* k8s service will forward the request to 4 of them. (This is what we discussed)

Now what has happened is?

Our applications' new version came.



* So right now – what wud happen?

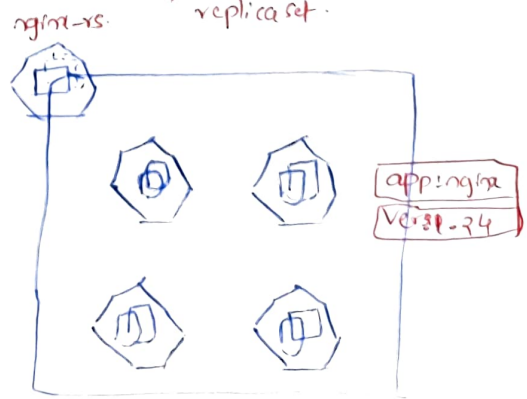
Your nginx-sv → will try to forward the request to 4 of these 4.

Then, what we have done is

↓

we have updated our replicaset

updating replicaset.



* Initially what wud happen is

src ↳ it will forward the request to upper (1.23)

Now, when you update and you have a label (nginx 1.24), all of these Pods (1.23) will be dead

Now if someone accesses k8s Service → it will not be able to forward the request bcoz → when you update (all the pods are gone).

You don't stop older Pods (on replica set.)

You create one more rs

So, 1.23 version → there is no pod.

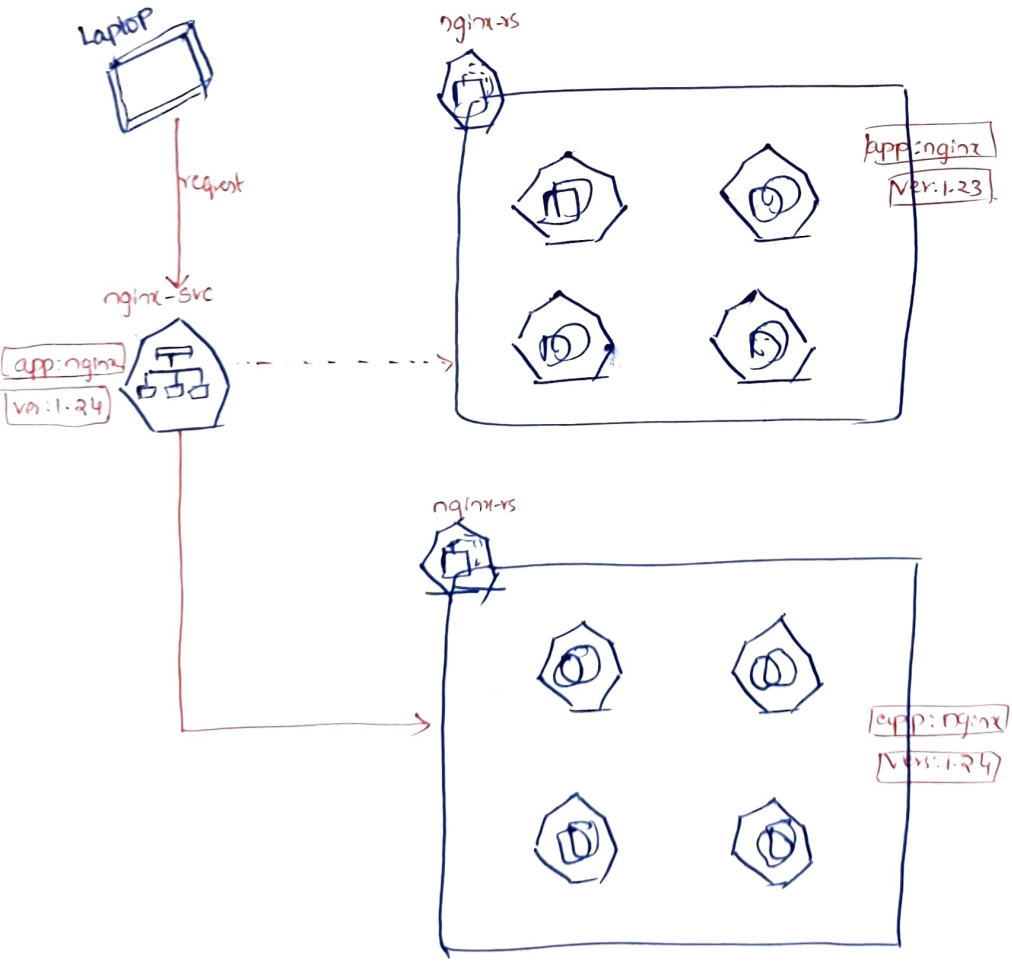
You will set the label in such a way that it will forward the request to older ones as well as new ones

Once if you think everything is working → replace it with version in svc.

Add label ver:1.24 to a k8s-Service. (then previous link will be broken).

*Generally, whenever you want to update to a new version, without having a downtime, this is the approach we follow

* when upgrading to newer versions of Pods, ensure right set of labels are present on k8s service (Selector).



L

* But, the problem is, if your application is running inside K8s → that means we are running appln inside pod → Indirectly, you are running appn inside the Container.

But is it guaranteed that every appn inside the Ⓒr will work without any problem? → NO.

- So, when your appn is not working → K8s shud know that. So, how will K8s know the component → what is the K8s component which repots the status of Something running on the worker node to master node?

↓

Component is → Kubelet → shud know what is happening in your pod.

for that, let us try to understand

↓

* Health Checks / Probes for Ⓒrs in K8s Pods :-

Configure kubectl to connect to your K8s cluster using az aks get-credentials command.

```
$ az aks get-credentials --resource-group myResourceGroup --name myAKSCluster.
```

y

y

```
Dell> az aks get-credentials  "  "  "  "
```

y

y

y

Google: health checks in K8s.

- So K8s supports 3 Kinds of checks

1) Liveness Probe
2) Readyness Probe
3) Startup Probe

For every K8s Ⓒr → we can try to write all 3.

The basic idea is ⇒ liveness is used for the K8s to know when it shud restart the Container.

we remember that when K8s goes into exited state → K8s will restart the Ⓒr.

- **Liveness Probe :-** Liveness is used for the K8s to know when it shud restart the C^r.

But, let us assume that, your app^n has become too much busy, some process has become still → restarting the C^r might solve the problem →

So for that, if you try to write a liveness probe (livness check) and then if that check fails → we want K8s to restart the C^r.

That is exactly what liveness Probe is all about.

- **Readyness Probe :-** Readyness Probe → what it does is → if Readyness Probe is passing, then only your service will forward the request to the Pod.

If Readyness check fails, your C^r might be up & your pod might be up → but it will still not forward the request.

K8s Service will not consider – your pod – to forward the request.

until & unless that check becomes passed.

Kubernetes has recently introduced →

- **Startup Probe :-** is just to figure out whether everything in your application is started (or) not.

Till Startup Probe is finished, it will not check for liveness & readyness.

Liveness Probe: If this check fails, K8s will restart the C^r.

Readiness Probe: If this check fails, the Pod will removed from Service (Pod will not get requests from service).

Startup Probe: This checks for startup and until startup is ok, the other checks will be paused.

It is always a good idea to write Health checks for your C^rs → b cos you can tell when your App^n is ready, when your App^n has started and when you think that you shud restart your App^n.

- To do these Health checks,
  - API reference → Pod v1 Core → PodSpec → Containers → Probes :- exec, grpc, httpGet

* Probes (or) checks can be performed by

- Exec : run any linux/windows command which returns status/exit code.

we all know that if exit code is zero (0) → it is success.

If it is anything other than zero → it is failure.

So, to know the health of your app — if you have to run some command — that is what you do using exec.

- http : We send http request to the app.

Any http requests will give you status codes (3 digit no.)

Based on status codes we can decide.

Google: http status codes.

Starts with 1 → i.e. Information
" " 2 → Success
" " 3 → Redirection
4 → client-side Error
5 → Server-side Error.

- Grpc : is also one of the communication mechanism

This communicates over grpc.

In K8s, if you are trying to run a pod which is a micro-service developed work — which exposes grpc — then you wud be using this grpc to figure out whether your app is healthy (or) not.

- tcp :- send tcp request.

~~rs~~

folder → healthchecks folder → httpd.yaml

To cross-check

Let us try to create any simple app →

FIR → Network

When you type → fb.com

it goes to facebook.com.

Status code : 301 (redirection)

There is nothing called as fb.com.

Eg: directdevops.blog → status 200 ✓ success

Eg: directdevops.blog/testing1234

↓

Error → 404.

(client-side Error).

- Now, we will be writing multiple yaml files in one file.

Here we will create 2 specs { Service, replica set.

--- httpd.yaml (Service file)

apiVersion
↳ v1

metadata
name → httpd-svc

Spec
ports (Service Port)
↓
port (port that will be exposed by this service) → 80.
name → httpd-svc
target port → 80.

selector
app: httpd.
type → LoadBalancer.

(You can use clusterIP & use curl requests for checking)

replica set
↳ 2 replicas of httpd:-

version → apps/v1
Kind → ReplicaSet
metadata
name: httpd-rs
labels:
app: httpd.

spec:
↳ replicaset Spec.
↓
minReadySeconds: 1
replicas : 2.
selector:
matchLabels:
app: httpd.

template:

metadata:
name → httpd
labels:
app: httpd.

spec:
containers:
- name: httpd
image: httpd:2.4.57
ports:
- containerPort: 80.

livenessProbe
Type (Probe)
↓
click on
exec (or)
grpc

failureThreshold = 3.
→ This check is considered to be failed when it fails 3 times.
↓
we can change that no → to 1 (can't put 0)
Default.
After success
(It will wait for 3 consecutive failures)

initialDelaySeconds = 1
→ when do you want to start probes (Immediately once the container has gone into running state (or) after some time)

PeriodSeconds = 10 → default
How frequently do you want to do the probes

Do you want to start

successThreshold = 1

↓

how many checks shud it pass to consider it a success.

Default = 1.

timeoutSeconds = 1

↓

Once you make this check → by what time do you expect your check to get response

(you have sent a request ↓ " response to come

Default = 1

httpGet

(HTTPGetAction)

↓

click onit.

↓

where I shud get 200 as code

There is no need for you to give host as you are taking → local host.

Path : / ⟶ means homepage.

Port : 80. → on which our appln is running

For some reason if this appn goes for a toss → Now K8s will try to restart.

---

readiness Probe :-

(Probe).

write same thing you have written in live probe as both are of same type.

- we will write failure scenario

↓ writing

exec :

command

- sleep

- 10MM → error → failed scenario.

→ in livenessprobe instead of httpGet : Path : / port : 80

Exec action 2nd method

and try to run some linux cmd which will check whether apache process is running or not.

If apache process is running → it will give return code zero (otherwise 1).

3rd one is → Sending tcp requests

4th " → Sending grpc requests.

Lets try to run this

docker playground.

↓

new instance

- docker container run -d -P --name apache httpd:2.4.57

docker container ps

O/P: ... httpd:2.4.57 ... Ports 0.0.0.0:49153->80/tcp

$ curl http://localhost:49153

O/P <It works>

we are not interested in response.

we are interested in status code.

Google: curl examples. (linux cmd)

curl to print status code.

$ curl -I http://localhost:49153

O/P: HTTP/1.1 200 OK. ✓

(or)

$ docker container exec apache

> cd Manifests\rs\healthchecks

> Kubectl apply -f . → If you give folder name — it will try to run everything in that folder.
                                ↓
                        we have 2 files.

o/p: pod/alpine created
     service/httpd-svc created
     replicaset.apps/https-rs created

> kubectl get po

		Status	Restarts	Age
o/p: alpine	1/1	running	0	69s
http..	1/1	"	2	"
http...	1/1	"	2	"

→ here we gave timeout seconds=10
↓
we are getting restarts (due to failure) → change to 1.

> Kubectl apply -f .

o/p: nothing changed.
     Only rs -configured.

> Kubectl get po -w

o/p:	alpine	1/1	running	0
	http	1/1	"	3
	"	1/1	"	3

> Kubectl describe pods https:..

o/p: Logs → will speak about Liveness
                              Readiness
     Events.
       ↓ will show → Liveness probe has failed.
                       ↓
                  so I'm supposed to restart my app?. (Actually my © has not gone into exited state but still I need to restart (bcoz liveness check-failed).

Now replace with

```
httpGet:
  path: /
  port: 80
```

in liveness.probe.

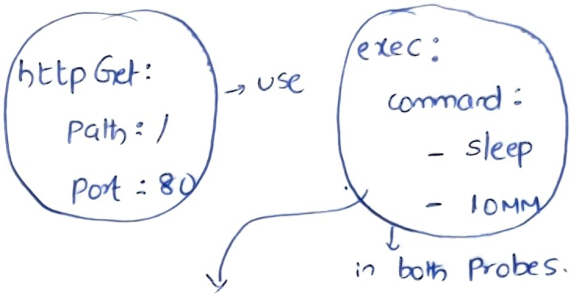
> Kubectl apply -f .
> Kubectl get po -w
o/p: From now restarts shud not happen
bcoz Our checks will pass now.

httpd.yaml :-

```
---
apiVersion: v1
Kind : Service
metadata:
    name : httpd-svc
    labels:
      app: httpd
Spec :
  ports:
   - name: httpd-svc
     port : 80
     targetPort : 80
  selector:
    app : httpd
  type : LoadBalancer
---
apiVersion: apps/v1
Kind : ReplicaSet
metadata :
   name : https-rs
   labels :
     app : httpd
spec :
  minReadySeconds : 1
  replicas : 2
  selector :
    matchLabels:
      app: httpd
  template :
    metadata:
      name: httpd
      labels:
        app: httpd
spec :
  containers :
    - name : httpd
      image : httpd:2.4.57
      ports :
        - containerPort : 80
      livenessProbe :
        failureThreshold : 3
        initialDelaySeconds : 1
        PeriodSeconds : 10
        successThreshold : 1
        timeoutSeconds : 1
        httpGet :
          Path : /
          port : 80
      readinessProbe :
        failureThreshold : 3
        mitialDelaySeconds : 1
        PeriodSeconds : 10
        successThreshold : 1
        timeoutSeconds : 1
        httpGet:
          Path : /
          port : 80.
```

http-faulty.yaml :-

Same as previous file except → instead of



Replica-set name : https-rs-faulty.

what wud happen?

Here we are running 2 pods → (we already have 2 healthy Pods → httpd.yaml)

Generally when you write this, service shud actually serve the 4 pods. It shud forward the request to 4 pods.

We have said in Service → selector → app: httpd.

So, here also in pods we have app: httpd (httpd.yaml)

whatever is faulty → there also i have app: httpd (http-faulty.yaml)

→ But, the point is, these pods shud not be served bcoz, Readiness is failing

→ Ideally, your service shud forward the request to 4 pods. But it will be serving to only two.

> Kubectl apply -f .

o/p: Required value → must specify a handler type. (Actually we are not worried about Liveness Probe. we want service to only serve 2 Pods by failing → 2 pods in faulty.yaml)

> Kubectl get svc

	ClusterIP	External-IP
o/p: httpd-svc		20.88.179.96
kubernetes		

> Kubectl describe svc httpd-svc.

o/p: ...

Endpoints: 10.244.0.10:80, 10.244.1.12:80.

...

we are seeing Endpoints.

Endpoints are your (o)r IP-addresses → your service is serving.

when we create service, end points are created

↓

End pt represents each (o)r (or) pod's IP-address.

* So, totally we have 4 Pods which have the matching labels

But totally, how many is it serving ? = 2. (bcoz in other 2 (faulty) → Readiness failed)

* Now to correct the http-rs-faulty.yaml → replace

exec:
  command:
    - sleep
    - 10MM

with

exec:
  command:
    - pwd

↓ will pass.

> kubectl apply -f .

> kubectl describe svc httpd-svc

> kubectl get endpoints

After changing exec and apply → wait for 30 secs

because → In probes → we have written success thresholds & failure thresholds. (wait for checks to be done)

Periodseconds = 10s

exec:
  command:
    - exit "1" } means return with Error code of 1

For a check to fail

↳ it shud fail 3 times after a success.

Threshold for failures is 3 after success

3 checks = 3 x 10s = 30sec.

- start-up check :-

Not many people use this -

The basic idea of startup probe is for you to tell when your application is started.

(Liveness probe is all about issues that may be resolved by using restarts)

* From now on, whenever we write Pods, we will try to write some necessary probes.

• Run Pods with Specific Resources (CPU/Memory) :-

So, How much of CPU & Memory are you speaking about ?

→ I want to run 10s in my pod with specific resources → I want to give them what is the minimum CPU which it shud consume & what is the maximum? what is the minimum RAM it shud consume & also Max

bcoz → lets assume that → On a historical basis, our Orgn is basically running a M/c with 4 CPUs & 16 GB of ram and there we can run 100 Pods.

But some appn is misbehaving and using too much of CPU. So → just bcoz that appn is behaving and too much of CPU → You are impacting other apps bcoz CPU of that appn goes for a toss.

How to limit that?

Exercise

Try running a docker container with 256 mb of ram & 1 CPU.

Run any ©r. → Open Dr playground & figure out a command for that.

Type the command → docker stats.

Ans: --cpus

--memory.

Eg: docker run -d -p 8081:80 --memory="256m" nginx

docker run -it --cpus=".5" ubuntu /bin/bash

↓

If you have 1 cpu, the command guarantees the ©r at most 50% of the cpu every second.

(or)

docker run -it --cpu-period=100000 --cpu-quota=50000 ubuntu /bin/bash

---

Google: Resource Management for Pods & ©rs

(Manage Resources - containers)

Here, in K8S, whenever you want to put restrictions for resources' usage → there are 2 fields → 1) Requests → is the minimum amount which you want.

2) Limits → is the maximum amount which your pod can go.

- for Eg: If I want to run a Pod → I want to give min of 256 mb, max of 1 Gb.

↓ will go in Request

↓ will go in Limits.

> Whenever you are running a ©r in a pod. It is always a good idea to put restrictions

You can also skip one. (Eg: You can write only limits)

- These will be Part of Container Spec.

→ rs folder → limits folder → nginx-rs.yaml :-

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
spec:
  minReadySeconds: 1
  replicas: 3
  selector:
    matchLabels:
      app: nginx
```

selector

```
template:
  metadata:
    name: nginx-pod
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1-23
        ports:
          - containerPort: 80
        livenessProbe:
          httpGet:
            path: /
            port: 80
        readinessProbe:
          httpGet:
            path: /
            port: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "256Mi"
            cpu: "1000m"
```

---

continued ↗

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    app: nginx
  type: ClusterIP
  ports:
    - name: nginx-svc
      port: 80
      targetPort: 80
      protocol: TCP
```

> kubectl delete -f .

1000m ≈ 1 CPU
250m = 1/4th of 1 CPU

Writing a Probe & Writing a resource limits } is always a good Practice
bcoz you will never be out of resources when you scale. (bcoz you have limited resources)

>cd limits
>kubectl apply -f .
>kubectl get Po
>kubectl describe pods nginx-rs-kqbnt

O/P: Limits:
cpu: 1
memory: 256Mi
Requests:
cpu: 250m
Memory: 64Mi;

Containers Types in Pods :—

- In Pods, we have 3 types of Containers
  1) Containers: these are why we write Pod spec
  2) init Containers
  3) ephemeral containers.

Google: init container kubernetes.

>kubectl delete -f .

→ Init Containers :-

When K8s starts Pods → In Pods you are starting C^rs

- First K8s. it will start init C^rs.
- Init C^rs are the C^rs which shud not run forever. They shud be Completed

If you remember Job → Job is a C^r which basically finishes in some time.

So Init C^rs are the C^rs where you do Pre-condition

→ For Eg: for running your application, if you basically need certain dependency, other C^r shud be up(on) not.

Till other C^r is up → don't start the C^r. If this is your use-case — then what we will try to do is →

we will try to put a init container in which we will wait for the other service to be up.

↓

Only after the other service is up → then we will finish the init C^r.

Once we finish the init C^r → then your actual C^rs will start in which your appn runs.

- So, the basic flow is :

  Pods will create init C^rs.

  In init C^rs you can write 100 containers if you want → but it will run one after other. It will not try to start them in parallel.

(In pod → if we have created 2 C^rs → it will start 2 C^rs at the same time)

(But init C^rs are not like that).

- Till init C^rs have reached its success state → your actual C^rs will not be started.

* If your appn has some pre-conditions to be met

↓

so, 1st, we'll be writing those pre-conditions as init C^rs.

$ dig google.com

This dig command will basically try to run and it will try to tell you what is a query time and some stuff.

But if the service is not running.

↓

$ dig google1.com → Here, it will basically try to give you Error state

Eg: for i in {1..100}; do sleep 1s; if dig myservice; then exit 0; fi; done;

ds folder → init containers

├ nginx-rs.yaml (rs & svc)
└ initcontainerpod.yaml :—

Pod
↓
PodSpec
↓
→ Containers
→ ephemeral C—
→ init Containers
(type: Container Array).

```
apiVersion: v1
kind : Pod
metadata :
   name : init-demo
Spec :
  containers :  ——————> my actual container over here.
  - name : httpd
    image : httpd
    ports :
      - containerPort : 80
  initContainers :
  - name : delay
    image : alpine
    command :
      - sleep
      - 15s
  - name : init-myservice
    image : busybox:1.28
    command :
      - 'sh'
      - '-c'
      - "until nslookup nginx-svc.default.svc.cluster.local ; do echo waiting for myservice;
                                                              sleep 2; done"
```

Over here
what is my service's name?
↓
nginx-svc.

Let me give something which doesn't exist →

I'm trying to run a script. → what this script will do is → it will try to wait till this service is up. and only when then service is up → then your actual containers will be created.

docker playground
↓
>ssh...

$ docker Container inspect apache

o/p : - -
- -
- -
IP address 172.17.0.2

$ docker container run -it alpine /bin/sh

/# for i in {1..100}; do sleep 1;
if dig 172.17.0.2; then exit 0; fi;
done; exit 1.

o/p: dig not found.

/# apk add dig bind-tools.

/# for i in ...
✓ ✓ = It is failing
If it exits out of this → it's Good.

$ docker container run -it alpine/bin/sh
/# for i in {1..100}; do s... 172.17.0.10.

1st it will sleep for 15 seconds and then it will wait for other service to be up

and then

Once that is done → our actual container will start.

> cd initcontainers

> Kubectl apply -f .

O/p: pod/init-demo created

replicaset.apps/nginx-rs created

Service/nginx-svc created

> kubectl get po -w

O/p:		Ready	Status	Restarts	Age
	init-demo	0/1	Init:0/2	0	9s
	nginx-rs-ct.	1/1	0	0	8s
	nginx-rs-gf-	1/1	0	0	9s
	nginx-rs-w7	1/1	0	0	8s
	init-demo	0/1	Init:1/2	0	16s
	init-demo	0/1	PodInitialising	0	17s
	init-demo	1/1	Running	0	18s.

→ Actual C^r started.

→ In Init it has to finish 2 containers (15 sec — 1st, nslookup.)

→ It has not gone into running state immediately. It is waiting for 2 init C^rs to finish its execution then it will go into Pod-initialization phase ↓ then, it will try to run your C^r. ↓ httpd

> Kubectl delete -f .

> Kubectl apply -f .

> Kubectl describe po init-demo

(Eg:- You shud never start app server without db server being up)

we use init C^r.

- Ephemeral Containers :-

These C^rs basically don't have any functionality.

If your pod is continuously failing → and if you want to debug (or) if you want to figure out what is the reason - why your C^r is failing?

But, the point right now is → your C^r is in exited state → You want to figure out the reasons → that is where Ephemeral C^rs come into play.

Google: Ephemeral Containers → new feature of k8s (added in 1.25)

This is for troubleshooting.

Troubleshooting k8s
- we will debug pods using Ephemeral Cs.

So,

* We know about Pods.

In Pods we have 3 types of Cs — 1) Normal Cs
2) Init Cs
3) Ephemeral Cs

Pods basically have labels.

For every C I can set the limits.

I can check whether my app is running or not by trying to write Probes.

Node Usecases :-

* Till now our focus was on Pods & Cs inside the Pod.

* Let us assume that, you have Pods which run on your nodes.

You want to run a pod on a Specific node. (Till now we didn't worry about it).

Use Case :-

→ In your Orgn there are some apps which require GPUs not CPUs.

Probably there might be some gaming server that is running inside the C which requires IIrd processors — GPUs.

So in your k8s cluster → there are 2 types of nodes (one which have CPUs, other which has GPUs)

when you are running this particular pod, that needs to go and run on any node which has GPU — (not CPU). → How do I do that?

* Remember Kube-scheduler

Till now we were trying to focus on Kubelet's functionality. How can I tell Kubelet that your C is not working. How can I tell Kubelet to restart the Pod? This was what our focus was majorly on.

Now I'm trying to speak about Schedulers.

1) How to schedule a Pod on a Particular Node.?
2) How to stop assigning more Pods to a node.?
3) How to move all the pods running on a node to other node?

Lec-57 - App-30 Part-1 | Deployment, DaemonSet, Annotations, Scheduling Pods, Headless Service

Deployment → Rollout, Rollback

(Cluster - Delete)

```
az aks delete --resource-group
MyResourceGroup --name MyManagedCluster
```

```
az aks delete --resource-group
MyResourceGroup --yes --no-wait
```

• Deployment :-

→ Deployment creates a replica set.
↓
creates Pod.

The basic thing which (D)t can do is → rollout & undo rollout

So, what do I mean by that ? <------

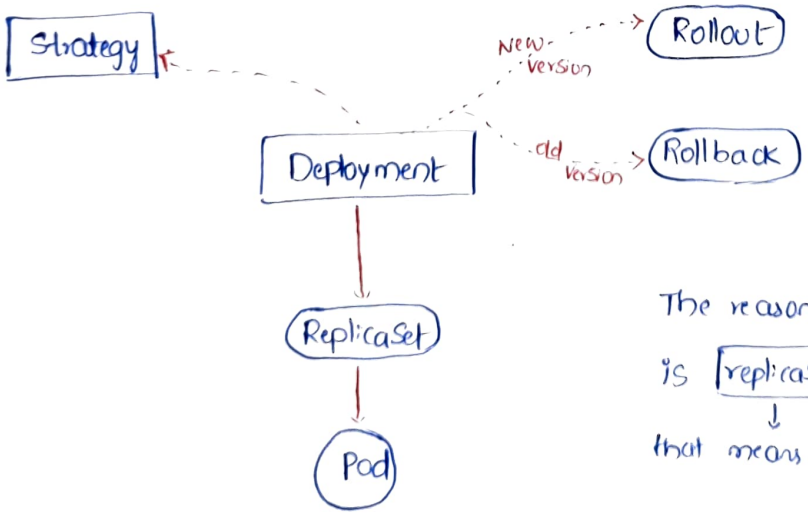
So, whenever a new version of your app^n comes → I can deploy.

When I deploy → there are strategies over here which can make your app^n (D)t to be zero-downtime. → we can have a zero-downtime (D)t. (zd-(D)t)

Now,
* If the new version of your app^n is not working → we can go back to the previous version.

→ (D)t is a K8s object which can help in rolling out and rolling back updates.

→ (D)t controls replica set and replicaset controls Pods.



- Rollout is giving a new release
- Undo Rollout ≃ Roll back
  ↓
  is giving a older version.

The reason why rs is diff-t from rep^n controller is replicaset can be persistent

↓

that means that → (D)t tries to maintain your replica set in etcd cluster

Now, whenever you update, it stores the newer version of replica set

↓

which is not possible using replication controllers.

* So, there are two things which are different when it comes to replicaset and replication Controllers :-

1) Replicaset can do set-based Equality when it comes to pods.

2) Replicaset is used in Dt which is further used for persisting the previous versions.

So, Dt uses replicaset ⟶ & in rs → we try to write pods.

- One more imp thing in Dt is strategy.

So in Dt, this strategy tries to ⟶ how do you want to do Dt?

Do you want to have downtime (or) do you want to have zdt?

↓

That is defined by this strategy.

→ Apart from that, in Dt we are doing - basically calling replicaset and

↓

Lets try this: Lets create a manifest with some application deployment :-

> mkdir deployments ⟶ nginx-svc.yaml.

nginx-svc.yaml :-

```
---
apiVersion: v1
kind: Service
metadata
   name: nginx-svc
spec:
  selector:
    app: nginx
  ports:
    - name: nginx-svc
      port: 80
      targetPort: 80
      Protocol: TCP
  type: LoadBalancer
```

* Now lets try to create → nginx-deploy.yaml

Since if you try to go from one version of nginx to other version of nginx, you'll not be seeing much of a difference → 1st I'll try to create a replica-set with image which is nginx → then I'll change the image to httpd → so that you can see the homepage difference.

→ API-reference → Deployment v1 apps

↓

apiVersion → apps/v1.

Kind → Deployment.

name → nginx-deploy. (you can also add labels).

spec

↓

Deployment Spec → will contain a template → that template is Replica set's template.

→ minReadySeconds = 1

replicas = 3

Selector :

[redacted].

Type: LabelSelector → matchLabels

app : nginx.

strategy

↓

Type: - DeploymentStrategy → rollingUpdate.

type → recreate (or) rollingUpdate → is default.

type : RollingUpdate.

rollingUpdate :- maxSurge

maxUnavailable.

recreate

↓

means it will delete the existing pod and then recreate the new one

↓

wherever you put → your type : recreate → you are having a down time.

How many replicas did we set? = 3.

For some time, let us assume that there are 3 pods which are of old version.

Now I'm trying to create a new version.

[redacted], whenever you are trying to tell strategy → You are trying to tell → How do you want to do this deployment.

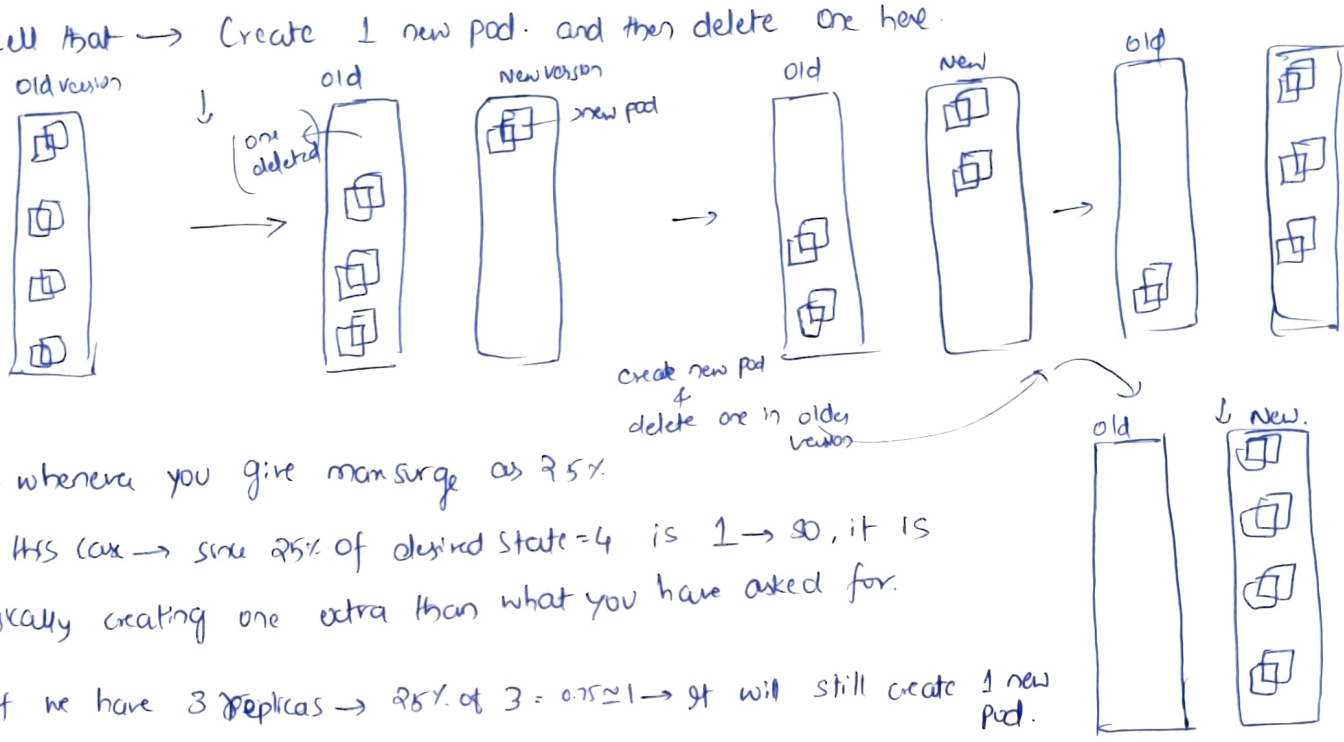
- So, basically we have 4 old Pods (replicas = 4)

Now maxSurge ↓ means

How many no. of pods (or) % of pods which I can create above desired state. ↳ = 4

Bcoz I want new pods → to have new version.

So, let us assume that I have put maxSurge = 25%. → that means, I'm trying to tell that → Create 1 new pod. and then delete one here.



So, whenever you give maxSurge as 25%

In this case → since 25% of desired state = 4 is 1 → so, it is basically creating one extra than what you have asked for.

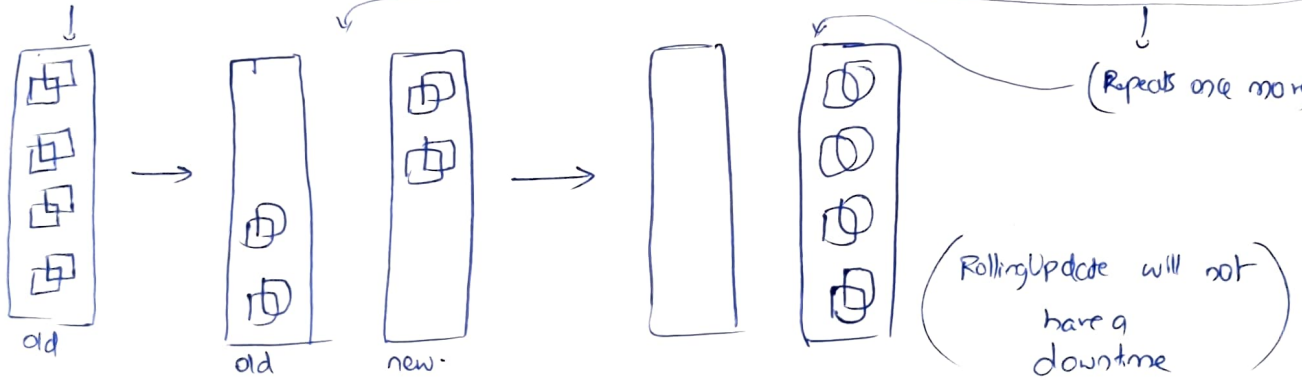
So, if we have 3 replicas → 25% of 3 = 0.75 ≃ 1 → it will still create 1 new Pod.

→ Let us assume → I have given max surge as 50%.

what it would do is

In this case → rather than creating 1 new Pod → it will create 2 at a time. Once those 2 are in running state, it will delete 2 in old.

bcoz → 50% of desired state = 2.



If it is recreate → 1st it will delete the old Pod and then it will create a new one

→ that might have down time. (bcoz recreate is not one by one → it is everything at once)

maxUnavailable :-

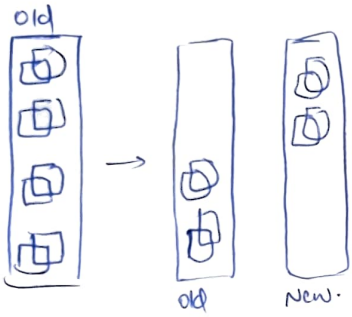
Whenever you are doing this (D)^t → how much % of pods can be unavailable.

So, let us assume that

maxSurge → 9 gave 50%
(i.e. 2 pods at a time)

maxUnavailable → 9 gave 25% = 1 pod. (Only 1 pod can be available)

So what it will do is



It will create 2 new Pods & it will not delete 2 at one shot.
It will delete only one → bcos only 1 pod can be unavailable. and then it will delete 2nd pod.

That is how this (D)^t strategy works.

maxSurge → default no. is → 25%.
maxUnavailable → 25%

Here, you can write % (or) numbers directly.

→ Next template (Pod template).

```
metadata:
  name: nginx
  labels:
    app: nginx
    ver: "1.23"
spec:
  containers:
    - name: nginx
      image: "
      ports:
        - containerPort: 80
```

But I'm not using ver in svc, as I might have 2 version.
So in my svc → I have only app: nginx - so I might get new version also. (so version can change).

- nginx-deploy.yaml :-

```
---
apiVersion: apps/v1
kind : Deployment
metadata :
  name : nginx-deploy
spec :
  minReadySeconds : 1
  replicas : 3
  selector :
    matchLabels :
      app : nginx
  strategy :
    type : RollingUpdate
    rollingUpdate :
      maxSurge : 25%
      maxUnavailable : 25%
  template :
    metadata :
      name : nginx
      labels :
        app : nginx
        ver : "1-23"
    spec :
      containers :
        - name : nginx
          image : nginx
          ports :
            - containerPort : 80
```

* Google : kubectl cheat sheet

kubectl deploy commands. (bluematador).

Exercise :-

Try writing deployment spec for a Jenkins app with 8 replicas and 50% surge & 50% unavailable.

Eg: jenkins/jenkins:lts-jdk11

$ kubectl get nodes

2 nodes ✓

> cd deployments

> kubectl apply -f .

O/p: deployment.apps/nginx-deploy created

service/nginx-svc created.

> kubectl get svc

O/p:

				External-IP
Kubernetes				
nginx-svc	LoadBalancer	10.0.114.92	20.232.81.56	

Go chrome

http://20.232.81.56 ⟶ Welcome to nginx!

* Lets get deployment information:

$ kubectl get deploy ⟶ short name.

O/p: nginx-deploy 3/3 3 3 .

$ kubectl describe deploy nginx-deploy.

O/p: Name : nginx-deploy

Namespace : default

Annotations : deployment.kubernets.io/revision: 1

Selector : app=nginx

Replicas : 3 desired | 3 updated | 3 total | 3 available | 0 unavailable

StrategyType : RollingUpdate

RollingUpdateStrategy : 25% max un- 25% max Surge.

$ kubectl get rs

		Desired	Current	Ready
O/p:	nginx-deploy-7f...567	3	3	3

I have created a (D)F. (D)t created rs object.

↓ Controls Pods

$ kubectl get po

O/p: - - v1 - -
- - v1 - -
- - v1 - -

Lets explore rollout Command :-

$ kubectl rollout --help

O/p: history → View rollout history
pause
restart → restart a resource.
resume
status → show status of rollout
undo → Undo a previous rollout

Refer here for the manifests used to create version 1 (or) revision 1.

$kubectl rollout history --help

$ kubectl rollout history deployment/nginx-deploy

O/p

Revision	change-cause
1	<none>

It says that you have only one version right now.

I have applied only once.

$ kubectl rollout status deployment/nginx-deploy

O/p: deployment "nginx-deploy" successfully rolled out.

→ Lets update the specs to change image from nginx to httpd :-

Go to nginx-deploy.yaml → chang image to httpd.

and minReadySeconds = 5 (for our Convience)

↓

that means - this Pod will come into running state after 5s.

> kubectl apply -f .

$ kubectl get deployments.apps -w.

we can see that it has increased by 1 and

then it deletes the older version.

$ kubectl rollout undo --help

You can just give current one.

But if you want to go to the specific revision

Currently we have 2 revisions.

--to-revision=54

For some reason:
you have 100

If you want to go to 54

$ kubectl rollout undo deployments/nginx-deploy

O/p: rolled back.

At the same time do

> kubectl get deploy -w

	Name	Ready	Up-to-date	Available
O/p:	nginx-deploy	3/3	3	3
	"	3/3	3	3
	"	3/3	3	3
	"	3/3	0	3
	"	3/3	0	3
	"	3/3	1	3
	"	4/3	1	3
		4/3	1	
		3/3	1	
		3/3	2	

While this is happening ←

↓

Try to access

http://20.232.81.56

↓

O/p: Welcome to nginx.

So, my app was — not down.

I was using my app continuously and then it moved to the older version.

$ kubectl rollout history deployments/nginx-deploy

O/p:

Revision	Change-cause	
2	<none>	Now it is not showing 1.
3	<none>	Bcoz 1 & 3 are same.

* Now, lets try to create one more (D)^r → image :- phpmyadmin.

$ kubectl get deployments.apps -w | > kubectl apply -f.

maxSurge : 50%.

maxUn : 50%

O/p:

5/3

5/3

20.232.81.56 → Now it is ~~failing~~ Working as php doesn't support

o/p:

Name	Ready	Up-To-Date	Available	Age
nginx-deploy	4/3	2	3	8m 16s
nginx-deploy	4/3	2	4	8m 19s
" "	3/3	2	3	8m 19s
" "	3/3	3	3	8m 19s
" "	4/3	3	3	8m 20s
" "	4/3	3	4	8m 25s

in the meantime

In chrome

http://20.232.81.56 → httpd → It works!

So, am I able to access the app^n while (D)^t is happening?

YES.

$ kubectl get rs

o/p: nginx-deploy-787...959
nginx-deploy-79d...567
} 2 replica sets

one refers to previous version when we (1st (D)^t) have done ↑

2nd one → since I gave new version it created new rs.

$ kubectl get po

o/p: newer versions of pods.

$ kubectl rollout history deployments/nginx-deploy

o/p:

Revision	Change-Cause	
1	<none>	→ 1st one was with nginx
2	<none>	2nd " " httpd.

$ kubectl rollout status deployments/nginx-deploy

* for some reason → let us assume that → you have given a new version → and it is not working acc. to your customer's expectations → then what we wd want to do is → we wd want to go back to the previous version.

→ Now to roll back to previous versions :-

$ kubectl rollout --help.

o/p: Undo.

It is ~~failing~~ working. So what shud we do? → Go back to previous version.

> kubectl rollout history deployment/nginx-deploy

	Revision	change-cause
o/p:	2	<none>
	3	"
	4 → only name (php) – sir's mistake.	
	5 → image also	

> kubectl rollout undo deployment nginx-deploy --to-revision=2

o/p: rolled-back.

So I'm able to move from one version to other.

> kubectl rollout history deployment nginx-deploy

o/p: 3
4
5
6

So, Everyday from Jenkins, whenever we get a build in the night → we will be doing kubectl apply and we will get a new version → which your testers will basically be using.

If that (new version) is not working → all you need to do is kubectl rollout undo → so that it will go back to the previous version → which is Yesterday's version ↓ which is working.

What is change-cause?

It is showing as none which is not good.

~~while applying~~

what can be done to have a valid change-cause?

So, change-cause cud be as simple as → I have updated to this new version bcoz I got a new build on monday (or just write date here ↑).

(or) just write some info. ↓ I want to write some info

For that I need to do certain things. → which cud be → making some arguments to be passed ~~while~~ using kubectl apply command.

Annotations :-

We spoke about Labels. Labels are → which are used by k8s for selections. There is one more key-value pair in k8s → but this is not meant for k8s → this is meant for tools which are using k8s → to add some logic.

→ For Eg :- Yesterday we created a LBalancer. Now, I want to create a LB in Azure but I don't want public loadbal. → i.e. I don't want it to create a public IP add., I want it to create a private-IP which works only within the network of Azure. So, how do I tell that information?

- so k8s manifest file doesn't have any such info.

So, Generally, those tools use something called as Annotations.

→ Annotations are key-value pairs which are used by External tools.

For Eg: Anything specific to Azure — if you want to write in your manifest → you can't use any other sections → the only section which is left to you is → Annotations.

Google: annotations in k8s.

- Annotations are also like metadata. But they will be used by tools.

So, the way you write annotations is not very much different from the way you write Labels. But, the usecase is different.

Google: k8s @ Anns. | change-cause Annotation.

whenever you want to basically deploy → what you can try to write is → You can try to write an Ann° which is called as change-cause. ✓

→ Anns have command line also. (just like kubectl label), but it is not a good idea to use commands. Try to write them in Deployment itself.

aspects

Google: k8s service annotations azure (Microsoft)

↓

use an Internal Load balancer with AKS

So, creating an Internal lb in Azure.

Service → type: LoadBalancer.

↳ metadata

annotations:

service.beta.kubernetes.io/azure-load-balancer-internal: "true"

we are telling Azure to do this so CCM will take control & do it. K8s has not control over it

So, the moment you add this ann? → what azure will do is → OK, he is asking for a lb but it shud not be public.

- So, for every cloud provider → You wud have some specific annotations, Especially whenever we are dealing with ingress and some cloud-specific things. In those cases, we wud be writing ann^ns.

→ Go to nginx-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  annotations:
    kubernetes.io/change-cause: "update to phpmyadmin"
```

(image: phpmyadmin)

> kubectl apply -f .

> kubectl rollout history deployment nginx-deploy

o/p:

Revision	CHANGE-CAUSE
3	none
4	"
5	"
6	"
7	update to phpmyadmin.

So, whenever you change a deployment, change that annotation and write reason why you are doing that. This is the better way.

The Usecase of Annotations is → to add any external functionality by third-party tools. This is not something K8S will use internally

Google - k8s Azure Annotations

Google: AWS k8s Annotation Ingress. (Github)

AWS k8s Service Annotations (service related aspects)

Google: k8s service annotations azure (Microsoft)

↓

use an Internal Load balancer with AKS

So, creating an Internal lb in Azure.

Service → type: LoadBalancer.

↳ metadata

annotations:

service.beta.kubernetes.io/azure-load-balancer-internal: "true"

we are telling Azure to do this so CCM will take control & do it. K8s has not control over it

So, the moment you add this ann? → what azure will do is → ok, he is asking for a lb but it shud not be public.

- So, for every cloud provider → You wud have some specific annotations, Especially whenever we are dealing with ingress and some cloud-specific things. In those cases, we wud be writing anns.

→ Go to nginx-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  annotations:
    kubernetes.io/change-cause: "update to phpmyadmin"
```

(image: phpmyadmin)

> kubectl apply -f .

> kubectl rollout history deployment nginx-deploy

o/p:

Revision	CHANGE-CAUSE
3	none
4	"
5	"
6	"
7	update to phpmyadmin.

So, whenever you change a deployment, change that annotation and write reason why you are doing that. This is the better way.

The Usecase of Annotations is → to add any external functionality by third-party tools. This is not something K8s will use internally

Google: k8s Azure Annotations

Google: AWS k8s Annotation Repos. (Github)

AWS k8s Service Annotations (Service related specific)

DaemonSet :-

* DaemonSet is a Controller which creates pod on every/selected nodes in k8s cluster.

For Eg: If you want to run one pod for every node in your cluster.

Use Cases :- This is not an application. This is additional functionality.

- log collectors
- agents etc.

For Eg:- In k8s nodes, you have certain data → you want to export that data all the time from every node in your k8s cluster → so, how do you do that ?

You basically create a DaemonSet.

DaemonSet ensures that whatever Pod you have configured in Daemonset runs on every node in k8s cluster.

Basic use case is → wherever you want to run agent-kind of softwares. (Agents are the softwares that get installed on every node and then they communicate with Server)

→ Let us assume that I want to install some agent → log agent → which collects logs and exports to Server.

Any kind of agenting kind of functionality → we will be using a daemonset.

Google: k8s daemonset.

→ This is again a Controller which controls pods.

When you delete the node → pod will automatically be deleted. It will not try to reassign-bcoz it is speaking about one pod per node.

Some typical uses of a Daemonset are :-

- running a cluster storage daemon on every node.
- running a logs collection daemon on every node.
- running a node monitoring daemon on every node.

>mkdir daemonset

↓

fluentd-ds.yaml :-

$kubectl api-resources | grep daemonset

↓

ds → is short form.

fluentd is one app^n which is log collector. It can read logs and it can send to whatever server you have configured.

So we are configuring fluentd.

You can configure different kinds of logs.

Google : api reference → Daemonset v1 apps

apiVersion: apps/v1.

kind : DaemonSet

name : fluentd-ds

annotations :- kubernetes.io/change-cause : "added fluentd debian".

spec

↓

Daemonset spec → minReadySeconds: 5

Here also, you can do deployments, rollout, undorollout - - -

revisionHistoryLimit → how many old history do you want to retain. to allow rollback

default = 10. (previous 10 things)

Selector → LabelSelector:

matchLabels

app : fluentd.

template → PodTemplateSpec

metadata:

name : fluentd

Labels:

app: fluentd

spec :

containers:

- name : fluentd

image : fluentd:v1.16-debian-1

Lets try to run fluentd container

Playground.	docker hub : fluentd.

$ docker container run -d -P fluentd

$ docker container ls

✓ ✓

fluentd-ds.yaml :-

```
---
apiVersion: apps/v1
Kind: DaemonSet
metadata:
  name: fluentd-ds
  annotations:
    Kubernetes.io/change-cause: "update to latest"
spec:
  minReadySeconds: 5
  selector:
    matchLabels:
      app: fluentd
  template:
    metadata:
      name: fluentd
      labels:
        app: fluentd
    spec:
      containers:
        - name: fluentd
          image: fluentd:latest.
```

2nd.

1st case

"added fluentd debian"

fluentd:v1.16-debain-1

> Kubectl apply -f .

O/p: daemonset.apps/fluentd-ds created

> Kubectl get ds

O/p

Name	Desired	Current	Ready	Available	Node selector
fluentd-ds	2	2	0	0	<none>

I have 2 nodes and it is running on both the nodes. I have not specified any count. The only thing which I specified is — I want a Pod called fluentd.

> kubectl get nodes

I have 2 nodes. It is running on both.

→ If I basically give my count → to 3 nodes → then it will automatically create 3 pods. That is how daemon sets work.

> kubectl rollout history ds/ fluentd-ds

Revision	Change-cause
1	added fluentd debian

> kubectl apply -f .

> kubectl rollout history ds fluentd-ds

o/p

Revision	Change-cause
1	added fluentd debian
2	update to latest

So
Let it be daemon sets, let it be deployments, let it be stateful sets

Every time add that annotations.

→ For 3 controllers → we can see rollout status — 1) Deployments
2) Daemonsets &
3) Stateful sets.

$ kubectl get po

o/p: fluentd-ds-kk. 1/1 Running
fluentd-ds-tpc. 1/1 Running.

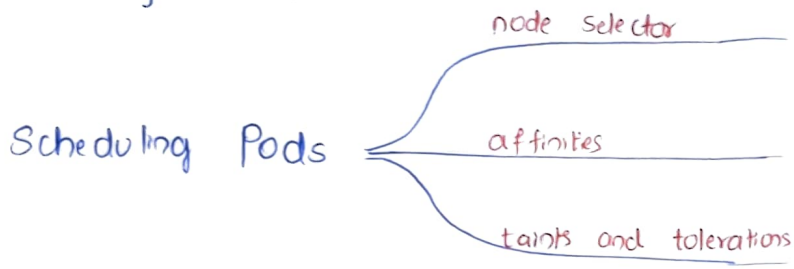
> kubectl delete -f .

x I want to schedule pod on some nodes.

So what are the options which K8s will try to give me.

• Scheduling Pods :-

There are 3 major ways :-



→ One is called as node selectors.

In node selectors, what you wud try to tell is,

in your podspec you will directly give node name (In which node you want to run the Pod)

→ Affinities and tains + tolerations are a bit tricky. Node selector is easy method.

Node Selectors :-

> mkdir scheduling

> cd scheduling

> mkdir nodeselector.

↓

jenkins.yaml

> kubectl get nodes

O/p: aks-nodepool1 - - - 000 I have 2 nodes.

aks- .. - - - 001

I want to run it on node-000 (whatever Pod which I'm trying to write).

→ Google: node selector k8s (Assigning Pods to nodes)

Docker hub: jenkins. ⟶ jenkins/jenkins : jdk11

- jenkins.yaml

```
---
apiVersion: v1
Kind: Pod
metadata:
  name: nodeselector
  labels:
    app: nginx
    purpose: nodeselector
spec:
  containers:
    - name: jenkins
      imagp: jenkins/jenkins:jdk11
      ports:
        - containerPort: 8080
```

- expose.yaml

```
---
apiVersion: v1
Kind: Service
metadata:
  name: expose-svc
Spec:
  selector:
    purpose: nodeselector
  type: LoadBalancer
  ports:
    - name: nginx-svc
      port: 80          ---> (If I give 8080 in port)
      targetPort: 8080      ↓ then I have to give loadbalancer's IP:8080
      protocol: TCP              while accessing jenkins.
```

> kubectl apply -f .\nodeselector (folder's name) (whatever is present in the folder will be created)

$ kubectl get po -o wide

O/P:	Name	Status	.....	Node	Nominated Node	Readiness Gates
	nodeselector	Running		aks-node...001	<none>.	<none>.

so

where did the pod get created ? → on nodepool-001.

* To be very specific → I didn't work any logic (ie. I didn't give which node I want my pod on)

It might have been created anywhere.

* So If I create one more pod → it might get created on node-000 (or) node-001.

Right now, I'm not controlling → on which M/C (node) my (pod) app^n is running

If I want to access my app^n :-

$ kubectl get svc

o/p. exposc-svc LoadBalancer -- 20.75.165.138 80:30890/TCP

Kubernetes clusterIP --

Chrome :→ 20.75.165.138 → open.

* Now, I'm interested in which node my app^n runs on :-

Go to API ref → Pod v1 Core.

↓

Spec → nodeSelector , nodeName.

↓

object.

* Now lets select node by its name :-

↓

But it is not a good idea.

> kubectl get nodes

o/p: Name

aks-nodepool1-...000 → I want my app^n on this node.

aks-nodepool1-...001

First → Edit jenkins.yaml

↓

nodeName : aks-nodepool-...000.

(Is this a wonderful way ?)

↓

NO.

will it work ? → Yes.

Spec :

nodeName : "

containers :

- name : jenkins

> kubectl apply -f .

- Since the pod was already created
- it is not a controller object
- You can't do changes.

$ kubectl get po -o wide

Op:-

Name	Nominated Node	Node
nodeselector	1/1	aks-- ooo ✓

Now, our pod is running on whatever node we specified in yaml file.

Using nodeselector (or) any others → to assign pods to nodes is not good idea.

* Rather than using this,

let us assume that, I have categories of nodes.

So, how can we select specific pods in K8s → we will go with Labels.

So, if you apply the same logic → so, while I'm basically creating nodes, for the nodes I will either use existing labels (or) I'll create labels and rely on those labels → bcoz →

let us assume that – I have a label called as hardware type : GPU →

So what I'm trying to tell in node selectors is → I wud want to run my pod on any of the nodes which have this label.

This is a better way rather than giving a node name (or) node's ip-address bcoz we will never know which node is available or which node is not available. So relying on labels might be a good idea.

* Lets look at nodeselectors :–

→ NodeSelector is a selector which must be true for the pod to fit on a node.

Selector which must match a node's label for the pod to be scheduled on that node.

→ we have two nodes. Lets attach the following labels:–

- Purpose: poc — node 0
- purpose: testing — node 1.

For poc purposes, If I'm developing a pod then I want to use node 0.

For testing whatever I have developed, I want to use node 1.

$ kubectl label nodes aks-nodepool1-328... -mss 0000 purpose=poc

o/p: Error.

There is no other way. I have to add another nodepool.

Our nodepools name is nodepool1.

Ideally you can apply labels directly
But, since this is a managed cluster it is not allowing you to do that.

* Go to my Aks cluster → nodepool1 →

nodepools → Scale nodepool

↓

change node count from 2 to 1.

I'm just making count back to 1.

az aks nodepool update \
--resource-group myResourceGroup \
--cluster-name myAksCluster \
--name labelnp \
--labels dept=ACCT purpose=poc \
--no-wait

↓

see for labels in Azure.

* I'll try to add one more nodepool: nodepool2

$ az aks nodepool add \
--resource-group myResourceGroup \
--cluster-name myAksCluster \
--name nodepool2
--node cnt 1 \
--labels dept=HR purpose=testing
--no-wait.

> kubectl get nodes

o/p: aks-nodepool1-... 000 Ready
aks-nodepool11-... 001 Ready/scheduling Disabled

↓

is disabled. (later discussion)

Eventually we can see other nodepool as coming up.

Now, go to Jenkins.yaml:

replace nodename with nodeSelector.

spec
nodeSelector:
purpose: poc ← label which I wants to match is

Any node which has this label → ... will run this Pod

$ kubectl get nodes

O/p: aks-nodepool-...-000 Ready ---

Go to Azure → myAKSCluster → we can see nodepool1 – succeeded
nodepool2 – Creating.

what we want is not this ←

> kubectl apply -f .\nodeselector
> kubectl delete -f .\nodeselector

> kubectl apply -f .\expose.yaml.

> kubectl apply -f .\jenkins.yaml.

Bcoz once the pod is created you can't change the scheduling. You can do all of that in controllers but not directly pods

> kubectl get po -o wide

It is going to nodepool1 bcoz of label matching.

Let us see if second nodepool is up.

$ kubectl get nodes

O/p: aks-nodepool1-...-000 ✓
aks-nodepool2-...-000 ✓

> kubectl delete .\jenkins.yaml.

Now Edit jenkins.yaml → purpose: testing. → we have given this label in nodepool2.

> kubectl apply -f .\jenkins.yaml

> kubectl get po -o wide

O/p: nodeselector - - - aks-nodepool2. It went to nodepool2.

kubernetes.io/os: linux | kubernetes.io/os: windows

Now, you wud write a node selector where you'll tell that – I'm interested in running this pod with . Then your pod will end up on windows servers if they exist.

> kubectl delete -f .\jenkins.yaml

This time lets give a purpose value which is not existing.

jenkins.yaml → purpose : development. (purpose : poc – nodepool1
purpose : testing – nodepool2)

↓ we have only 2 labels.

> kubectl apply -f .\jenkins.yaml.

> kubectl get po -o wide

		Ready	status	Node	Nominated Node
O/p:	node selector	0/1	Pending	<none>	<none>

Your K8s Pod is in pending state. Bcoz Your Scheduler is unable to find any node which has a label called as purpose : development.

So Kube-Scheduler is unable to decide which node your pod shud be send to.

When label doesn't match → your pod will always be in Pending state.

→ When we have tried to create a Pod with nodeSelector matching Purpose : poc it was created on nodepool1 and

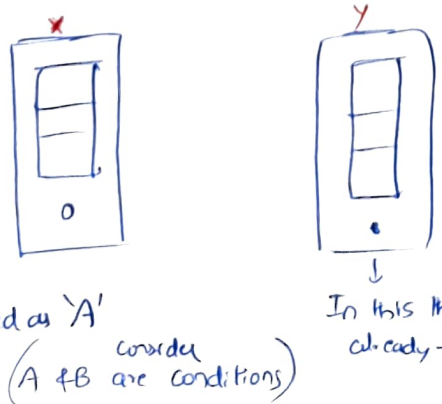
when we created a pod with Purpose: testing it created in nodepool2 and

when created a pod with Purpose : development it was created in Pending state (not created)

* when I want to run a pod → I want to run a pod in a node which has these conditions → that is node affinity.

I want to run a pod in any node where there are pods with these settings → that is Pod affinity.

Let's assume I have 2 zones. Now I have to schedule a Pod.



(A & B are conditions considered)

↓
I can try to tell that
↓
Now create a Pod based on condition X. ( ) then it will run in 1st one.

X & Y are node related conditions

A & B are Pod related conditions

- So, whenever you are relying on node-based conditions → that is called as node affinity
- Now, if I try to tell that → I want to run this pod where there is no 'B' running then it will go & run on 1st one.

I want to run this pod where 'B' is running → then it will go & run on 2nd one.

This is called as Pod affinity.

→ Here we are trying to help Kube-Scheduler to find the right node to run your workload.

In an Ideal case → You shud not worry about it.

The reason why we basically do these kinds of things is w.r.t diff.t hardwares

So, I might have nodepool1 where there is 4GB Ram node.

I might " nodepool2 " " " 16GB " " .

This pod for some reason has to run on 16GB node. So then how do I schedule it?

This pod has to run on node where there are 8 CPUs → " " "

* I don't want to run 2 mysql pods on the same node — for some reason,

↓

Then I'll try to go with pod affinities.

→ There is something called as affinity and anti-affinity.

- Affinity/Anti Affinity Based :-

* This scheduling happens o.b.o affinity.

Affinity means when it matches schedule it here.

Anti-Affinity means when it matches → don't run it here.

Google: node affinity in K8S → Assign pods to nodes using Node Affinity.

Here you will have 2 interesting sections

In Pod → nodeaffinity:

requiredDuringSchedulingIgnoredDuringExecution

- This basically means that ← (when you are using required section)

whereva your pod is getting scheduled – this shud match → when it matches only – your pod will be selected

But when you are using preferred section :-

nodeAffinity:

PreferredDuringSchedulingIgnoredDuringExecution

It will try to see if it matches → (Even) If it doesn't match also → It will schedule.

(*) Preferred is not a mandatory rule → It is 'good to have' kind of rule.

required means → it shud mandatorily follow.

In earlier case → when labels were not matching → your Pod was in Pending state.

In this case, what wud happen is

So, if you use it in preferred section → Even if it is not matching → it wud apply somewhere → It will create a Pod.

API-ref

→ Podspec

↓

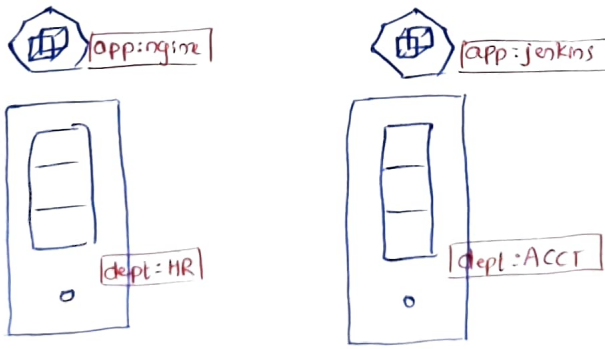
affinity (It is basically matching constraints)

Type (Affinity) → nodeAffinity → Speaks about labels of the nodes (2 cases: mandatory rule / good to have rule.)

PodAffinity → Speaks about labels of other Pods which are running on that node.

PodAntiAffinity → means I want to run this pod on a node which doesn't have Pod with this label.

So, there are 3 kinds of rules over here.



Now, whenever I'm trying to write an affinity rule.

↓

I'm trying to write an affinity rule which is based on node affinity

↓

I'll try to tell → dept:HR (But I have written this in requiredDuring....)

So then what wud happen is

↓

It will try to see that, there is a label called dept:HR in 1st node → let me try to run it over here.

* Now, I have written this label not in 'required' but in 'preferred'

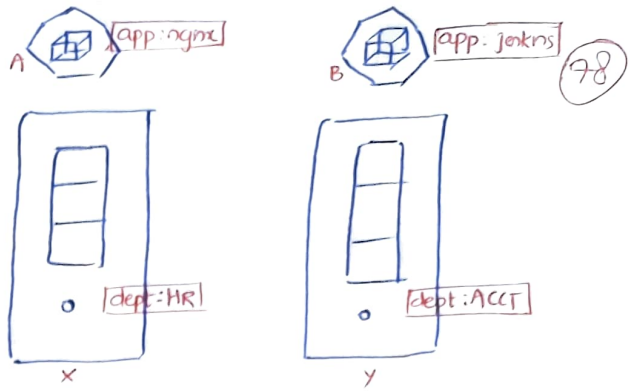
↓ Eg: dept: HT (not necessary to match)

So, there is no node that is matching it

↓

But this might be executed on any one of the node.

→ This is how node-affinities work.



* Now, lets speak about Pod-Affinity :-

I'm trying to apply pod affinity which is trying to tell that → app: jenkins

so, now, what I'm trying to tell is → I want to run this pod on a node which has any pod → which has a label → app: jenkins → Now it will go & schedule on 2nd Server(node).

* Now, if I have written this in Anti-Affinity section →

Since there is already a Pod with label → app: jenkins → It will not schedule over here (in 2nd node)

→ It will schedule it over 1st node.

* Taints and Tolerations :-

Google: Taints and Tolerations in K8S.

Node affinity is basically the property where you are trying to tell that → run this pod on a node where there are matching labels.

But here, it is exactly Opposite.

Taints, what they try to do is → they allow node to → not allow pod to run over here.

→ Taints are the opposite — they allow a node to repel a set of pods.

- So, basically what you wud do is → You wud apply taint to the node.

So Taint is some condition.

Now, there are tolerations.

Tolerations is something which we apply to the pod.

* Now K8s will apply pod to a node with matching tolerations.

→ In all the cases, what is it that we are trying to do →

I have a pod → I want it assigned to the node → Here, based on the Podspec — we are making a decision.

But, ova here, the case is reverse → Here, node has a taint and based on the taint → pods will be scheduled.

Eg: Tainting a node → kubectl taint nodes node1 key1=value1:NoSchedule.

↓

places a taint on node → (node 1).

The taint has Key → key1 and Value → value1 and taint effect → NoSchedule. This means that no pod will be able to schedule onto node1 unless it has a matching toleration.

Eg:

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
```

It is written ↓

that → when key1 = value1 → do not schedule.

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

↓

If key1 exists do not schedule.

Taints & Tolerations → because these things are req'd when you have nodes with specialized hardware

The default value for operator is Equal

* Here, the priority is given to the node and whatever taint is applied to the node → there shud be a toleration against it.

Pod-Affinity, node selector → these are all priorities given to the pod and node has no decision to make.

But here → a decision-maker is actually a node.

(So, in the node, we will be trying to apply a taint saying that → apply pods only when they have matching tolerations, otherwise do not schedule)

- K8S is a cluster.

So let us assume that you have created a AKS cluster. And now, your Orgzn has 5 or 6 Environments. Dev, QA1, QA2, UAT, staging, prod? Envts.

So, do I need to create 6 K8S clusters for that. The answer cud be Yes (or) no.

But the point is, K8S tries to tell that → I have a virtual cluster in me. You can create a virtual cluster, not a physical cluster. Virtual cluster → which is called as namespace. ↓ is K8S cluster.

All these we are using namespace and our namespace is default.

* Yesterday, we have created a nginx-replicaset (nginx-rs) and nginx-service (nginx-svc).

↓

- Headless Service in K8S :—

Google → K8S headless Service → Service

Generally whenever we create a Service → Every Service gets an IP-address.

mkdir headless-svc

nginx-svc.yaml :-

```
---
apiService: v1
Kind: Service
metadata:
  name: nginx-sv
spec:
  selector:
    app: nginx
  type: ClusterIP
  ClusterIP: None
  ports:
  - name: nginx-svc
    port: 80
    targetPort: 80
    Protocol: TCP.
```

I'm speaking about type: clusterIP.

when we create clusterIP → we get IP address

ClusterIP: none

↓

what this means is → I will not get an IP-address to the Service.

Any Service which does not have its own IP-address is called as a Headless Service.

nginx-rs.yaml :-

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
  labels:
    app: nginx
    layer: web
    version: "123"
spec:
  minReadySeconds: 1
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
        version: "1.23"
    spec:
      containers:
        - name: nginx
          image: nginx:1.23
          ports:
            - containerPort: 80
```

alpine-pod.yaml :-

```
---
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
    - name: alpine
      image: alpine
      command:
        - sleep
        - 1d
```

> kubectl apply -f .

o/p: pod/alpine created

replicaset.apps/nginx-rs created

service/nginx-svc created.

> kubectl get po

o/p: 1 alpine

5 replicas.

> kubectl get svc

			External IP	
o/p:	kubernetes	clusterIP	<none>	443/TCP
	nginx-svc	clusterIP	<none>	80/TCP

$ kubectl exec -it alpine -- /bin/sh

/# nslookup nginx-svc

o/p: I got 5 IP-addresses of k8s replica sets.

When you query a Headless Service → it will not give its IP-addr. It will give IP-addresses of all the pods which (it is connected to) (or) (it exposes)

This is useful when we try to learn something called as Stateful sets.

But if write cluster-IP → I'll not be getting IP-addresses of Pods. I'll just get Service's IP-address.

Lec 57 - Apr 30 - Part-2 | Storage Solutions in K8S, Stateful Sets

* W.r.t storage, first of all, let us quickly reiterate the classical problem with Docker Containers.

The problem with Dr Cs is → Every Dr Cr whenever it writes any content into the local system → it wud be present in Read-write Layer and when you delete the Cr → you will loose all that data.

→ To resolve this, in the case of Dr, we have used Volumes.

The basic idea is, a volume which we have created (a) a which helps in retaining data → how can I use it in K8S.

what are the different options that are basically present?

- Stateful Applications store data locally. In Containers the data created locally will be lost once you delete it. So to solve this in docker, we have used volumes.

Volumes have a lifecycle which has no relation to Container Lifecycle (refer dr Cs, image layers, volumes)

→ For Eg: If I have a Web appn and whenever I try to access the web appn & if it is storing all the data in database → then this web appn will become stateless and that database becomes stateful. ('stateful appn use local storage to store some data whereas stateless appns do not do that)

* In K8S, we are running docker containers, K8S is an orchestration solution.

Lets see what are options for storage Provisioning in K8S

Google: K8S storage.

→ The most widely used storage types :-

- Volumes
- Persistent Volumes
  - → Storage classes
  - → Persistent Volume Claims.

* All the other storage options you see in k8s are basically an extension to the ones mentioned before.

- **Volumes :-**

A Pod can have any number of volumes. But this volume will be available as long as Pod is running. Once you delete the Pod → volume will also be lost.

But in the Pod, no matter how many times your ©ʳ will restart → it will still have the same volume.

The scope of this volume is limited to — the lifetime of Pod. Once you delete the Pod you will loose the volume.

still, this volume will have lifetime other than container. No matter how many times ©ʳ restarts → volume is preserved as long as Pod is alive.

- Volumes can be mounted to containers and they have lifetime equivalent to Pods.

→ It is not a very useful scenario for us → bcoz, in k8s we generally don't create ©ʳˢ directly → we create pods → so, when you are deleting the pods, you are loosing the volume.

→ Types of volumes :

* storage on cloud
  - ebs
  - azure disk
  - efs
  - azure file
  - gcs
* Empty dir → means when you create a volume → it will give you an empty directory.
* host Path → means → your Pod runs on some node. On the node there is some folder or file which will be mounted.

* Lets create a manifest with mysql pod with volume

* Manifests → mkdir storage → mkdir volumes → mysql-vol.yaml

mysql-vol.yaml :-

```
---
apiVersion: v1
kind: Pod
metadata:
  name: mysql-vol
  labels:
    app: mysql
    layer: db
spec:
  containers:
  - name: mysql
    image: mysql:8
    ports:
    - containerPort: 3306
    volumeMounts:
    - name: test-volume
      mountPath: /var/lib/mysql
  volumes:
  - name: test-volume
    emptyDir:
      sizeLimit: 100Mi
```

docker hub
↓
mysql
↓
mysql:8

In (C), whenever I want to try to use a volume, I will use → volumeMount
↓
array.
↓
will have mountPath & name
↓
must match with name given in Volumes
↓
that is how the link gets created

volume ↴
This is not useful.

we have something that is useful → Persistent volume.

Persistent Volumes :-

→ These volumes will have a lifetime different than a Pod → i.e. they exist even after the Pod is dead.

→ Types of PVC (Google: K8s Persistent Volumes)
→ Storage classes in Azure (Google: Aks Storageclasses)
→ Storage classes in AWS (Google: Aws storage classes)

There are 2 kinds of Persistent volumes :-

(1) Static Provisioning → Your k8s administor needs to create the storage.

(2) Dynamic Provisioning

↳ when you go with Dynamic volume, k8s will automatically create the storage dynamically by taking the help of CCM.

In Azure → it will create Azure storage

In AWS → " " AWS "

- So, we are speaking about Persistent volumes which not only have lifetime other than your pods, they also can create storage for you.
- For creating the storage, what they use is, they use something called as Storage classes.

In many of the managed k8s clusters → storage classes already Exist.

So, you can take the help of them to create this dynamic storage (or) you can write something on their own.

- So, persistent volume is a disk.

Storage class is something which can help you in creating disk.

Ex: If I want 1 GB of storage → so creating that 1 GB of storage is something which storage class can do.

- In your pod what we can do is → we make a Persistent Volume claim (PVC)

So, whenever we are trying to write the pod → we'll be trying to create a PVC.

- PVC :-

* A PVC is a request for storage by a user. It is similar to a Pod.

Pods consume node resources and PVCs consume PV resources.

* Pods can request specific levels of resources (CPU and Memory).

Claims can request specific size and access modes (Eg. they can be mounted ReadWriteOnce, ReadOnlyMany (or) ReadWriteMany).

- Persistent volume is a actual storage which can be dynamically created with the help of storage classes,

PVC is a link b/w your pod and the volume that is actually created.

* We have a kind → PersistentVolumeClaim.

kind → PersistentVolume

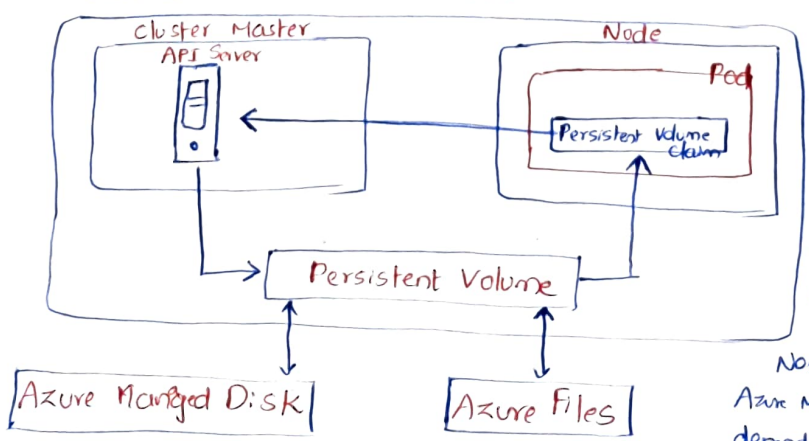
In your pod, whenever you want to use the mount, you will be using this claim.

→ Types of PVC :-

* azure disk
* azure file
* aws EBS
* gce Persistent Disk etc.
* csi → is a container storage Interface → which can be implemented by any plugin → then they can create a storage for you.

For Eg: If your company is using NetApp storage → there are companies who sell NetApp's csi-plug-in. Now you need to add it to the k8s cluster → Once you do that → Your manifest will be able to create NetApp's storage.

*Google : Azure AKS Persistent volume.



Whenever you are creating a node,
↓
In the node, you have a pod
↓
In the pod, you create PVC.
↓
This PVC will basically get attached/linked to your Pod with the help of Per. volume.
↓
Now whether you want to create Azure Managed disk (or) Azure file depends on storage class

* You can use different kinds of storages.

→ Volumes defined and created as part of the Pod lifecycle only exist until you delete the Pod.

Pod often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in stateful sets.

→ A Persistent Volume (PV) is a storage resource created and managed by the K8S API that can exist beyond the lifetime of an individual Pod.

→ You can use Azure Disk (or) Azure files to provide the P.V.

- To define different tiers of storage (such as premium & standard), you can create a Storageclass.

The StorageClass also defines the reclaimPolicy.

When you delete the P.V, the reclaimPolicy controls the behaviour of the underlying Azure storage resource. The underlying storage resource can either be deleted (or) kept for use with a future Pod.

- storage classes :- use managed-csi → to get standard SSD (you get standard SSD disk)

managed-csi premium → creates a Azure Premiumdisk.

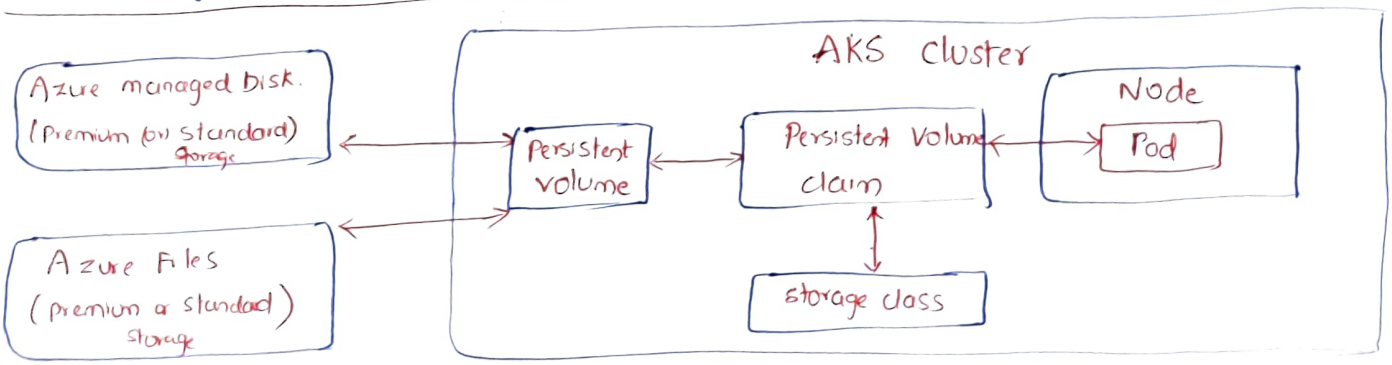
Azurefile-csi-premium → Creates a Azure file share.

* If you want, you can create a storage class for additional needs using Kubectl.

↓ kind : StorageClass.

→ A PVC requests storage of a particular storageclass, access mode and size.

The K8S API Server can dynamically provision the underlying Azure storage resource if no existing resource can fulfill the claim based on the defined Storageclass



- Once an available storage resource has been assigned to the Pod requesting storage, Persistent Volume is bound to a Persistent Volume claim. Persistent volumes are 1:1 mapped to claims.

→ Pvc takes the help of Storage class to create a P.V.

In case of Azure, storage class can be one of these 2 ⟨ Azure Managed Disk / Azure Files

PV will be created and attached to your pod.

Now, CCM will create a 1.5GB of disk depending on which storage class you have asked for.

Now, when writing a pod (remember, we used host = emptydir) → here you will be using something called as

```
- name : volume
  persistentVolumeClaim :
      claimName : azure-managed-disk.
```

and we can mount it into whereever we want

```
volumeMounts :
- mountPath : "/mnt/azure"
  name : volume.
```

Storage classes in AWS :-

- we have a storage class called gp2 (general purpose 2)

↓

It is a disk that gets created when you create an EC2 Instance.

→ cd storage → mkdir Persistentvolumes & cd :-

1) mysql-pvc.yaml
2) mysql-pod.yaml

$ Kubectl api-resources | grep storageclass

O/p :- storage classes — false — SC ↓ short name.

$ kubectl api-resources | grep persistent

O/p :- persistentvolumeclaims — true — Pvc — v1

persistentvolumes — false — Pv — v1

$ kubectl get pv

O/p : No resources.

$ kubectl get pvc

O/p : No

$ kubectl get sc.

O/p :- — — — — — — Then are storage classes which are available.

For me to create a claim, all I need is storageclass

* Right now, the provisioning which we are using is Dynamic Provisioning → why?

I'm not creating storage. I'm directly writing PVC → there i'm making a request saying that I need 1 GB disk and then it is getting created.

* Lets create a my-sql Pod with dynamic volume Provisioning :-

- Get storage classes. I got it with kubectl get sc

I can use any of the storage classes.

Google: az aks dynamic volume provisioning.

Lets Create a Persistent volume claim :- PVC → is making a request to create volume dynamically.

API-ref → (version 1.25)

↓

PersistentVolumeClaim v1 Core.

version = v1

kind = PersistentVolumeClaim.

metadata

↓

objectmeta → name = mysql-pvc → This name is important → as we will be using this name in actual ☺.

spec

↓

pe..v.claimSpec.

→ accessModes

In K8s, whenever we are trying to write PVs there are diff accessmodes.

I'll be using ReadWriteOnce

↓

bcoz I cannot have 2 mysql dbs trying to write at the same time bcoz — it will mess.

(NO disk will handle concurrency for you)

ReadWriteOnce → this volume can be mounted as read-write by a single node.

ReadOnlyMany

ReadWriteMany

This access still can allows multiple pods to access the volume when the pods are running on the same name.

the volume can be mounted as read-only by many nodes.

the volume can be mounted as read-write by many nodes

For Eg: If you put your Excel file in shared disk and then if you open, If someone else tries to open → it will try to tell that you cannot write anything to this file bcoz there's some lock that has been created. That lock is not created by the disk It is created by the Excel app?.

But if you open notepad app? → it will not complain → whether 2 people open (or) 200 people open from the shared disk bcoz → disk cannot do anything. It is done by that app?. For that we are writing access mode.

- ReadWriteOnce → we are trying to tell → now whenever you are mounting this → mount this to the node and ensure that any pod who are basically running on the same node can mount it as a volume.

RWOP → ReadWriteOncePod is also available.
↓
means give only to one pod & not to node.

→ resources → requests:
→ storageClassName = ?
default (or)
managed

storage: 1Gi

For static Provisioning
↓
Google it.

* mysql-prc.yaml :-

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-prc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: managed
```

* mysql-pod.yaml :-

```
---
apiVersion: v1
kind: Pod
metadata:
  name: mysql-dynamic-vol
  labels:
    app: mysql
    layer: db
spec:
  containers:
    - name: mysql
      image: mysql:8
      ports:
        - containerPort: 3306
      volumeMounts:
        - name: mysql-pvol
          mountPath: /var/lib/mysql
  volumes:
    - name: mysql-pvol
      persistentVolumeClaim:
        claimName: mysql-pvc
```

} will automatically create a disk for me.

> kubectl apply -f .\mysql-pvc.yaml

> kubectl get pvc

O/p: mysql-pvc (status) Pending.

> kubectl get pv

O/p: No resources

> kubectl apply -f .\mysql-pod.yaml

O/p: pod/mysql-dynamic-vol created.

> kubectl get pvc

o/p:-

Name	Status	Volume	Capacity	Access modes	Storage class
mysql-pvc	Bound	pvc-1a.... cebc	1Gi	RWO	managed

Is it bound now? = YES.

Until and unless you are using that – it will not create a disk.

> kubectl get pv

It created 1 Gb disk now.

o/p:-

Name	Claim	Capacity	Access Modes	Reclaim Policy	Status	Storageclass
pvc-1a...cebc	default/mysql-PVC	1Gi	RWO	Delete	bound	managed

1st I have created a PVC. → In default namespace, I have mysql-PVC.

↓

After that, I asked for pvc → it said – it is in Pending state

↓

Then, I have applied my-sql-pod

↓

Once I applied mysql-pod → if i asked – it said it is bound.

* What we have done is not very Effective one. We'll be changing that.

We just created some storage space. Lets try to put it to action.

> kubectl delete -f .

> kubectl get pvc

o/p: No resources

> kubectl get pv

o/p:				Status	Storage
	pvc-1a...cebc	1Gi	RWO	released.	managed.

I still have my disk. I can still use it.

* Now, I want to create mysql pod. while creating it, it have to set some Environmental variables → to try to tell what is mysql user, mysql root password etc.

* Now, lets modify the spec → to add mysql related Environment variables.

Exercise:

Try to create any Pod spec by setting Environment variable. (Eg: Alpine Pod with 1 env var → 'PURPOSE=TEST' with cmd → sleep, id.)

env is an array of EnvVar. → name, value.

* mysql-pod-yaml :-

```
---
apiVersion: v1
Kind: Pod
metadata:
   name: mysql-dynamic-vol
   labels:
     app: mysql
     layer: db
Spec:
  containers:
    - name: mysql
      image: mysql:8
      ports:
        - containerPort: 3306
      volumeMounts:
        - name: mysql-pvol
          mountPath: /var/lib/mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: rootroot
        - name: MYSQL_USER
          value: qtdevops
        - name: MYSQL_PASSWORD
          value: rootroot
        - name: MYSQL_DATABASE
          value: employees
volumes:
  - name: mysql-pvol
    persistentVolumeClaim:
      claimName: mysql-pvc
```

writing passwords in plain text is not a good idea (we will solve it later)

> Kubectl apply -f .

$ Kubectl exec -it mysql-dynamic-vol -- mysql -uroot -p

O/p: Enter Password: rootroot

I'm inside mysql

mysql> show databases;

O/p:

Database
employees
information_schema
mysql
performance_schema
sys

Google:
mysql sample create table Example
w3schools.

mysql > use employees;

O/p: Database changed.

mysql insert into.

```
mysql > CREATE TABLE Persons (
    -> PersonID int,
    -> LastName varchar(255),
    -> FirstName varchar(255),
    -> Address varchar(255),
    -> City varchar(255)
    -> );
mysql > INSERT into Persons (PersonID, LastName, FirstName, Address, City) VALUES (1,
        'one', 'one', 'one', 'one')
        (2,
        'one1', 'one1', 'one1', 'one1')
mysql > Select * from Persons
```

O/p-

PersonID	LastName	FirstName	Address	City
1	One	One	One	One
2	One1	One1	One1	One1

mysql > Exit

Now, delete Pod and recreate.

$ Kubectl delete Pod mysql-dynamic-vol.

> kubectl apply .\mysql-pod.yaml (wait for pod to be created & running).

> kubectl exec -it mysql-dynamic-vol -- mysql -uroot -p

O/P: Enter Password : rootroot

mysql > use employees;

Database changed.

mysql > Select * from Persons;

O/P :-

PersonID	LastName	FirstName	Address	City
1	One	One	One	One
9	One1	One1	One1	One1

The data is still available even after we deleted and recreated the Pod.

So, can I persist the data? → Yes.

→ Go to Azure → myResource Group → myAKS cluster → we can see PVC.
↓
disk that got created (1 GiB).

- Actually I have written a PVC
↓
that has spoken with StorageClass → and actually created a disk in Azure.

(Azure is something which has nothing to do with what K8s is) → (since there is a CCM → it is creating the stuff for you).

* Writing a Pod might not be a good idea. Bcoz Pod cannot maintain its state.

So, we need some controller.

One controller which we can use for this is → We can try to write replicaset and give one replica.

But there is some problem with that approach.

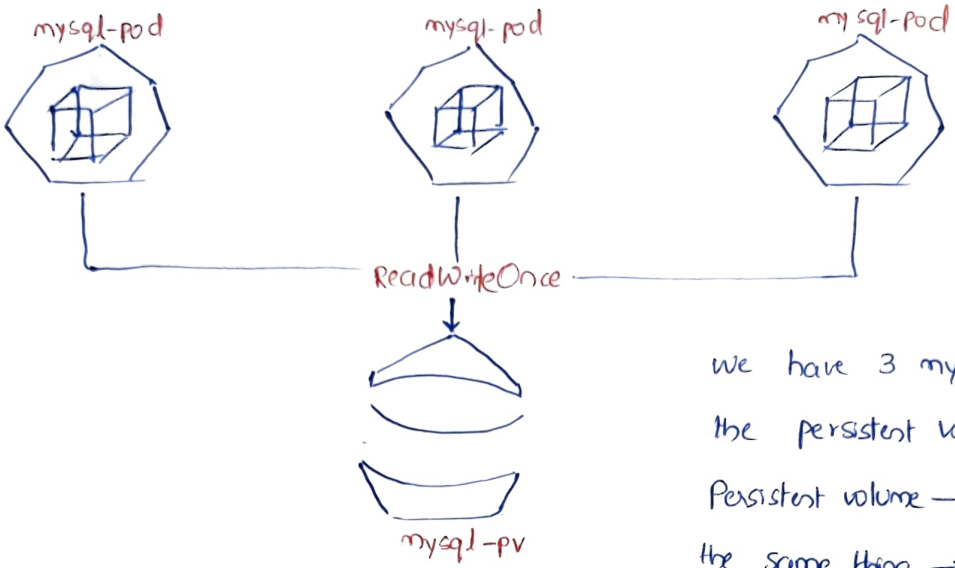
we have a better solution for this.

> kubectl delete -f .

-> If you want to scale pods we can use * deployment
* replicasets.

* You will have 3 Pods . You will have a mysql-pv and all these pods are using mysql-pv.

Our basic architecture :-



we have 3 mysql pods which use the persistent volume.

Persistent volume — If they are sharing the same thing → that means they have the same data. But there will be Concurrency problems over here.

So, it is better to create these kind of things with ReadWriteOnce.

↓

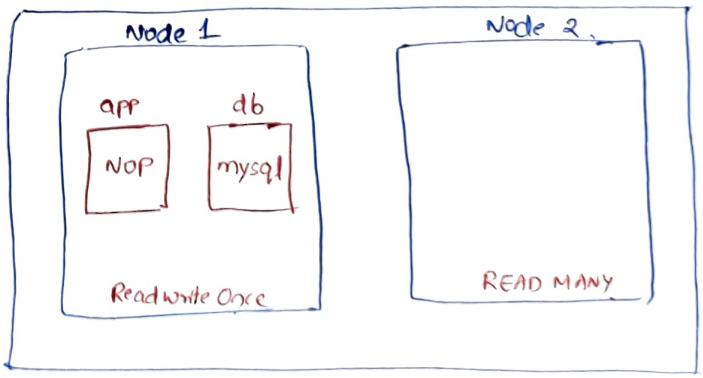
means give Read write Permission to only one node.

I can also give one more access = ReadMany → I don't have a problem with multiple Pods reading.

But write Permission — I want to give it only to one Pod.

Here you might have difficulties figuring out → which Pod has write permission & which pod has read Permission.

TO avoid this problem People don't give ReadMany.



• Stateful Sets :-

In stateful sets, basically what wud happen is you wud have a Pod.

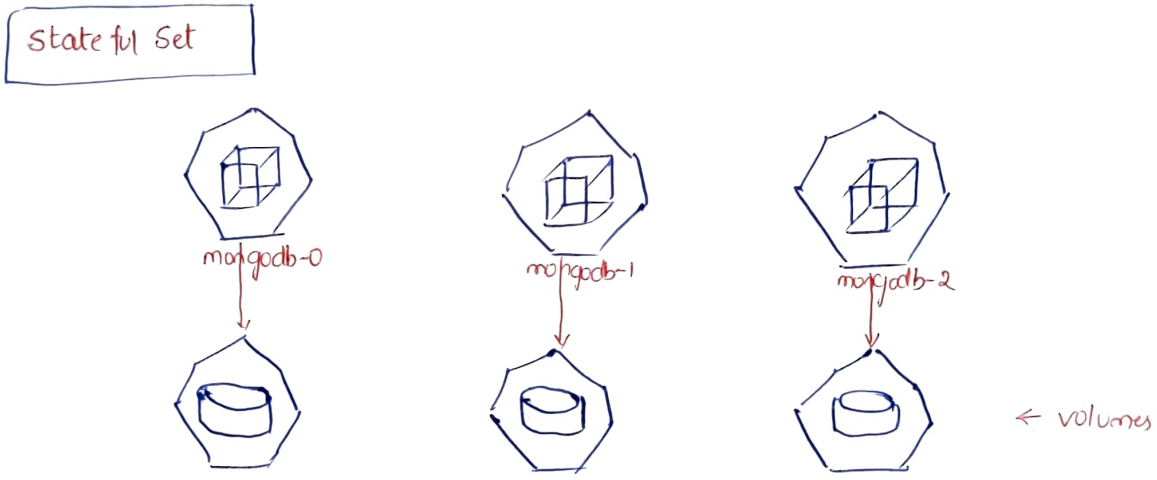
Generally, if you try to look at replica sets → In replica set, whenever it creates a replica — it gives a random name to the pods (Eg: mysql- some digits)

↳ replicas = 3 →
- mysql-0
- mysql-1
- mysql-2

I'll be getting procedural names (or) predictable names.

(Now, whenever, you try to create a P.V → you will be getting a volume for every Pod. → in Stateful sets)

- Stateful Set is like deployment with replicas. But each pod gets its own Volum
- Stateful Set is for Stateful applications.



Each pod gets its own Persistent volume. This is something which we call as a Database Cluster.

So, we have basically an app^n which is running in Pod which has its own state — which is basically maintained.

Now, for some reason, if you delete mongodb-0 pod, then → a new Pod will be created which exactly gets mount to the same volume.

Here each pod gets its own volume. They don't rely on 3 pods using same volume.

- If you remember, K8s stores everything in etcd cluster. All the modern dbs are designed to be distributed. This is a distributed system.

* Now, here what happens is → whenever you try to use the data, data gets stored in its own storage (Pod's):

Elastic Search :-

In Elastic Search, we have a concept called shards.

So generally what happens is → whenever you store the data in these kind of databases → what they do is →

they don't store the entire data at one place. They break that data into multiple pieces and they store in multiple things.

For Eg:-

lets assume, we have data - 'A'. This 'A' data gets broken into number of shards

↓ is a record

shards is how do you want to break this record.

So, I want to break it into 5 equal pieces.

A

↓

A1, A2, A3 A4 A5.

R1A1 R1A2 R1A3 R1A4 R1A5.

R2A1 R2A2 R2A3 R2A4 R2A5.

and

you can configure how many replicas you want.

Replica A1 = R1A1

You can configure → saying that → for every document that got broken → I want to have 5 shards and 2 replicas.

So, whatever you are storing is broken into 5 pieces and after replication → you are playing with 15 pieces.

Lets assume — I have 3 databases → A, B, C.

A-node	B-node	C-node	
A1	A2	A3	→ A3 gets stored here.
A4	A5	R1A1	→ R1A1 gets stored here.
R1A2	R1A3	R1A4	
R1A5	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	

A1 gets stored here

Let us assume that, if your B-node is failed

↓

what are the parts which you don't have.

I can get it from other node.

So, at any moment, even if one node goes for a toss you can have your whole doc. retrieved like this.

→ The process of breaking your one record into multiple smaller portions is called as Shards.

For every shard → You will create replicas.

So, if you are having a 3-node cluster — If you try to create 5 shards and 2 replicas → You will not loose data even if 2 nodes fail.

This is how modern databases work.

- So for us to understand this functionality,

I'll be trying to create a stateful set of nginx.

* So, when we create replicas in stateful set we get Predictable names.
* We can access individual pod, by creating headless service and by using .....svc.Cluster-local.
* Now I can individually access everything and then I'll be able to get the info. of that Pod, bcoz I have headless service in front of it.

we all know that headless service does not have an IP-address. It gives the IP-addresses of the Pods. and we are using <pod-name>.....svc.cluster.local.

* Let us experiment with stateful sets and create nginx pods :—

we know that nginx is stateless. but I'll be trying to do something special in nginx. whatever we do now → we can apply it to any database.

→ storage → (new folder) statefulsets → nginx-statefulset-yaml :—

$ kubectl api-resources | grep statefulset

sp: true sts apps/v1

→ Docker Hub → nginx

docker playground

$ docker container run -d --name nginx -P nginx

$ docker container -it nginx /bin/bash.

Port: 49153 → Open → Welcome to nginx ✓

Let us see from which folder its coming from

root@ /# ls /usr/

o/p: bin/ lib/ share/ .

Google: nginx html file location docker.

↓

usr/share/nginx/html.

/# cd /usr/share/nginx/html

/# ls

o/p: index.html

/# cat index.html

o/p:-
```
<!DOCTYPE html>
<html>
<head>
<title> Welcome to nginx ! </title>
<style> ..... </style>
</head>
<body>
<h1> Welcome to nginx ! </h1>
- -
</body>
</html>
```

Over here lets try to create → test.html

vi test.html

o/p: command not found.

echo "<h1> i found it </h1>" > test.html

Now Go to home page → IP:3xxxx : /test.html → o/p i found it.

I'll be trying to create 3 nginx-pods and then ill be creating 3 html files with 3 different values.

Then let us try to delete nginx pods by using kubectl delete commands.

It will recreate the pod and let us see whether this exists (or) not.

- So, I'm trying to create a volume which mounts this → /usr/share/nginx/html.

Ideally, what we are about to do → shud be used for stateful application.

(nginx became a stateful appn bcoz I'm storing that html Page)

which is specific to this Pod and then for that I'm trying to create a vdume.

In reality, you will not be using this for nginx. You'll be using it for databases.

→ API reference. → StatefulSet v1 apps

↓

apiVersion: apps/v1

kind: StatefulSet

metadata

↳ name = nginx.

spec

↓

type: StatefulSetSpec.

↳ minReadySeconds = 1

↳ PersistentVolumeClaimRetentionPolicy → describes the lifecycle of PVCs created from volumeClaimTemplates.

By default, all PVCs are created as needed and retained until manually deleted.

↓

we want the same.

↳ PodManagementPolicy

↓

controls how pods are created during initial scale up.

Default is ok for us.

The default policy is 'OrderedReady' where pods are created in increasing order (pod-0, then pod-1 etc) and the controller will wait until each pod is ready before continuing. when scaling down, the pods are removed in opp order.

The alternative policy is 'Parallel' which will create pods in parallel to match the desired scale without waiting and on scaledown will delete all pods at once.

replicas - 3

selector

LabelSelector.

matchLabels:

app: nginx

template

metadata

name → nginx

labels:

app: nginx

spec:

containers

- name: nginx

image: ..

ports ... 80

volumeClaimTemplates:

- metadata:

spec:

storageClassName: managed

- -

- -

nginx-svc.yaml :-

```
---
apiVersion: v1
Kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    app: nginx
  type: ClusterIP
  clusterIP: None
  Ports:
    - name: nginx-svc
      port: 80
      targetPort: 80
      protocol: TCP
```

I need to give service name in stateful set.yaml.

Now if you want to access any pod

↓

pod name . service name .

>Kubectl get po

o/p: alpine 1/1 . . . → Let alpine pod be running → bcoz we have work with that pod.

> Kubectl apply -f .

o/p: service/nginx-svc created
statefulset.apps/nginx created.

> Kubectl get svc

o/p : clusterIP : none

> Kubectl get sts -w

o/p: nginx ready
0/3
:
3/3.

If is taking time.

(bcoz PV needs to be created and it needs to be mounted.

so we are creating disks also.

If it is a stateless app? → we don't need to do all of this.

Ideally you shld be running databases over here.

nginx-statefulset.yaml :

```
---
apiVersion : apps/v1
kind : StatefulSet
metadata :
   name : nginx
spec :
  minReadySeconds : 1
  serviceName : nginx-svc
  replicas : 3
  selector :
    matchLabels :
      app : 'nginx'
  template :
    metadata :
      name : nginx
      labels :
        app : 'nginx'
    spec :
      containers :
        - name : nginx
          image : nginx
          ports :
            - containerPort : 80
          volumeMounts :
            - name : htmlhome
              mountPath : /usr/share/nginx/html
  volumeClaimTemplates :
    - metadata :
        name : htmlhome
      spec :
        accessModes :
          - ReadWriteOnce
        resources :
          requests :
            storage : 1Gi
        storageClassName : managed
```

This will create a Persistent volume claim (PVC) Per Pod.

Now, I'll get 3 pods with name nginx-0, nginx-1, nginx-2

/# curl nginx-0. nginx-svc

o/p : curl: not found

<Pod-name>.<Service-name> (Headless)

/# apk add curl

/# curl nginx-0.nginx-svc

o/p: <html>
<head> <title> 403 Forbidden </title> </head>
<body>

403 Forbidden

</html>

It says that it is forbidden.

/# nslookup nginx-svc

o/p: Name: nginx-svc.default.svc.cluster.local
Address: 10.244.1.16
Name: — — —
Address: 10.244.1.17
: — — —
: 10.244.0.40

API ref
Look into service name in
Stateful set v1 core

Pods get DNS/hostnames that follow the pattern:

pod-specific-string.serviceName.default-sv.
cluster-local.

/# curl http://nginx-0.nginx-svc.default-svc.cluster.local

o/p: Forbidden.

↓

cd /usr/share/..

ls.

→ There is no index.html. → That is the reason

we created an Empty volume.

/# curl http://nginx-0.nginx-svc/sts-html

o/p: <h1> Pod0-STS </h1> ✓

<Pod1 -1 ✓

<Pod2 2 ✓

* Now create a different html page for every Pod.

$ kubectl get po

o/p: alpine Running

nginx-0 "
nginx-1 "
nginx-2 "

→ names are predictable

Naming is predictable in stateful sets

$ kubectl delete po nginx-0

o/p: pod 'nginx-0' deleted.

$ kubectl get po

	Ready	Status
alpine	1/1	running
nginx-0	0/1	ContainerCreating
nginx-1	1/1	running
nginx-2	1/1	running

→ It is not creating nginx4.
It will again try to create nginx-0

↓

This is the major difference when it comes to deployment & stateful set

$ ~~kubectl exec nginx-0 --~~

Get into Pod & create.

$ kubectl exec -it nginx-0 -- /bin/bash

/# cd /usr/share/nginx/html

/# echo "<h1> Pod0-STS </h1>" > sts-html

/# Exit.

do same for nginx-1 & for nginx-2.

Pod1-STS Pod2-STS

So, I have created 3 different html files in 3 different pods which are stored on 3 different persistent volumes.

$ kubectl get svc

o/p: name
nginx-svc

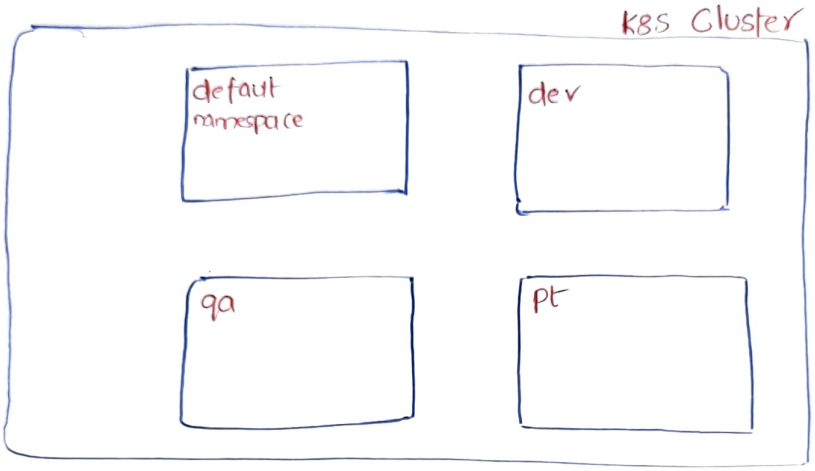
$ kubectl exec -it alpine -- /bin/sh

/#

* Now, some of the K8s resources will get created inside this namespace and some don't get created inside this namespace.

- In reality, what your Orgzns will do is → They will try to create dev namespace, qa namespace, pt namespace

So this namespace is like a virtual cluster.



$ kubectl api-resources

O/p: we can see where are some resources where namespace = true and some resources where namespace = false.

→ Namespace = true means → they are created within that namespace.

Eg: pods → for pods namespace value is true.

But

For Eg: PersistentVolumeClaims → namespace is false.

↓

what it means is → when you create a K8S pod → they belong to some namespace

As we saw just now → to access K8S service → we have used

service-name.default.svc.cluster.

↓

name of your namespace.

Eg: Services → for services → namespace = true → so whenever you are creating a service → it exists in a namespace.

* we sent a curl request from alpine pod.

> kubectl delete po nginx-0.

o/p: Pod deleted

wait for Pod to be up

> kubectl get po -w

o/p. ✓ ✓ ✓

lets send request to nginx-0. from alpine

curl http://nginx-0.nginx-svc/sts.html

o/p: <h1> Pod0-SIS </h1>

→ So, even if I deleted the pod-it was recreated and we got the same data back

Kubernetes namespace :-

- Generally whenever we are working with K8S, there are various namespaces that exist.

$ kubectl get ns

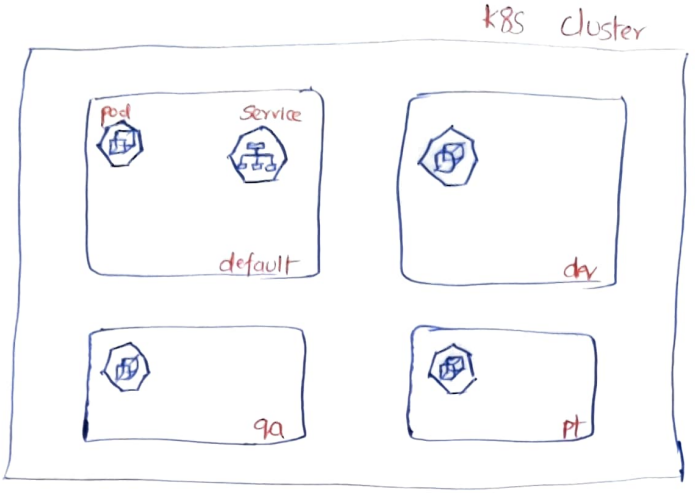
o/p: default
gatekeeper-system
kube-node-lease
kube-public
kube-system

- Any namespace that starts with 'kube-' is something which is created by K8S

→ Gatekeeper-System → is the namespace created by Azure to do some of its activities.

- All these days we were using a namespace called Default.

* Lets assume, we have 3 nodes and K8S cluster. We are assuming that Pods get created in these nodes in cluster.
But in reality it doesn't work that way.
In each of the K8S cluster, what K8S does → it tries to create something called as namespace. This what we're trying to speak about is a default namespace.



So, if you want to access within this cluster →

you have to give → resource-name . namespace . svc.cluster.local.

↓

this is fully qualified domain name.

* The basic idea is to separate one section of pods with other sections.

> kubectl delete -f .

> kubectl create ns dev

o/p: namespace/dev created.

> kubectl get po

o/p: alpine 1/1 Running → I'll be getting pods. But I'll be getting only pods of bydefault namespace which is 'default'.

> kubectl get po --namespace default

o/p: alpine 1/1 running (All these days whenever you were executing the cmd → it was equivalent to this.)

> kubectl get po --namespace kube-system

o/p:-

Name	Ready	Status ...
ama-logs-85984	3/3	
ama-logs-fw	3/3	
ama-logs-rs	2/2	
:	:	

To run your k8s cluster, there are already lot of pods doing the job for you. But these are all created by k8s cluster. we don't need to worry about it.

> kubectl get po --all-namespaces

o/p: You have these many pods which are running across diff.t namespaces.

- But, for the user, acc to the permissions which he has, he has permissions on particular namespaces.

* You don't give any namespace → it will use 'default'.

$ kubectl get po --namespace dev

o/p. No resources found.

In metadata, there is one interesting thing called as namespace.

(Namespace defines the space within which each name must be unique.)

(An empty namespace is equivalent to "default" namespace).

Eg:- nginx-statefulset.yaml → apiVersion: apps/v1 (I'm creating this stateful set in a namespace called dev.)
kind: StatefulSet
metadata:
  name: nginx
  namespace: dev

nginx-svc.yaml (in dev namespace)
apiVersion: v1
kind: Service
metadata:
  name: nginx-sv
  namespace: dev
  :

> kubectl apply -f .

o/p: created

$ kubectl get sts

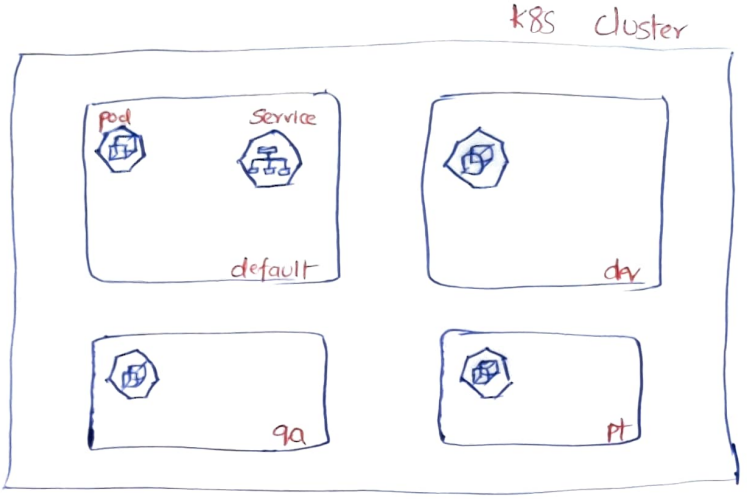
o/p: No resources found in default namespace. (Bcoz it is querying only default namespace)

$ kubectl get sts --namespace dev

o/p:
Name	Ready	
nginx	1/3	19s

* So, in your org^zs, basically what wud happen is you will have a production k8s cluster, staging/UAT k8s cluster and then you wud have one k8s cluster for dev, test- and all testing Environments.

* Your org^zs ask you to specify the namespace. Depending on which Env you want to deploy — you have to pass that namespace.



So, if you want to access within this cluster →

you have to give → resource-name . namespace . svc.cluster.local.

↓

this is fully qualified domain name.

* The basic idea is to separate one section of pods with other sections.

> kubectl delete -f .

> kubectl create ns dev

O/p: namespace/dev created.

> kubectl get po

O/p: alpine 1/1 Running → I'll be getting pods. But I'll be getting only pods of bydefault namespace which is 'default'.

> kubectl get po --namespace default

O/p: alpine 1/1 running (All these days whenever you were creating the cmd → it was equivalent to this.)

> kubectl get po --namespace kube-system

O/p:-

Name	Ready	Status ...
ama-logs-85984	3/3	
ama-logs-fw	3/3	
ama-logs-rs	2/2	
:	:	

To run your k8s cluster, there are already lot of pods doing the job for you. But these are all created by k8s cluster. we don't need to worry about it.

> kubectl get po --all-namespaces

O/p: You have these many pods which are running across diff.t namespaces.

$ kubectl get po

O/p: alpine

$ kubectl exec -it alpine -- /bin/sh

I want to access the other pod. How do I do that?

/# curl nginx-0.nginx-svc.dev.svc.cluster.local

O/p: test.

Full name includes
↓
podname.namespacename.cluster.local
since i'm using headless service over here
↓
( )

Now lets directly try to access this

/# curl nginx-0.dev.cluster.local.

O/p could not resolve host

/# nslookup nginx-0.dev
Server: 10.0.0.10, Address: 10.0.0.10:53

/# nslookup 127.0.0.1

O/p: server can't find 1.0.0.127.... : servfail.

Google: Pods fqdn

↓

DNS for services and Pods.

pod-ip-address.my namespace-pod.cluster-domain.example. (In general a Pod has the foll. DNS resolution)

Eg:- if a pod is in default namespace.
domain name for your cluster is = cluster.local.
then pod has a DNS name
↓
172-17-0-3.default-pod.cluster.local.

→ Any pods exposed by a Service have the foll DNS resolution available.

↓

pod-ipaddress.service-name.my-namespace.svc.cluster-domain.example.

Google: get fqdn in linux.

↓

hostname --fqdn.

/# hostname --fqdn

O/p: nginx-0.nginx-svc.dev.svc.cluster.local (Since there is a headless service over here → it gives headless service's name but actually → you'll get ip-addr stuff

> kubectl get svc --namespace dev

o/p:- nginx 2/3 56s

But →

$ kubectl get pv

- Persistent volumes do not belong to any namespace.

So, they are shared resources.

- So, wherever you are seeing → with namespaced value as false.

$ kubectl api-resources →

They don't belong to any particular namespace.

That means that They are not different acc. to the namespaces.

- So, Persistent volume Claim can be created within the namespace, but persistent volume can be shared across namespaces.

So, for dev namespace as well as for test namespace → I might be using same persistent volume.

→ So, this namespace is an abstract idea → where you can create cluster within cluster and this is a logical cluster.

Note: Not every K8s resource will be part of namespace.

Ex: volumes are not part of namespace.

- So, if you want to communicate from a pod which is running in one namespace to the other namespace → you have to use the fully qualified domain name.

ie.: <pod-name>.<namespace-name>.svc.cluster.local.

> kubectl exec --namespace dev -it nginx-0 -- /bin/sh

cd /usr/share/nginx/html

ls

o/p lost+found. There are nothing in this.

echo "test" > index.html

ls -l o/p index.html

ETCD Database backup and restore

→ ETCD :-

* ETCD is a distributed reliable key-value store that is simple, Secure & fast.
* The ETCD database stores info. regarding the cluster such as nodes, pods, configs, secrets, accounts, roles, bindings etc.
* Every info. you see when you run the 'kubectl get' command is from the ETCD database.
* ETCD starts a service that listens on port 2379 by default.

~# etcdctl -version

O/P: 3.3.13

API version: 2.

we need API version 3 for this.

etcd is deployed as a static pod.
↓
It will be in kube-system n.s

kubectl -n kube-system get pod

O/P:

Name	Ready	Status
coedns		
etcd-controlplane	Running	1/1
kube-apiserver-controlplane		
kube-controller-manager- controlplane		
kube-flannel-ds		
kube-proxy		
kube-scheduler - controlplane		

→ All are deployed as static pods (done by kubeadm tools)

kubectl -n kube-system describe pod etcd- controlplane

we get all the info. regarding key, certificates

↓
volumes

etcd-certs: Type: HostPath
Path: /etc/kubernetes/pki/etcd
HostPathType: DirectoryOrCreate

etcd-data: Type: HostPath
Path: /var/lib/etcd
HostPathType: DirectoryOrCreate.

kubectl get deployments.apps

o/p : blue 3/3
red 2/2

kubectl -n kube-system describe pod etcd-controlplane

o/p :-

etcd:
--cert-file = /etc/kubernetes/pki/etcd/server.crt ✓
--data-dir = /var/lib/etcd
--name = controlplane
--peer-cert-file = /etc
--key-file = /etc/kubernetes/pki/etcd/server.key ✓
--trusted-ca-file = /etc/kubernetes/pki/etcd/ca.crt ✓

Google :-
Kubernetes
↓
search: etcd snapshot

Command:

ETCDCTL_API=3 etcdctl --endpoints = https://127.0.0.1:2379 \
--cacert = <trusted-ca-file> --cert = <cert-file> --key = <key-file> \
snapshot save <backup-file-location>
↓
Eg: /opt/snapshot-pre-boot.db

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
> --cacert = /etc/kubernetes/pki/etcd/ca.crt --cert = /etc/kubernetes/pki/etcd/server.crt
--key = /etc/kubernetes/pki/etcd/server.key \
> snapshot save /opt/snapshot-pre-boot.db

o/p :- Snapshot saved at /opt/snapshot-pre-boot.db

(Backup of etcd is now taken)

Now something disastrous has happened and etcd is not responding
↓
kubectl get deployments.apps
o/p: No resources found → data lost. How to recover ?

we can just restore etcd.

Command for restoring ETCD :-

ETCDCTL_API=3 etcdctl --endpoints 10.0.0.9:2379 snapshot restore snapshotdb

↓

/opt/snapshot-pre-boot db

we are moving our db to particular locations. So - we will use --data-dir=/var/lib/etcd-backup.

ETCDCTL_API=3 etcdctl --data-dir=/var/lib/etcd-backup snapshot restore /opt/snapshot-pre-boot.db

o/p - etcdserver/membership: added member 8e9... 94d [http://localhost:2380] to cluster cd...32

cd /var/lib/

ls

o/p: apt cni containerd docker dockershm etcd etcd-backup git kubelet ....

↓ we can see that folder has been created.

cd /etc/kubernetes/manifests/

Here we have all config files (static pods)

ls

o/p: etcd.yaml kube-apiserver.yaml kube-controller-manager.yaml kube-scheduler.yaml

vi etcd.yaml

You need to change the path →

Previous

volumes:

path: /var/lib/etcd

Now

/var/lib/etcd-backup.

When we save this file,

It will do proper linking.

docker ps | grep etcd

There is nothing pending — linking has been established.

kubectl get deployments.apps

o/p: blue 3/3 ...

red 2/2 ...

* Kubernetes Cluster Upgrade *

- At a time Kubernetes supports only 3 minor version.
- Currently, Main version is 1 and minor version is 21. (1-21) { On an avg they release 4 versions within a year
- K8s releases minor versions very frequently in a year.
- We will be upgrading the cluster using Kubeadm utility.
- Upgrade will be carried out one node at a time, starting with master node.
- We need to drain the master/worker node before doing upgrade.

→ Master node Upgrade Command :

* Kubeadm upgrade plan
* Kubectl drain controlplane --ignore-daemonsets
* apt update
* apt install kubeadm=1.19.0-00
* kubeadm upgrade apply v1.19.0
* apt install kubelet=1.19.0-00
* systemctl restart kubelet
* Kubectl uncordon controlplane. (making master-node ready to work).

→ Worker node Upgrade Command :-

→ Kubectl drain node01 --ignore-daemonsets
→ ssh node01
→ apt install kubeadm=1.19.0-00
→ kubeadm upgrade node
→ apt install kubelet=1.19.0-00
→ systemctl restart kubelet
→ kubectl uncordon node01

Kubernetes upgrade from v1.18 to v1.19 :-

Lets monitor the Cnt first.

$ kubectl get node

o/p: controlplane Ready v1.18.0 (version)
nodeo1 notReady v.1.18.0

Lets drain the Master-node first.

$ Kubectl drain controlplane --ignore-daemonsets

o/p: node/controlplane cordoned.

evicting pod kube-system/coredns

node/controlplane evicted

$ kubectl get node

o/p: controlplane Ready, SchedulingDisabled
nodeo1 Ready.

$ kubectl get pod → we will find only pods from node

o/p: gold-nginx . . running : . node 1 & not find any pod from Control plane.

Lets start the Upgrade Process.

$ apt update

$ apt install kubeadm=1.19.0-00

$ kubeadm upgrade apply v1.19.0

o/p. Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.

$ apt install kubelet=1.19.0-00

$ systemctl restart kubelet

$ kubectl get node

o/p- controlplane v1.19.0 Ready, schedulingdisable
v1.18.0 Ready

Now uncordon to allow master node to work.

$ kubectl uncordon controlplane

o/p: node/controlplane uncordoned

$ kubectl get node

o/p:		status	Version
	controlplane	Ready	v1.19.0
	node01	Ready	v1.18.0

$ kubectl drain node01 --ignore-daemonsets

Now its evicting all the pods we have on node01 to control plane compute

$ kubectl get pod -o wide

o/p:	Name	Ready	status	Node	Nominated node
	gold-nginx...	1/1	running	control-plane	<none>

Pod which was running previously on node1 is now running on controlplane.

So, we moved it, bcoz we want to upgrade worker node.

controlplane $ kubectl get node

o/p:			
	controlplane	Ready	v.1.19.0
	node01	ready, schedulingDisabled	v.1.18.0

$ ssh node01

node1 $ apt update

$ apt install kubeadm=1.19.0-00

$ kubeadm upgrade node

$ apt install kubelet=1.19.0-00

$ systemctl restart kubelet

ctrl+D to come out from Worker node.

$ kubectl uncordon node01

$ kubectl get node

o/p: Both are Ready. ✓

Controlplane $ kubectl get node

o/p:			
	controlplane	Ready	v.1.19.0
	node01	Ready, schedulingDisabled	v1.19.0