Lec-51- Apr-23

Kubernetes (K8s)

Need for K8s :-

Right now, we know that we can take any app^n and then we can containerise that app^n and make it run on a docker platform.

So what was the basic need for K8s ?

Needs :

1) High Availability (HA) :-

Lets assume we are running our App^n in D^r C^r. and for some reason that C^r is in exited state/ stopped state. So your app^n will have a downtime. So, to start the app^n again → we may use D^r C^r run/start cmd and make the C^r work.

The problem with this approach is → This approach works well when you are working in Single System.

So, if there is some software which recognizes — that when C^r goes down, it will automatically do the work of — starting the C^r back → so that your app^n is always up & running → if there is some app^n that can do that → that will help us.

So, when we run our app^{ns} in docker C^r and if the C^r fails, we need to manually start the C^r (1st case → container is down)

→ If the node i.e. the machine fails → all the C^{rs} running on the M/C shud be re-created on the other M/C (2nd case → Here M/C is down)

- K8s can do both of the above (which docker cannot do) (Docker swarm does it)

2) Autoscaling :-

For Eg. we are running an e-commerce app^n in d^r C^r and there is a discount season (or) sales (3 days of discounted Sale) → There will be a lot of traffic to your website. when there is lot of traffic — you cannot run your app^n

in exactly the same number.

For Eg: Earlier you were trying to run your app^n is 10 C^{rs} → you might need 30 now.

The point is, if i'm trying to run by using D^r commands → I have to look at traffic and I have to increase manually, the no. of C^{rs}.

But, k8s can do this based on the configurations which we provide.

- Containers don't scale on their own.
- Scaling is of two types
  - * Vertical Scaling
  - * Horizontal Scaling.
- k8s can do both horizontal and vertical scaling of C^{rs}.

Vertical scaling → 256 Mb
1 cpu → 2 cpus ↓ 512 Mb.
Horizontal → Increasing no. of C^{rs}. 10 to 20.

3) Zero-Down time Deployments :-

Generally when we run our app^n in C^{rs} → it is not guaranteed that, the same C^r will be running forever → we will always get new releases.
when we are moving from older version to newer version → we wud want zero down-time (or) near zero down-time deployments.

Eg: Yesterday you were using nop Commerce. Let us assume, developers gave a new version with new set of features → then you might need to update nop Commerce to enable your customers to use new features.

↓
So if we do this manually → docker C^r rm (removing existing C^r) and create the new C^r with new image tag → there may be downtime.

k8s can handle deployments with near zero-down time deployments (near zero (or zero depends on number of nodes you have).

* There are 3 major advantages.

Google: Kubernetes → There are many other features.

One more thing is

Even if your new release is not working → you can go back very easily (bcoz K8s can remember the older versions - which your Dr Cs cannot).

↓

called as Rollback

Giving a new release = Roll-out

- K8s can handle rollout (new version) and roll back (undo new version ⇒ previous version)

* K8s is described as 'Production grade Container Management'

(So K8s is not replacing Docker. so whatever we learnt in Dr will be continued thru K8s but to run our appn in nodes → we will not be using Dr commands anymore, we wud be using container management).

- History of Kubernetes :-

Even before Dr started, very few companies were using Cr techgy. So Google from early 2000's was using Cs → but they were not Dr Cs, they were linux Cs.

(In Google, → there is loads of traffic → everything runs on Cs)

* Google had a history of running everything on Containers.

* To manage these containers, Google has developed Container Management Tools (these are inhouse tools)

↓

- Borg (used by Google internally)
- Omega (right now, Google is still using Omega).

* With Docker publicizing Containers (where it made easy to use Cs), with the experience in running & managing containers, → Google has started a project - Kubernetes- (so both of earlier tools → Google has developed internally using C++ but K8s was developed in Go language)

Google started a project K8S and then handed it over to Cloud Native Container foundation (CNCF)

Now CNCF (in which Google is a member) manages Kubernetes project.

K8S project is available on Github.

Does Google have any Competitors?

Competitors :-

* Apache Mesos
* Hashicorp Nomad
* Docker Swarm

But K8S is clear Winner.

Initially AWS has built a product which is called as ECS (Elastic Container Services) where they wanted to do what K8S is doing. So ideally, amazon was a competitor of K8S. But even they had to implement K8S on that cloud.

** Terms :-

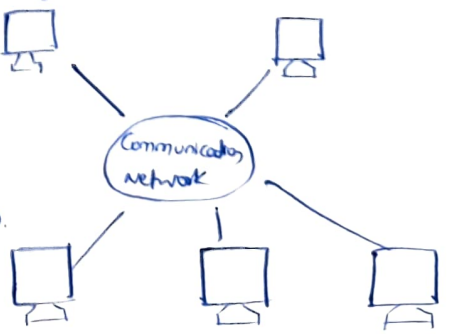
- Distributed System :

Node and cluster are kind of part of Distributed System.

You might have heard of Client-Server model (where we'll have clients who speak with servers)

The way Distributed Systems work is → You wud not have one server doing the job it wud be multiple systems doing the same work and work might be assigned to any of these M/c's or to all of these M/c's. (depending on model which you have chosen)

Bitcoin — Blockchain. → The way this techy works is ⇒ work will be assigned to everyone. whoever does the work 1st he will be rewarded. That is the approach they have taken.

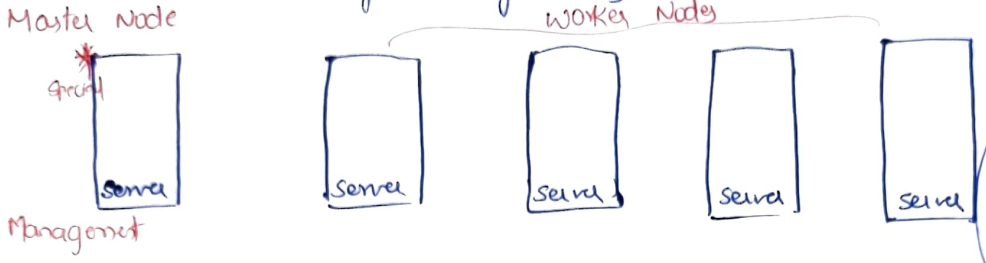


* Generally we have client and server.

Here what we are trying to tell is → 9 will have group of servers.

And each of that server is called as Node

and in distributed systems generally, we need someone to manage these nodes.



Consensus is a way where these nodes figures out who is their leader. (For that there are many algorithms)

* Over here, we have some nodes which are little bit special than others.

The reason is they do management. (whether they do the work or not is our choice)

This node can do the management and also do the work

we have { Node which does the management
Node which does the work.

→ For that reason, we basically call it as Master Node

The nodes which does the work are called as Worker Nodes.

Any work which you assign will reach Master Node ~~first master~~

There are 2 approaches :-

1) Give work to Everyone (or)
2) Find one system to do the work.

In our case, it would be (2).

So, master node will figure out on which ~~node~~ node the work has to be done.

↳ so node does the work and respond back to master.

Even though there are 5 servers, → to the outside world → it would look like one huge Server.

Any tech^gy that does this → is called as cluster.

- So cluster is combination of multiple machines – But it would look as if it is a single M/c.

Eg: MySQL cluster, Postgres Cluster

↓ will have multiple postgres dbs (But to the user of that it would look as if he is speaking with one M/c)

Eg: Hadoop clusters

↓

lot of systems doing hadoop activity or Big data activity.

* So, you speak with cluster ⟶ & cluster gets the work done.

⟶ In the case of K8S, work is - running your app^n in a Docker container (cluster that is responsible for running ©^rs)

So, the work for that cud be anything to do with management.

Eg: Database clusters ⟶ Their work is about storing data.

In Distributed System

↓

there is no client, there is no server.

Anything can become client & anything can become server.

Stateful Applications: are the app^ns which store data somewhere in the local system.

Stateless Applications :- do not store it locally. They store data in an external system that cud be database or something else.

State :-

state is some meaningful information your app^n (or) anything.

Monolith :-

Monolith is running whole of your application as if it is one huge component.

so running whole of e-commerce in one server. (or) designing the whole of eco- to run in one server.

Microservices :-

You break that huge services into small components based on the functionality (or) based on requests (or) based on domains etc.

Task :- find simplest definitions :

Distributed System	Stateful Applications
Node	Stateless Applications
cluster	Monolith
state	Microservices

* Declarative Approach v/s Imperative Approach :-

is where you tell K8S
I want to run 3 MySQL containers
In K8S, whenever we are speaking, we always try to speak about →
→ what is it that we want?
↓
That is something which we basically call as Desired state.
So, in K8S, you alway speak about Desired state (what is that your Appn wants?)
&
Since K8S can manage Containers, it can basically create those many ©rs (or) it can do that config whatever you have asked.

For running that there are 2 ways

1) Imperative
↓
means you type commands

2) Declarative means
↓
you try to write a manifest file (nothing but some yaml file)
Manually running the commands to create Nop Comm → was a Imperative way.
Writing a Docker-Compose → is a Declarative way.

* So, Organizations expect you to use the Declarative approach → bcoz Decl. appr. can be Version Controlled.

* Pet v/s cattle :- It is about mindset.
Generally, we run our appn in a ©r. If that fails → we will login and fix the same.

* In classic, on-premise cases, Our admins deals with servers as if they are pets
The moment we start speaking about distributed systems, cloud computing → we start treating all of them as cattle.
So if the thing is not working, get a replacement. Ensure that things are working. If the failure happens all the time, then we try to fix it.

* So, what K8S tries to convey is
↓
You are trying to run Distr. Systems with loads of Containers
fixing the problem in one ©r
↓
doesn't make much sense. Better replace it and try to allocate on a different node.

* We will be using cattle mindset (replace the things which are not working) and not pet mindset (trying to fix things)

previous we wrote Docker-compose file and manually ran the commands
↓
It is imperative way.

* K8s is not designed only for Docker

⇓

Initially K8s used Docker as a main Container Platform and Docker used to get special treatment.

- From K8s 1.24 Onwards, special treatment to Docker has stopped.

What is special treatment ?

⇓

K8s is designed to run any Container technology, for this K8s expects Container Technology to follow K8s Interfaces.

→ All the other Container Technologies have implemented this. But Docker never cared. Since Docker was not caring

what K8s did is → K8s has created a shim Component so that, using this shim you can speak with Dr ©s. (Docker shim (or) Docker CRI shim)

From K8s 1.24 → CNCF has stopped writing Docker shim.

(But now we can still run Dr in K8s).

⇓

When we were speaking about Dr, we said that, Dr creates OCI Specification images

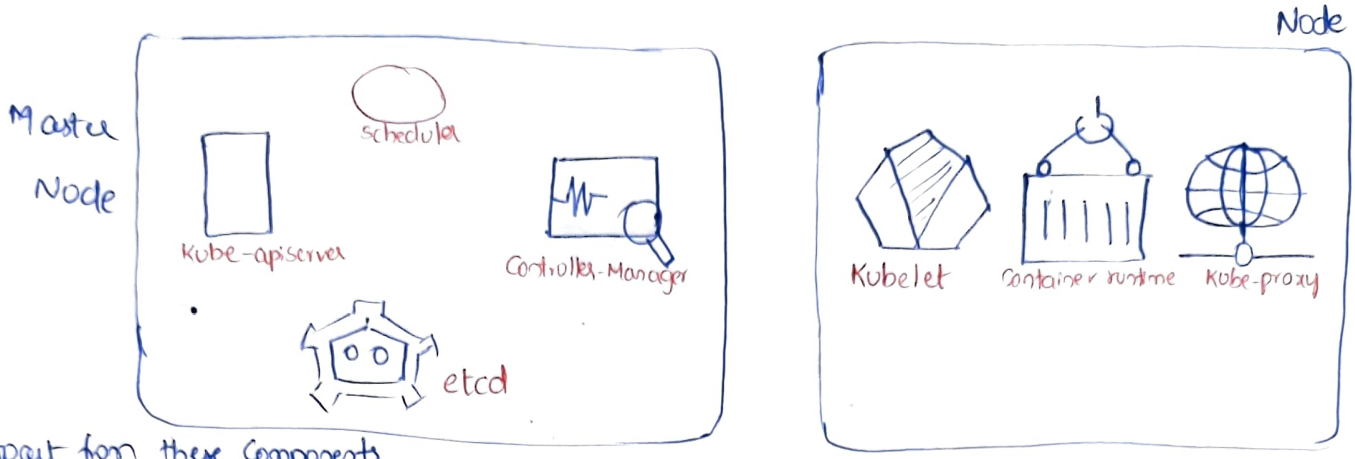
So, whenever you create a Dr image → that image can be run by any other Platform → bcoz both of them follow the OCI Spec.

So you can create a Dr image and run that in a diff·t ©r Tech.

⇓

That was the reason why K8s didn't stop supporting this activity.

* Kubernetes Architecture *



Apart from these components we also have → Clients:

- kubectl
- any rest based client

What K8s is trying to tell is → If you want to use K8S,

You and me want to use K8S → Once the cluster is up — we wud be using Kubectl or we can have our own client.

* whenever you are creating K8s cluster → there are 2 kinds of nodes < Master Node, Node.

- On Master Node there are 5 components < 4 comes into play in on-premises; 1 comes into play on cloud.

4 components are →
1) Kube-apiserver
2) etcd
3) Scheduler
4) Controller-Manager

- On Node → you have
  1) Kubelet
  2) Kube-proxy
  3) Container runtime

These are the major components of K8S which do the work

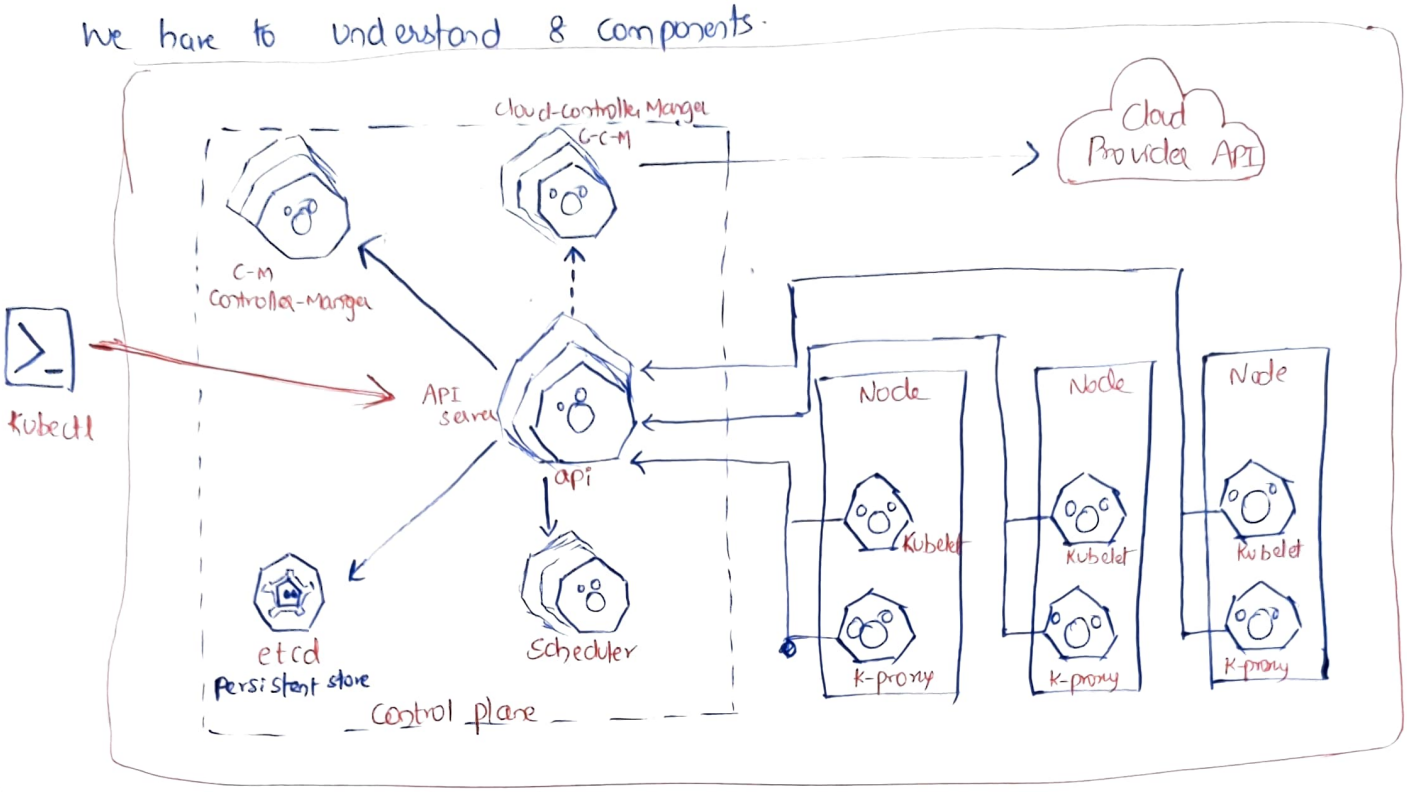
- Using kubectl you wud interact with Kubernetes

Logical View :→ 

So, you & me to speak with K8s, to get the container mgt → we'll be using kubectl.

* Actual View:- With Kubectl You wud be speaking with Kubernetes cluster.

We have to understand 8 components.



CCM → cloud controller Manager

↓

It is specifically written for any cloud who wants to implement K8S as a Service.

When we say that we are running K8S on Azure (so Azure implements this)

" " " AWS (AWS " ")

Here, we didn't use the term 'Master', but rather we used the term 'Control Plane'

* So, K8S is production-grade ©ⁿ Mgt. Once your App^n has ©^ns — K8S can do whatever is necessary to take your ©ⁿ techᵍʸ into production where n no. of users can basically use your app?

↓

How can you give such guarantees without knowing about K8S?

Bcoz K8S was developed 0.b.o. 2 popular Container Mgt systems which Google has developed where all of your gmails, youtube ... are running.

So this techᵍʸ has to be perfect.

Orgⁿˢ who have implemented K8S → Eg: adidas, squarespace, Yahoo Japan, Spotify, WINK, Ygrene etc.

* Kubernetes Components *

→ Control Plane Components (Master Node Components) :-

1) Kube-Apiserver :-

It will look as if it is Kubernetes.

It is one of the most imp. component of K8S.

what is the reason for that is?

All the communication in the K8s is handled by kube-apiserver.

So when you want to get anything done with K8s cluster, → you speak with kube-apiserver.

when any of the other components need to communicate with any other thing → It is again kube-apiserver. So it handles all the comm's of K8s cluster.

Let it be internal (or) External communication.

* kube-apiserver ~~can be communicated~~

exposes functionality over HTTP(s) protocol and provides REST API for communi

REST API :- is a communication style where you use http protocol to get the work done.

So, when you want to speak with K8s, what is happening is → You are speaking with kube-apiserver ——→ and kube-apiserver handles all the comm's

So, whenever you type any command in K8s → kube-apiserver is involved.

when node is down (or) when ©r is down → new one has to be created → communication related to this is handled by kube-apiserver.

2) etcd :-

It is not developed product (database) by K8s.

I can use etcd in any Distributed systems.

when K8s was being developed, → to store the state of K8s we need some database

Anything in K8s has to be stored somewhere.

how many nodes are there
" " ©s did you create
what are its configs

That somewhere is etcd.

- This is memory of K8s cluster.

(If you clean this — it will forget everything).

* Kube-apiserver is stateless, but etcd is stateful.

K8s never wants you to handle etcd directly.

If you want you can store etcd cluster outside K8s cluster also. (not recommended practice)

If you want to take backup of K8s cluster → just take backup of etcd.

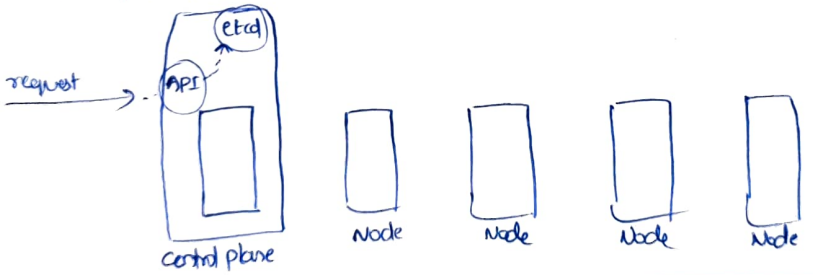
Bcoz all the info. of K8s cluster is stored in etcd. (what you have asked? what is present? what is not present?) will be in etcd cluster.

3) Scheduler :—

Now, in K8s whenever you ask to create something new → for eg → (I want to create MySQL container. → info.)

→ API server stores that info in etcd (saying user has requested for creating MySQL)

we have 4 other nodes where work can be done.



Now, there will be a scheduler → Scheduler's responsibility is to schedule whatever you have asked one of these nodes.

whenever you have to create anything new → On which node it has to create — what is the best fit for whatever you have asked → That is decided by scheduler.

* So, scheduler also doesn't speak directly with etcd.

It speaks with API manager. → API manager speaks with etcd.

* Scheduler is responsible for creating K8s objects and scheduling them on right node. (This comes into play only when you have to create)

4) Controller (or) kube-controller manager :-

Initially, there was only one controller component called as kube-controller manager.

Now there are many controllers

But the basic idea of controller is

→ Controller is a watchdog (or)
It is a loop that runs all the time.

Eg: you said that you want 3 MySQL ©'s.

This Controller will always check whether 3 are running (or) not.

You asked for 3 → If only 2 are present → then controller will figure out that → desired state is 3 → & actual state is 2. So I need to create one more ©

* Controller Manager is responsible for maintaining desired state mentioned in the manifest.
* This is reconciliation loop that checks for desired state → and if it mismatches → doing the necessary steps is done by Controller.

For Eg:- If container is down → creating one more ©

If count is not maintained → trying to create one more
(asked for 3 ≠ 2 found)

If node is down → all the ©'s that used to run on that node
↳ scheduling them on other nodes.

- It looks like single component, but within it, it has:

→ Node Controller : for noticing & responding when node goes down.

→ Replication Controller: for maintaining the correct no. of pods for every replication controller object.
(Asked for 3 (but only 2 are running))

→ Endpoints Controller : Populates the Endpoints object.
(Endpoint is how do you access the App?)

(Scheduling is done by scheduler but Controller tries to initiate that)

(Creating something new is done by scheduler. But controller tells the scheduler to do so)

Cloud Controller Manager (CCM) :-

In the case of cloud, when you are running K8s on AWS/Azure/GCP etc (EKS, AKS)

Elastic Kubernetes Services (K8s on AWS)

Google Kubernetes Engine (GKE on GCP)

Azure Kubernetes Service (AKS Azure)

- Anything that K8s needs to communicate with the cloud provider → to do something special → One of that special activity is creating disk / EBS
  Eg: LoadBalancer, VPC, network etc.

CCM tells that → I have a volume now → TO store that volume I need a Disk storage → so in Azure go and create a disk

" in AWS go and create EBS

↓

These kind of things are handled by CCM.

Any additional cloud-specific logic will be handled by CCM.

These 5 components make up Control plane.

Generally when we install K8s in our Orgs, we will not have CCM Bcoz CCM is a component specific to the cloud.

So when you are installing it on Linux servers → you wud have only 4 components.

When AWS is running on AKS → then they will have that Extra Component.

* The difference b/w control plane and Master Node is

All of these 4 components → how do you arrange them is your choice.

If all of 4 components I'm running it on One Server → that server is called Master Node

* Node Components *

1) Kubelet :-

This is an agent of the control plane.

It is not intelligent. It listens to what control plane has to say.

* it says
Create a new C^r → kubelet will listen and does whatever is necessary.

Give the status → " " give the status.

Eg: when we say → scheduler schedules a node → scheduler is basically speaking with kubelet. and kubelet will do the work and report back the status. → status is stored in etcd.

* Any work that has to be done on the node is handled by kubelet.

2) Container runtime :-

In our case - it is Docker.

↓

- Container Technology to be used in k8s cluster.

The docker which you have installed on every node is container runtime.

- kubelet → to do its work → needs to take help from C^r runtime.

If you want to create a C^r

↓

kubelet will speak with C^r runtime — to create C^rs.

3) Kube-Proxy :-

→ This component is responsible for networking for containers on the node.

wherever you create C^s → you need to access those C^s →

kube-proxy does whatever is necessary to provide networking.

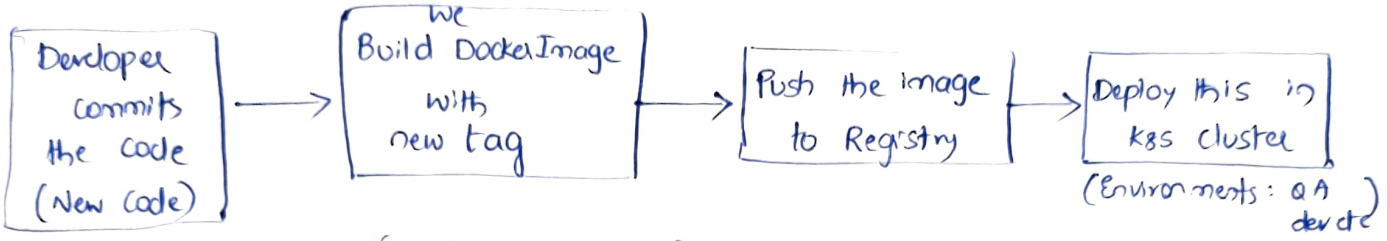
* No component can speak with other component (except via api manager)

All the communications are handled by kube-apiserver.

* Kubectl :-

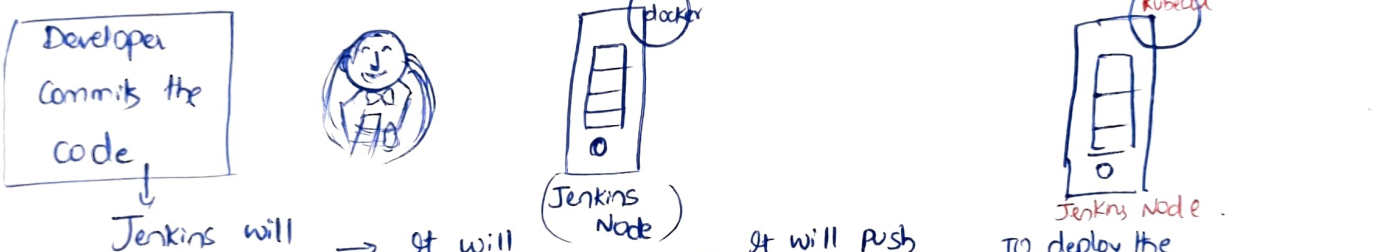
→ This is a Command Line tool that can be installed on the machine from which you communicate to k8s cluster.

→ This tool is created to make commun^n with api-server simplified.



Basic Workflow (CI/CD workflow)

If it is Jenkins

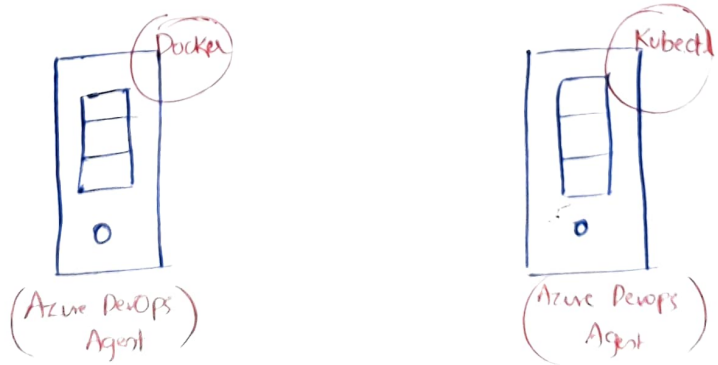


Jenkins will know about it. → It will build the D^r Image → It will push it to registry → TO deploy the app^n we wud have a Jenkins Node with Kubectl installed in it. (Kubectl will try to deploy the app^n)

If it is Azure Devops

Azure Devops is also a CI/CD tool with Project Management features.

Same story



* Kubectl has a config file (KUBECONFIG) which contains
  - api-server information
  - Keys to communicate with api-server. (bcoz ^ comm^n with it is not open to everyone. It is present only to a specific set of users)

* Kubectl allows to communicate with cluster to create resources
  - Imperatively : Type commands
  - Declaratively : write manifests (YAML files)

* What is K8S Manifest ?

This is a yaml file which describes the desired state of what you want in/using K8S cluster.

Kubectl has
↓
- config file (KUBECONFIG) → which speaks about
  - which api server it needs to speak with
- manifest file (has ^ info about what has to be done)

↓

So K8S reads this Manifest files and converts them into Rest API calls and speaks with API server bcoz API server doesn't understand yaml files. It understands http calls.

Kube-api-Server can understand http & json

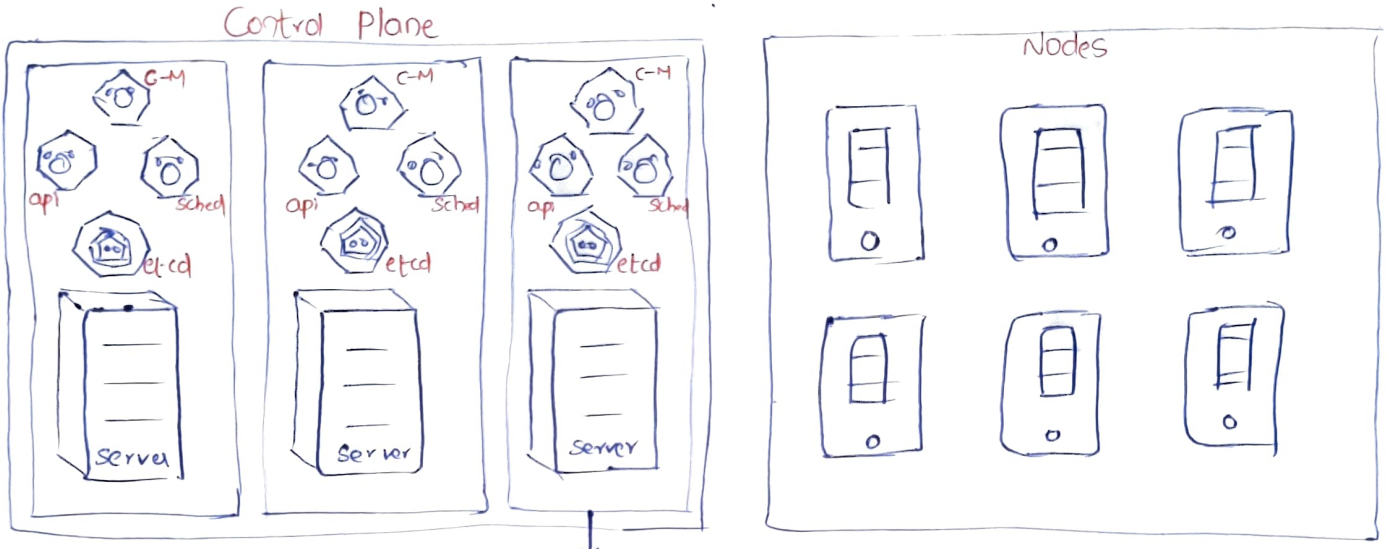
(Kubectl will convert it into http call and will communicate in json)

Kubectl → reads manifest (yaml) and connects to api server (KUBECONFIG comes into Play) → (Converts the manifests into REST API calls over JSON)

Ideal K8s HA-cluster :-

You can run K8s on one M/C itself, where that will act as a master and as a node also.

But since we are speaking about prod^n which is not a ideal case - we'll be speaking about multiple nodes.



we will have

Master Node (Since we have all components)

↓

If we have one Master Node

↓

It means single point of failure.

We can add as many nodes as we want. (On each node, we will have 3 components)

↓

- Kubelet
- Kube-proxy
- C-runtime

etcd was a key-value pair for a distributed system. (so whether you write etcd to any one of them → all of them will be in sync)

Why did I choose 3 master Nodes? why not two?

Even if there are two → It is highly available. so why did I use 3?

There is one study which says that whenever you want to go for HA choose odd no. of servers. (Google the reason) (why 3, 5, 7?)

(Ideally we'll be having load-balancer in front of API servers which will forward the request to any of the nodes.

In K8s, I have 2 kinds of servers

- Control Plane (Master nodes)
- Nodes.

* Kubernetes as a Service *

→ All popular clouds are offering K8s as a service.

- AKS (Azure K8s Service)
- EKS (Elastic " ")
- GKE (Google K8s Engine)

→ All cloud providers manage control plane for you and they charge hourly.
(There is no need for me to worry about control plane Mgt :→ cloud does it for us. 0-? $/hour.
The only thing I need to worry about is node mgt)

For nodes we pay the similar costs of virtual machines.

In prev. image → we are dealing with 3 + 6 servers = 9 servers.
(On-premise)

If it is K8s as a service → we will be managing only 6. (remaining 3 will be by cloud)

* K8s Installations :-

→ Single Node Installation → only one server

- minikube
- kind

→ On-prem Installations :

- Kube-admin

→ K8s as a Service :-

AKS
EKS
GKE

→ Playground (for learning)

* Installing K8s cluster on Ubuntu VMs :-

→ Create 3 ubuntu VMs which are accessible to each other.

(You can't create K8s on t2.micro — Free M/c doesn't work)

bcoz you need atleast 2 V. CPUs & 4 GB of RAM. (Try playground)

→ Installation Method (Kubeadm) which is something we will be using in on-premises K8s.

Ideally, when we are working on on-premises → our kubectl will be in master

But eventually we'll be removing kubectl from master and install it on our laptops. (Ideally it shud be present in your CI/CD pipeline's build server)

↓

i.e. Jenkins Node / Azure Devops Agent

Go to AWS Console

↓

Launch an Instance (3 servers)

↓

Ubuntu

↓

t2.medium / t2.large

(Try to ensure that within your network all the communications are open

(For now → open all TCP)

↓

Launch Instance

1) master
2) node 1
3) node 2

Google :- kubeadm Installation

↓

required ports

↓

These are all the ports that are req. → 5: All

Requirements :-

- Compatible Linux host : ubuntu is compatible ✓
- >2GB RAM ✓
- >2 CPUs ✓
- Full network connectivity ✓
- Unique host name for every node ✓

(For now All TCP)

Port	
6443	TCP - kubernetes API server

* Swap must be disabled. You MUST disable swap in order for the kubelet to work properly. (1)

For Eg:- sudo swapoff -a will disable swapping temporarily. To make this change persistent across reboots, make sure swap is disabled in config files like /etc/fstab, systemd.swap, depending how it was configured on your system.

→ Generally, in on-premises — on physical servers → people enable this swap memory — which will not allow k8s to work.

* Verify the MAC address and product_uuid are unique for every node.

(ip link) (or) (ifconfig -a) sudo cat /sys/class/dmi/id/product_uid.

For us we will not have this problem → bcoz every VM will have its own unique content.

→ Ensure that required ports are open.

* We need to install container runtime on all the machines.

1st thing is Install container runtime.

Kubernetes → click on container runtime.

You can run any of these container runtimes with Kubeadm:
- containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime.

You need to install a container runtime into each node in the cluster so that pods can run there.

K8s 1.27 requires that you use a runtime that conforms with CRI (Container Runtime Interface)

We will be using → container runtime → Docker Engine.

* So on each of your nodes install Docker
* Install Cri-dockerd

(Remember, K8s has stopped giving special privileges to Docker. This is the project which basically enables the Communication b/w Docker & K8s.

CRI → Container Runtime Interface.

For Cri-dockerd, the CRI socket is /run/cri-dockerd.sock by default.

* Now, what k8s tries to tell is

any ©r Techy → if you want to work with k8s → You need to implement CRI

Docker didn't implement it. Some open source has done it.

* Lets start by installing docker on all 3 M/c's.

Open 3 terminals
rename master, node1, node2.

There are certain commands which 9 need to run on all the nodes.

for Eg:- container runtime needs to be installed on master as well as all the nodes.

→ Login into all the M/c's.

Step 1 :-
master

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sh get-docker.sh
$ sudo usermod -aG docker ubuntu
$ exit
Relogin
$ docker info
```

node1: same step; Same; "; "

node2: same step; Same; "; "

Step 2 :- Install CRI-dockerd (cri-docker daemon) (Github)

The basic reason for this project to be present is → bcoz special treatment to D^r is finished.

You don't need to do this step → if you are installing k8s < 1.24.
bcoz k8s itself used to handle all of this.

You need to execute these commands ( ) Google: cri-dockerd → github/Mirantis

Run these commands as root user in all the nodes.

* First Install GO language

Since this runtime is developed in GO language.

$ sudo -i #cd ~

root:~ # wget https://storage.googleapis.com/golang/getgo/installer_linux

O/p :- 'installer_linux' saved.

root:~ # chmod +x ./installer_linux

./installer_linux

node 1	node 2
same steps	same steps

O/p: Welcome to the GO Installer!

Downloading GO version go1.20.3 to /root/.go

- - -

Setting up GOPATH

GOPATH has been set up!

One more thing! Run 'source /root/.bash_profile' to persist the new environment variables to your current session

(or) Open a new shell prompt.

source /root/.bash_profile

node 1	node 2
same step	same step

Next, we need cri-dockerd :-

git clone https://github.com/Mirantis/cri-dockerd.git

cd cri-dockerd

~/cri-dockerd/ # mkdir bin

So, we are taking the code → building the code → and then copying the result of the code into some executables. Then - docker will work.

~/cri-dockerd/ # go build -o bin/cri-dockerd

node 1	node 2
same step	same step

mkdir -p /usr/local/bin

~/cri-dockerd/ # install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd

/# cp -a packaging/systemd/* /etc/systemd/system

~/cri-dockerd/ # sed -i -e 's, /usr/bin/cri-dockerd, /usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

systemctl daemon-reload

systemctl enable cri-dockerd.service

systemctl enable --now cri-docker.socket

node 1	node 2
same step	same step

* If you want to run all of these commands at the same

If you have a Linux M/C → then you have something called as Tmux

Tmux : is a Linux Application that allows multi-tasking in a terminal window.

It stands for Terminal Multiplexing.

Pre-requisites :- A Linux based System.

A User A/C with sudo (or) root privileges

Access to a terminal window/command line.

command :-

sudo apt-get install tmux.

It doesn't work on Windows

It works on Linux / MAC system.

* So on all the 3 machines ; we enabled cri-docker.service.

Documentation :- Getting started → Production Environment → Container Runtimes.

* Now we need to install Kubeadm, kubelet and Kubectl

kubeadm : the command to bootstrap the cluster.

kubelet : the component that runs on all of the M/C's in your cluster and does things like starting pods & containers.

kubectl : the command line util to talk to your cluster

This also needs to be installed on all the 3 machines.

~/cri-dockerd/ # cd ~

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl

sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

node 1 | same steps

node 2 | same steps

* So on all the three machines we installed all components.

Configuring a cgroup driver :-

↓

Later

* Now create a cluster from a master Node.

Now, this kubeadm command

↓

From whereever I run this command → It will become master. (That M/c)

- So when you install k8s cluster, what it does is, it installs k8s cluster without network implementation.

In Docker, we said that there is something called CNI (Container Network Interface) where there are 3 things — Sandbox, Endpoint etc.

But in k8s, what happens is, this Container Networking is implemented by many companies.

After installing the cluster, you can decide which Networking Policy do yo want to implement.

So, initially, when I initiate the cluster → It wud not have networking by defa... Then, I need to do an extra step.

- Kubeadm is the command to create the cluster.

Bootstrapping :- It is the process of creating a new k8s cluster from scratch and getting it up and running.

- Bootstrapping a k8s cluster involves setting up the control plane and worker nodes and determining which node has the correct info with which all the other nodes shud synchronize

kubeadm --help

o/p: init → to set up the K8S control plane.

↓

This M/c will be come master

kubeadm init --help

You can pass multiple arguments

--pod-network-cidr string (specify range of IP addresses for the Pod network. If set, the control plane will automatically allocate CIDRs for every node)

- If K8S, you wud be implementing network separately.
- So you can try to tell what is the ip-range which you want using this network policy.

Google : K8S flannel → github

Flannel is a simple, lightweight layer 3 fabric for K8S.

Flannel manages an IPv4 network b/w multiple nodes in a cluster.

It does not control how containers are networked to the host, only how the traffic is transported b/w hosts.

kubeadm init --pod-network-cidr "10.244.0.0/16"

o/p: Found multiple CRI endpoints on the host. Please define which one do you wish to use by setting the 'criSocket' field in the kubeadm configuration file:

unix:///var/run/containerd/containerd.sock, unix:///var/run/cri-dockerd.sock.

↓

This fools trying to tell that → you have 2 things { containerd socket, cri-dockerd socket. ↓ we installed it. } { One which comes with Docker Installation ↓ not usable for us so we went with cri-dockerd }

kubeadm init --help

(The socket which we shud be using is: cri-dockerd.)

o/p: --cri-socket string.

kubeadm init --pod-network-cidr "10.244.0.0/16" --cri-socket "unix:///var/run/cri-dockerd.sock"

This we shud do as extra step. After K8S 1.24 → This is what has happened.

* So when you execute this command - K8s cluster

if you remember → we need to configure Kubectl

and Kubectl requires a configuration file.

↓

This is all about - setting that up.

Right now, we have kubectl in the same node where we have master.

* To start using your cluster, you need to run the foll as a regular user :-

Master

exit

* Setting up kubeconfig :-

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

$ kubectl get nodes

o/p:-

Name	status	Roles	Age	Version
ip-172-31-7154	NotReady	control-plane	93s	v1.27.1

It is in NotReady state. → Reason for this is

Network policy is not implemented.

You need to install add-ons.

Google : K8S Add-ons.

↓

I want to add Flannel (click on it → github.com → install steps)

↓

Reason:- In DY we spoke about Underlay and overlay networking.

Flannel also uses similar concept.

Install Flannel :-

Master:

~ $ kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml

o/p : namespace/kube-flannel created.

serviceaccount/flannel created

configmap/kube-flannel-cfg created

daemonset.apps/kube-flannel-ds created.

$ kubectl get nodes -w → stands for watch. It will watch for any changes that exists.

O/p:- NAME STATUS ROLES VERSION AGE

ip-172-31-7-154 Ready control-plane 1.27.1 - -

Now, node is ready.

Now, we have to add node1 & node2 to the cluster.

↓

we have a command for this. But to this command we need to add one thing extra.

You can join any no. of worker nodes by running the following on each as root

```
kubeadm join 172.31.7.154:6443 --token 3cralf.f4y8zpzdj51xxmq2 \
        --cri-socket "unix:///var/run/cri-dockerd.sock" \
        --discovery-token-ca-cert-hash sha256:e3b.... - - -
```

That one thing extra is → Remember everytime we have to tell what is the cri-socket which we want to use.

node 1

root@.. # kubeadm join 172 - - - - - -

node 2

root@ # kubeadm join - -

Master :-

$ kubectl get nodes

O/p: NAME STATUS ROLES VERSION AGE

ip:172-31-7-154 Ready control-plane v1.27.1

ip-172-31-13-35 Ready <none> "

ip-172-31-8-106 Ready <none> "

* Now we have k8s cluster.

* This is a time-taking Installation. The problem why we are seeing this installation is:

There 2 things are Extra :-

(1) Before 1.24, there was special treatment for docker.

You just install docker, install kubeadm ⟶ Everything used to work.

But now,

that is not the case.

we had to install cri-dockerd

and everytime whenever we are executing `kubeadm command' → we have to give cri-socket

(2) K8s by default doesn't come with default network Policy.

If you remember → when you install Docker ⟶ you get bridge.

Similarly when you install K8s ⟶ you don't get any network. So I have to choose a network policy or you have to choose a network driver.

So, we have chosen → flannel. There are many other CNIs.

↓

(The reason we chose it is
An underlay and overlay network gets created)

Flannel expects this ⟶ --pod-network-cidr "10.244.0.0/16"

other CNIs do not require this.

* Bootstrap a Kubernetes Cluster :-

- Bootstrapping a Kubernetes cluster involves setting up the control plane and worker nodes and determining which node has the correct information with which all the other nodes should synchronize.
- You can do this manually, programmatically (or) with Kubeadm, a K8s tool to setup and manage clusters.

* Kubeadm v/s Kubectl :-

Kubeadm	Kubectl, Kubelet
* An administration tool that operates at the cluster level. * You'll use this to create your cluster and add additional nodes.	• Kubectl is the CLI you use to interact with your K8s cluster once it's running. • Kubelet: This is the K8s process that runs on your cluster's worker nodes.

* Minikube v/s Kubeadm

Minikube	Kubeadm
• Minikube is a fast and simple solution for deploying a single node cluster.	* Kubeadm builds a minimum viable, production-ready K8s cluster that conforms to best practices.

Lec-52- Apr-23 Kubernetes - Objects, Workloads, Resources, Manifest File, Pods

* API Manager is responsible for all the communications in K8S.
  It exposes its functionality over http (which is REST API).

* Lets look at fake rest api :-

Rest API is anything that can be communicated over http.

So, I'm sending a curl request.

Curl → is where you can send web request.

Eg:-

```
$ curl https://jsonplaceholder.typicode.com/todos/1
```

Op:-

```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

The response which you are getting is in json format

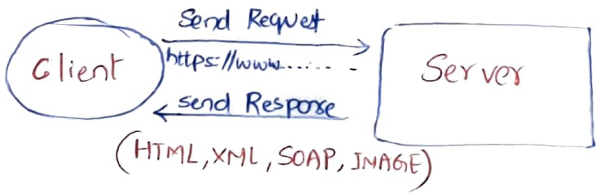
Fake REST API :- is a free online REST API that you can use whenever you need some fake data. It can be in a README on Github, in code examples on stack overflow... or simply to test things locally.

↑

- Today there are 3 categories of API protocols/architectures
  - REST
  - SOAP
  - RPC

Create ← POST
retrieve(or) read ← GET
Update ← PUT
modify ← PATCH
delete ← DELETE

REST API :-

- It is an Interface that two computer systems use to exchange information securely over the Internet.
- A REST API is a way for two computer systems to communicate using the HTTP technologies found in web browsers and servers.
  Sharing data b/w 2 (or) more systems has always been a fundamental requirement of software devt.
- It is used to fetch (or) give some information from a web service.
- All communication done via REST API uses only HTTP request.
- 

- A request is sent from client in the form of a web URL as HTTP GET or POST or PUT (or) DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image (or) JSON. But now JSON is the most popular format being used in web services.
- In HTTP, there are 5 methods that are commonly used in a REST-based Architecture.

* JSON is also like YAML - which is a key-value pair.

* So K8s API server also understands this kind of communication.

curl https://jsonplaceholder.typicode.com/todos/1

todos is → basically what is it that you are supposed to do.

Now, lets remove 1 & see the result.

we will get multiple todos.

Generally, what happens in http requests is (or) in REST API's is :-

You will have a HTTP URL where you will have some functionality around some resource.

For Eg :- In an eCommerce system, catalogue shows all the products.

Now, what this HTTP URL (or) REST API tries to tell is → If you want to get the info send a GET request.

If you want to create an object → use the POST request.

If " " " update → use the PUT request

If " " " delete → " " DELETE request.

All of these things are already available in HTTP.

Request Methods
GET
PUT
POST
DELETE

* In K8S also, there is some REST API

Google :- kubernetes rest api.

↓

whenever you want to speak with K8S, ideally you shud be using K8S apis.

For this what K8S says is → I have client libraries.

(They have diff-t language based apis)

The whole idea is K8S can be interacted by anyone who can send a HTTP request.

* Kubernetes Objects :-

- Everything in K8s is an object

So, whatever we are going to create → let it be something to do with Containers,
" " " network
" " " load balancer

Anything which you create in K8s is basically an object.

- When we are speaking about these objects → what it tries to tell is →

Every object has 2 important things
- Spec → Specification (what is it that you have asked)
- Status → what was created

* Kubernetes API - Resources :-

* The resources exposed by api-server are K8s api-resources.

So whatever api-resources you get → you can act on them.

* Lets use Instances created last time (1 master, 2 Nodes)

Currently kubectl is in master node

Login into the M/c's.

Master :-

$ kubectl api-resources → Many things which we do in K8s will be around this.

whatever is described in this → These are the resources which we wud be working on.

* From version to version → there might be some resources added, some modified - - -

Right now — we are working on K8S-1.27. (In 1.28 you might have 2 or 3 resources added)

But the standard resources don't change.

* Whenever we are working with K8s → we are working with these resources.

Whole of your journey is dealing with those api-resources.

* Kubernetes Workloads :-

Google: K8s Workload Resources

⇒ whenever we speak about K8s workloads

> K8s creates a smallest unit which is called as Pod

The 1st & primary workload which we are going to discuss about is Pod.

* So, K8s creates Pods and not Containers.

whenever I'm speaking about Hypervisor

↓

what is the smallest unit which the Hypervisor can create

= virtual M/c.

Dr can create a smallest unit which is = container.

• Pods :-

Pod — contains Container — Specification | Container

→ primitive of K8s.

→ The smallest unit of creation in K8s is Pod.

→ Pod has container(s). (Your Dr © will be inside Pod)

→ Every Pod gets an IP address (not every container)

* Now, around this Pod → there are some other workloads. There is something called as ReplicaSet

• ReplicaSet :-

It speaks about → How many Pods?

when you run the pod → the thing which you are describing in Pod is — Specification (in Pod) → will be around - what is the container which I want to run.

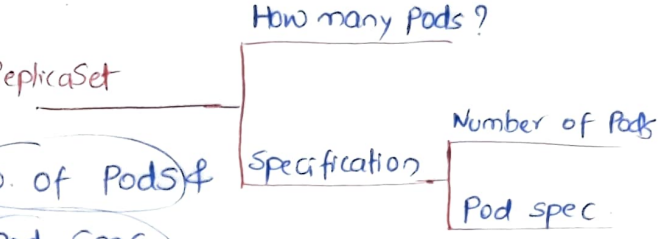
- All the desired state - whenever you are trying to write Pod is → to describe a container.
- The problem here is → Pod tries to create container.

But the point is → If the Pod dies, If anyone messes with Pod → those things are not your desired state → Desired state is all about Container.

So K8s will try to just run ©r — whereas ©r is not a primitive unit in K8s.

* ReplicaSet :-

- It speaks about — How many Pods ?
- In this, specification speaks about (No. of Pods) & (Pod Spec)



Eg:-
So, here I will tell that → I want 3 MySQL Pods (not containers)

↓ &

K8S has to maintain that state.

In Pod → there is no such state.

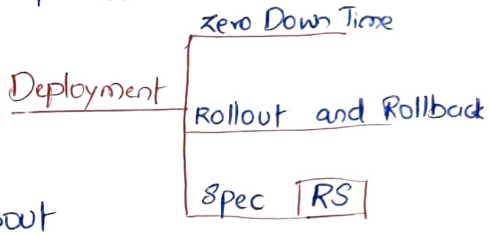
* Deployment :-

→ It is all about Zero Down time Deployment of application.

→ It speaks about Rollout and Rollback

→ Here spec speaks about Replicasets

(In the Deployment spec — you try to speak about Replicasets)



So, we will not be directly running Containers. We will be creating Pods. Inside the Pod, you will have One (or) more Containers. →

→ Replica set is all about running multiple pods → Deployment is all about making your application deploy → & then if it is not working — going back to the previous version → so that means it supports Rollouts & Roll backs.

* we also have Statefulsets

* Statefulsets :-

- These deal with state.
- Remember (C)r has one serious problem → (C)r when deleted → all the data in the Read-write layer is gone → So for that — in the case of Docker we create Volumes.

If you have some kind of state application — where you have to scale → so then you will be going with statefulsets.

* Jobs :- Job means -

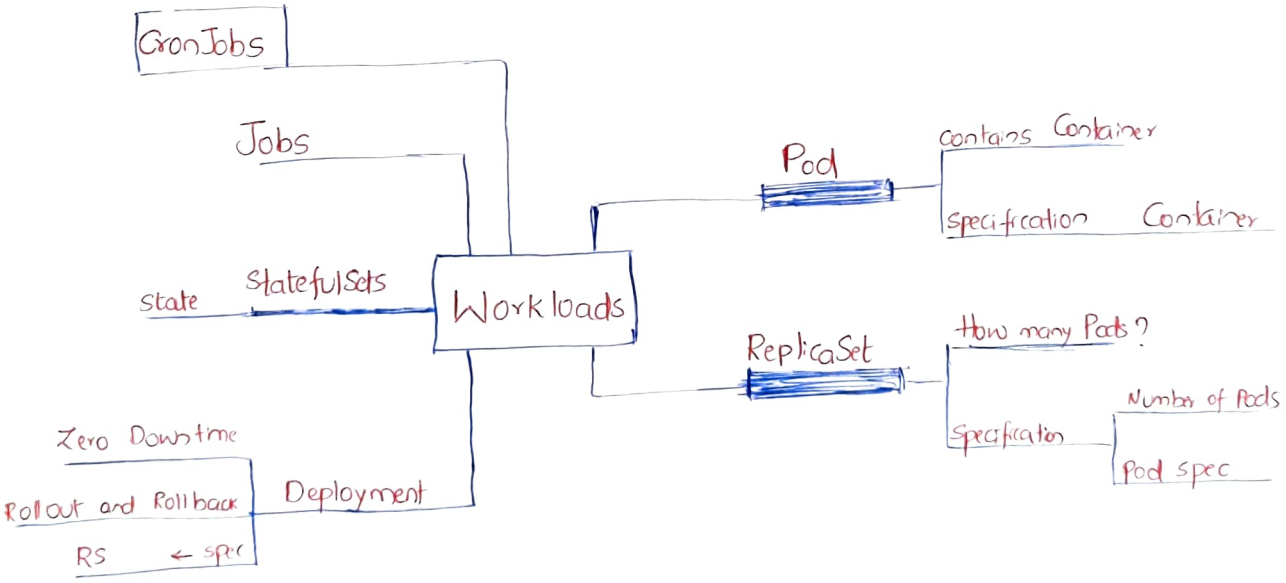
Generally whenever I'm trying to run MySQL — what is my intention ? → MySQL shud be running forever. But let us assume that I want to run something inside a pod which will run for 2 hours and after 2 hrs it will complete — Everyday at night 12 'o' clock this work shud start which will end at 4 'o' clock → Here I'm trying to speak about something which will finish in 1 (or) 2 hrs.

So, Pods do not speak about something which is finishing. It shud always be runing. That is where Jobs come into play. (Jobs are something which you run onetime)

* CronJobs :- CronJob is scheduling those Jobs. (Cronjob is something which you do repeatedly)

* So, in K8s, we are not going to directly deal with (C)'s. we will deal with Pods.

Anything and Everything which we create after that will be revolving around these pods.



$ kubectl api-resources

o/p:

Pods	po	v1
deployments		
statefulsets		
replicasets		
daemonsets		

Everything in k8s is an object and you communicate to that object using api-resources.

(Anything & Everything in k8s is basically around api-resources)

* Ideally k8s can be spoken over Rest-API (or) else kubectl [we will use kubectl]

* Now, in k8s, we are going to create objects. So objects are exposed as api-resources.

To deal with these objects — we have to write manifest files. (which is a yaml file).

* How to create Resources in k8s :-

- We wud be creating k8s manifests ie. yaml files.
- For this — we need to understand
  - → Yaml
  - → api versioning (In k8s, there are lot of api versions)
  - → spec and status (whenever you try to create anything in k8s → you try to write spec → k8s will give you back status)

→ API Versioning :-

Google: k8s api versioning. (API overview)

* So k8s is just like any other software where you get new features added, you get many other things.

So what k8s tries to tell is → whenever they give a new feature you wud have these channels — { Alpha, Beta & Stable

- Let us assume that k8s wants to start all together a new feature, it wud not directly be a stable version.
  It wud be in Alpha. There will be some of the changes which are in Beta stage.
  ↓
  and then it will become stable.

- So, what is the difference ?

→ Alpha means Community (Cloud Native k8s Foundation) has accepted that change — but there might be lot of changes. There might be changes in the api also.
So, for Eg, if k8s want to create something like api-gateway — a new k8s resource → so it is in Alpha stage
It means that they basically created this feature and there are going to be lot of changes inside that → & Don't use it in production.

The basic point is, when the feature is in Alpha, it might be dropped also.

→ The moment, they go to the Beta, k8s is not going to remove that feature. But there might be some minor changes.

→ Once you go into the stable version → that means that → there will be support for some time.
With every standard k8s release, they will have certain support. (atleast 2 or 3 years)

- What k8s tries to do is
  Since we speak with apis, they also have this concept of apigroups

- APIs are grouped as apigroups. (Google it)

API groups: They basically speak about, whatever resource which you are trying to accept or use — which group does it belong to.

*Some of the API groups are :- core
batch
networking.k8s.io

(In groups, you wud have multiple resources)

* In groups, you wud have multiple resources and we write specs for those resources.

* API version: It is written as <groupname>/<version>

If the group name is core → then you can directly write <version>
core is minimum functionality of K8S. (Default group is : core)
So whenever you are dealing with core objects, there is no need for you to tell the group. Bcoz it assumes that the default group is core.
(Just like tag in Docker: if you dont tell tag → it takes 'latest' as default tag)

* In this groups, we have kind of api-resources. $ kubectl api-resources

For Eg: pods belong to API version → v1 ⇒ means → core/v1

Eg: customresourcedefinition → belong to → apiextensions.k8s.io/v1
groupname / version.

So, within group, you wud have multiple resources.

• Defining Resources in a Manifest File :—

To define a resource in a manifest file we create a yaml file with following structure :-

```
---  Yaml
apiversion:
kind:
metadata:
spec:
---
.
```

So whenever you try to create anything in K8S, these are the 4 major sections which you wud try to write.

1st one → apiversion (to write this → we need to know what is the group that it belongs to and what is the version)

2nd → kind. ↓ is what you are trying to create.

3rd → metadata → where we try to give some info about names of resources
Eg: I want to create MySQL Pod (name: mypod).

4th → spec → is your desired state.

* Pods :-

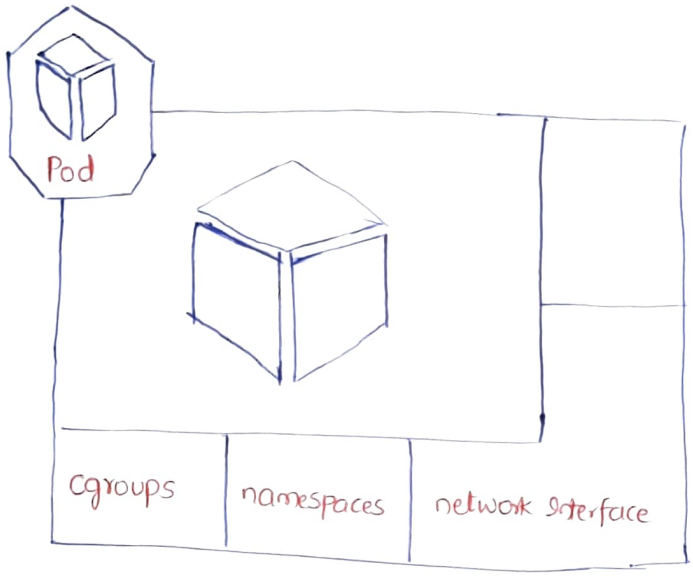
Google : k8s Pod yaml → Pods (see yaml files)

Google : k8s deployment yaml → Deployments (" ")

Anything & Everything which you try to create will have 4 sections

- apiVersion :
- kind :
- metadata :
- spec :

- Pod is atomic unit of creation in k8s cluster.
- Pod contains container(s).



Docker uses namespaces of various kinds to provide the isolation that C's need in order to remain portable and refrain from affecting the remainder of the host system. Each aspect of a C runs in a separate namespace and its access is limited to that namespace.

- Inside k8s pod, you wud have container.

To create Docker Container — We need namespaces & cgroups.

- Containers inside the Pod will share
  - cgroups
  - namespaces
  - network interface.

- Cgroups manage resources, and namespaces isolate and secure them
- Cgroups play a role in containerization solutions like D & k8s, where they control C resource allocation and ensure isolation from the host & other C's.

In the pod, ideally it is always a good idea to have only one container.

- Pod can have multiple containers

On the node → we have
- KubeProxy = networking
- Kubelet = agent
- C-runtime = creates C's

- Each Pod gets a unique IP address → KubeProxy is the component that gives this.

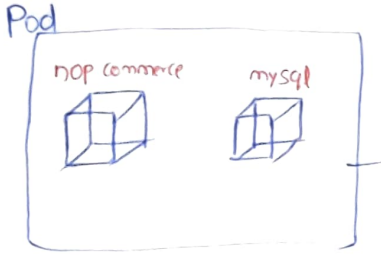
These networking capabilities to the D C are basically given by KubeProxy.

(a) IP address

* Now, the problem with running two containers inside a Pod is

① both containers will be running on same IP.

That means that → if one ©ʳ wants to speak with other one inside a Pod they have to use local host (loop back).



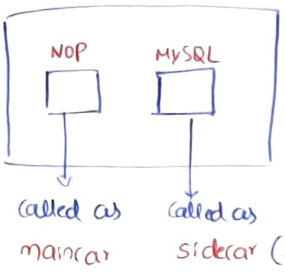
→ This is a bad idea.

Bcoz if ©ʳ is the smallest unit of creation then we wud have done this.

What we shud be doing is

we shud create nop-Pod and a mysql-Pod.

* when you have more than one ©ʳ



When shud we use this? or why do I use this?

For Eg: Let us assume - you are running NOP commerce. and all the logs that nop-com generates - I want to store it in Azure.

So my sidecar will be an agent which reads logs from nop ©ʳ and stores it in Azure (or) Elastic Search.

So, you wud have it do those kind of things as sidecar (But not major functionality)

* Scaling in K8S is definitely not this

Scaling in K8S is increasing number of pods and not containers.

* Pod gets assigned to the node.

(So whereva you try to tell that → K8S → I have written a manifest file for Pod

↓ we write yaml file

Execute kubectl command

↓

kubectl will [illegible], speaks with api-server

↓

api-server will store api-resource info in etcd cluster.

↓

The moment a new resource request is stored in ↓ Scheduler will try to create a new pod on whatever node it finds interesting.

* Google:- Kubernetes Pod yaml.

↓

Pods

↓

copy yaml file

workflow

Lets try to write Pod manifest for httpd.

```
---
apiVersion: v1
kind: Pod
metadata:
    name: httpd
spec:
    containers:
    - name: httpd
      image: httpd      → will take tag as latest (It is a Docker Image)
      ports:
      - containerPort: 80
---
```

* Lets try to write a manifest for Jenkins :-

```
---
apiVersion: v1
kind: Pod
metadata:
    name: jenkins
spec:
    containers:
    - name: jenkins
      image: jenkins/jenkins
      ports:
      - containerPort: 8080
---
```

(Never use this approach.
Don't look at some Example and try to write yaml file.)

* Google : Kubernetes Pods
* Create a new folders :→ k8s → Manifests → Pods (Open with code)

  new file : httpd.yaml

  Google : Kubernetes API reference

Right Way of writing a Manifest file

* I want to run httpd

httpd.yaml (vscode)

---
① apiVersion:
② kind:
③ metadata:
④ spec:
---

* Whenever we are writing manifest file → we will have 4 things

This is something which you are going to write for every file.

Google : Kubernetes API reference
↓
API reference

For now, we will be using K8s 1.27 (if you want 1.24 → change it in url)

→ We want to create a pod (httpd pod)

API versioning
↓
It is combination of 2 things / api group
\ version.

Docker Hub
↓
search : httpd.
(latest version is 2)

Right now, my intention is to create a pod.

Pod will have containers.

I want to create httpd-pod.

* So, in API reference doc? → we see Pod v1 Core → click on it.

If it is core, there is no need for me to write → core/v1.

I can directly write v1.

↓
we see group's name as core
" " version : v1
Kind : Pod.

(Since it is a yaml file → after ':' give 1 space)

In Docs :—

→ apiVersion → type is 'string'. (Since it is a string → there is no need for quotes)

→ Kind → type = string → for us Kind = Pod. (It is case-sensitive)
P → capital.

K8s is Case-sensitive

→ metadata → type = ObjectMeta. (In this you can write many things)

- Whenever it is of type : objectMeta (give 2 spaces in next line)

↓

Eg: metadata:

name: httpd-pod

(In docs → click on objectMeta)

↓

Version: v1

Group: meta

kind : ObjectMeta.

(we can see in docs what all we can write in objectMeta.)

In Docs

* Spec → type : PodSpec (it is not basic type)

" " list

since it is an object

↓

Go to next line & give 2 spaces

In Docs

(Open PodSpec)

↓

read all fields.

In the list → we can see Containers (it is an array)

Pod is collection of containers.

we can see in docs → containers. :- type:- Container Array

↓

means multiple

(Individual item is of type container)

↓

click on Container.

↓

There is a list

* Image (string)

So whenever we are speaking about multiple

↓

give 2 spaces and a hyphen '-'

whenever we are trying to write about Containers,

it will have image. (type: string)

it will have name (type: string)

it will have ports (type: ContainerPort array)

↓

Since it is an array ⇒ 2 spaces & '-' i.e hyphen.

click on ContainerPort.

↓

list of integers

containerPort → integer.

You can also give a name.

You can give protocol

- Manifest File — httpd.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: httpd-pod
spec:
  containers:
    - name: httpd-cont
      image: httpd:2
      ports:
        - containerPort: 80
```

TYPE
Array
Containers → multiple
ports → Array multiple

- Manifest File — alpine.yaml (Inside Pod I want alpine)

Docker Hub (alpine)

The problem with alpine is,

when you run the alpine ☹ → it will immediately be in exited state.

I want to try to write alpine → so that it runs for 1 day.

We can use sleep command. (you shud not be using it — but that's okay)

Here I don't have ports.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: alpine        → Pod's name
spec:                   (can have same name.)
  containers:
    - name: alpine    → Container's name
      image: alpine
      args:
        - sleep
        - 1d
```

Pod
↓
apiVersion: core & v1
Kind: Pod
metadata → object type
↓
next line (2 spaces)
spec
↓
type: PodSpec
Container Section
↓ type ContainerArray
click on Container
↓ next line & - /
→ args (string array)
→ command (string array)
→ name (string)

* We have to work with new CLI tool called as kubectl

Google: kubectl cheatsheet.

When you are working on a Linux M/c → you can go with autocomplete (tab)

Master:

$ source <(kubectl completion bash) => for Kubectl Autocomplete

Docs

* Kubectl → how do you Create objects

(will give us K8s related help.)

The Pod specification which you have written

↓ is all about creating an Object;

we have written yaml files.

So, we need to do only simple command

↓

kubectl apply

You can Add Autocomplete permanently to your bash shell

⇓

echo "source <(kubectl completion bash)" >> ~/.bashrc

master (Sir is doing it)

$ git clone https://github.com/asquarezone/kubernetesZone.git

$ cd KubanetesZone/April23/Manifests/pods/

$ ls

o/p: alpine.yaml httpd.yaml.

$ kubectl apply -f httpd.yaml

o/p: pod/httpd-pod created.

$ kubectl get pods

o/p: Name Ready Status

httpd-pod 1/1 Running

You can get any api-resource ↓ using (kubectl get). Here we created Pods ↓ lets get pods.

* Basic commands which we wud be using ctcl-

→ kubectl apply -f

→ kubectl get <api-resource>

→ kubectl describe <kind> <name>

* we will be using 'apply'. In 'apply' you can pass the manifest file.

TO pass the manifest file → you will be using -f and give the path of the file. kubectl apply -f

* We will be using 'get' : kubectl get <api-resource>
* We will be using 'describe' : kubectl describe <kind> <name>
* whenever you are working with K8S → you can add -w

-w → is watch when things change.

$ kubectl api-resources

Lets focus only on pods.

$ kubectl api-resources | grep pod

	Name	shortform	version
o/p:	pods	po	v1

podtemplates

$ kubectl get po (or) kubectl get pods

will give you the same result.

o/p: httpd-pod 1/1 Running

* If you want to get more information → you can just try to write -o wide

$ kubectl get pods -o wide

o/p:-	Name	Ready	Status	Restarts	Age	IP	NODE	NOMINATED NODE	READINESS GATES
	httpd-pod	1/1	Running	0	3m22s	10.244.1.2	ip-172-31-8-106	<None>	<none>

IP address of Pod

On which node it got assigned.

Go to AWS.

We see the ip of nodes.

We can see that node 1's ip is 172.31.8.106.

So, our httpd-pod is running on node 1.

$ kubectl describe pods httpd-pod

o/p: we are getting some information } detailed info about Pods.

Events :-

Type	Reason	From	Message
Normal	Scheduled	default-scheduler	Successfully assigned default/http-pod to ip-172-3-1-8-106.
Normal	pulling	kubelet	Pulling image "httpd:2"
Normal	Pulled	kubelet	Successfully pulled image "httpd:2"
Normal	created	kubelet	created container httpd-cont
Normal	started	kubelet	started container httpd-cont.

$ kubectl get pods -o yaml

o/p:- apiVersion: v1
kind -
metadata
spec - -
Containers -
volume.

we didn't write this much of yaml.
But internally, all of it got piled up.
And at the end
we can see status.

To view the complete manifest created by kubernetes

⇓

kubectl get <kind> <name> -o yaml

If you want to remove the volume (or) remove the pod

⇓

$ kubectl delete pods httpd-pod (or) $ kubectl delete -f httpd-yaml.

o/p: pod "httpd-pod" deleted.

$ kubectl get po

o/p: No resources found in default namespace.

mavtea

$ kubectl apply -f alpine.yaml

o/p: pod/alpine created.

$ kubectl get po

o/p:- alpine 1/1 Running

$ kubectl get po -o wide

o/p:			IP	Node
	alpine	1/1	10.244.2.2	ip-172-31-13-35

$ kubectl get po alpine -o yaml ↳ node 2.

o/p:

$ kubectl describe pods alpine

$ kubectl apply -f httpd.yaml

o/p: Pod/httpd-pod created.

$ kubectl get po -o wide

o/p:-				IP	
	alpine	1/1	Running	10.244.2.2	ip-172-31-13-35
	httpd-pod	1/1	Running	10.244.1.3	ip-172-31-8-106

* Each Pod is getting its own IP-address.

In this case, one pod is running on node 1 and other pod on node 2.

So, Scheduler did this. ↓ (we can change this. for now leave it)

$ kubectl delete pods httpd-pod | $kubectl delete -f alpine.yaml

Exercises :-

- Write a manifest file to create
  - * nginx
  - * nginx and alpine with sleep 1d
  - * nginx, alpine with sleep 1d and alpine with 10s
  - * nginx and httpd with 80 port exposed.

- As of now, we are using kubectl which is present in the Master Node

Pods (contd) :-

→ There are 2 ways of running Containers
- Declarative (is always the way to run C's)
- Imperative.

Master

$ kubectl get po

o/p: No resources found in default namespace.

$ kubectl api-resources

(Everything in K8s which you can communicate is exposed as an api-resource

↓

And each api-resource is basically an Object which gets stored in etcd

↓

Generally you wud have →
- Create an Object
- Update an object
- Delete an object
- Retrieve an object.

* For creating an object, generally, we go with apply command. [kubectl apply ....] (by writing the manifest file)

Updating also we use the same thing.

* To delete → we use 'kubectl delete'
* To retrieve an object → .. 'kubectl get'
* If you want to know more info → we will use → 'describe'
* '-w' → is for watching for the updates.
* Now, lets try running K8s → in Imperative way. (Not a good way). objects.

Google : K8s Imperative Commands

Imperative way to manage k8s objects :-

→ Everything in k8s is an object. (but not preferred)

Refer docs. (not a great way)

(just learn it)

Ex

$ kubectl run nginx-pod --image nginx

o/p: pod/nginx-pod created.

$ kubectl get po

o/p: nginx-pod 1/1 Running

$ kubectl get pods nginx-pod -o yaml

Pod Lifecycle :-

Google: k8s Pod Lifecycle.

↓

Pod phase

⇨ Kubernetes Pods will have following states

* Pending → means you have tried to tell k8s – apply this pod

↓

k8s has accepted your request

↓

To create the pod, to assign to the resource – it wud take some time → till it goes into whatever the next state is.

Once your pod has been accepted by k8s cluster → it gets into the pending state.

* Running → means → it is after Pending state.

It means – it is running without any problems.

* Succeeded → means → whatever ©ˢ which are present in the Pod, they have already finished their execution.

So in the Pod, you have asked ©ʳ to run something, → it has successfully finished it.

* Failed

* Unknown

Failed state: Failed means one container in the Pod got terminated. That is, the ©ʳ either exited with non-zero status (or was terminated by the system

(Generally, we don't see this, but)

Unknown state: means k8s cluster is unable to determine what is the state of your Pod for whatever reason.

* For the container, that is running inside the Pod, you have different states

- **Waiting:-** If the © is not in either the Running (or) terminated state, it is waiting. A ©ʳ in the waiting state is still running the operations it requires in order to complete start up: for eg, pulling the ©ʳ image from a ©ʳ image registry.
- **Running**
- **Terminated:-** Your ©ʳ got into the exited state.

* In Pods, we can specify Container restart policy. (after they go into exited state we can restart them) ©ʳˢ

* **Container Restart Policy :-**

Google kubernetes api reference

↓

Pod v1 core

↓

PodSpec → restartPolicy (string) — Always (default)
— OnFailure (only when your ©ʳ exited & returns non-zero)
— Never

Lets try to create a short-lived ©ʳ with diff restart Policies.

* Create a new folder

↓

restarts

↓

write files inside the folder.

① My Pod's CMD wud run for 3 sec & then it wud exit.

Once this happens, your ① will go into stopped.

Here I have not written any restart policy. If I don't write any restart policy → it's default value will be → always → That means it will continue re starting :)

restartalways.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: restartalways
spec:
  restartPolicy: Always
  containers:
    - name: alpine
      image: alpine
      args:
        - sleep
        - 3s
```

restartnever.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: restartnever
spec:
  restartPolicy: Never
  containers:
    - name: alpine
      image: alpine
      args:
        - sleep
        - 3s
```

restartonfailure.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: restartonfailure
spec:
  restartPolicy: OnFailure
  containers:
    - name: alpine-failure
      image: alpine
      args:
        - sleep
        - 3SM
---
    - name: success
      image: alpine
      args:
        - sleep
        - 3s
```

This command will give Error code

Lets write 2 Pods in same Yaml file

2 → 1) restartonfailure
2) restartonfailure2

Copy & paste → name: restartonfailure2

Manual

$ sleep 1s

$ echo $?

Google : shell cheatsheet

$? → Exit status of last task

o/p: 0 → is a success code.

In linux, whenever any command returns zero '0' → that means it is successful.

Here, we asked it to sleep for 1 sec → It has done so successfully.

But, if I try to run

$ sleep 1SM ——→ there is nothing called as SM

o/p :- Error.

$ echo $?

o/p - 1 ——→ This is anything other than zero(0). → That is a failure state.

* We wrote 3 yaml files — 1) always (Default)
2) never
3) onFailure

$ kubectl get po

$ kubectl delete pods nginx-pod

2 pods in 1 yaml file
- Exit code => success
- Exit code => failure.

Now, we don't have any pods which are running.

(1) $ kubectl apply -f restartalways.yaml

O/p: Pod/restartalways Created.

$ kubectl get pods -w

			status
O/p:	restartalways	0/1	completed
	restartalways	1/1	Running
(ctrl+C)	"	0/1	completed
	"	1/1	Running (4 sec)

But K8s will not allow it to be in completed state.

It will restart the ©r.

Your pod has finished execution with a success code.

(we wrote sleep 3s)

↓

will give you zero (0) as a return code.

↓

(Success. => Completed).

How many times it will restart?

↓

we will speak later.

$ kubectl delete pods restartalways

(2) $kubectl apply -f restartnever.yaml

O/p: Pod/restartnever created.

$ kubectl get pods -w

O/p	restartnever	1/1	Running
	restartnever	0/1	Completed
	restartnever	0/1	Completed
	restartnever	0/1	Completed

So, whether it is success or failure, what K8s is trying to tell is → I'm not going to restart your ©r Now, it will not restart

So, after 3 seconds, K8s has gone into completed state.

$ kubectl delete pods restartnever.

③ $ kubectl apply -f restartonfailure.yaml

O/p: pod/restartfailure created

pod/restartfailure2 created

$ kubectl get pods -w

			Status	Restarts
O/p:	restartfailure	0/1	Error	1
	restartfailure2	1/1	Running	0
	restartfailure	0/1	CrashLoopBackOff	1
	restartfailure2	0/1	Completed	
	restartfailure2	0/1	Completed	
	restartfailure	0/1	Error	2

restartfailure2
↓
is a Success code.
It returns a success code.

restartfailure
↓
It returns Error code.

Only restartfailure pod is getting restarted. (restartfailure2 is not restarting as it has given a success code)

* So, I can set what my restart policy could be.

when I want to restart (or) don't want to restart my (c)^rs.

Default behaviour is: K8S will always try to restart your pods.

Controller, Jobs, CronJobs

Generally,

* k8s is trying to Pods. These Pods basically have containers which are designed to run all the time bcoz they run your applications.

* Sometimes, I might want to run some component of my appn which will run a script (or) which will basically run an application → which will try to take its own time and then it will finish. It need not run all the time.

Eg: Everyday night at 12'o' clock, I want to check all my database records. This is not something that runs all the time. It runs the script. The script will correct the records. It will be finished.

* Those kind of things are basically called as Jobs

Google: Kubernetes Jobs object.

↓

Jobs.

Controllers in Kubernetes :-

→ Controllers are k8s objects which run other k8s resources.

So I might write a Controller which speaks about running 3 pods.

Now, if you try to look at Pod, → Pod is a workload

So the basic idea of Pod is to run your appn inside the ©r.

But,

the basic idea of Controller is to run some other k8s resource.

This k8s resource will be part of specification – generally in template section.

For Eg: I'm trying to write k8s controller. (name: x)

The basic idea of x is to run 2 pods in 2 diff-t nodes.

while I'm specifying x, in the specification of x, I'll be trying to tell about which pods I want to run. So this specification will contain something called as a template.

- So this Specification will contain something called as template which is Pod template.

(If you remember, in the Pod → we were writing container template)

Now, we are speaking about Controllers, which basically try to run Pods, which is smallest unit of K8S.

- So, Controllers maintain desired state.

And the desired state is basically to run whatever pods which you are basically trying to speak about.

- Some of the Controllers are :-

  * Replication Controller / Replica Set
  * Deployments
  * Stateful sets
  * Jobs
  * CronJobs
  * Daemonsets

These are the kind of Controllers which we use. And in these Controllers, there will be other K8S resources which will try to maintain desired state.

* In all of the Controllers which we have written, the desired state will be for a Pod.

* K8S Jobs :-

The basic idea of this Job, here we are wishing for Pod, to run the Job and finish

For Eg: I want to download the file & then run that file.

(takes 5 sec) (takes 10 sec.)

So this Job will be alive for 15 seconds and after 15 sec, it will go into completed state and then we are not worried about restarting it → bcoz it is a Job & Job is supposed to be finished

- k8s has 2 types of Jobs :-

1) Job (object) ⟶ Run an activity/script to completion.
2) CronJob (object) ⟶ Run an activity/script to completion at specific time period (or) intervals.

Eg:- (I want to run some Job for every 1 hr (or) 1 day (or) once in a week.)

> mkdir jobs

↓

hellojob.yaml :-

Google: k8s API reference

↓

Job v1 batch.

apiVersion : batch/v1 (Group name/version name)

kind : Job

* metadata → type: objectMeta (next line & 2 spaces) (for now, we know only name)

name: hellojob

* spec ⟶ JobSpec (It is not array) ⟶ so, go to next line & 2 spaces.

↓

click on it. (lists all variables)

↓

we can see ⟶ template → This is the Controller.

template ⟶ type: PodTemplateSpec

Controller consist of another k8s object

Describes the Pod that will be created when executing a Job.
The only allowed template.spec.restartPolicy values are "Never" or "OnFailure"

I have a Job which is a Controller.
And this Controller will run a Pod.
and to specify that pod → I'm going to use a section which is called - template
which is of type → PodTemplateSpec ⟶ (next line & 2 spaces)

↓ click on it

- metadata (of Pod which you are trying to write)

| name = jobPod

- spec. → type: podSpec. (click on it). → Containers (array of {}s)

↓

2 spaces & a hyphen

hellojob.yaml :-

```
apiVersion: 'batch/v1
kind: Job
metadata:
  name: hellojob
spec:
  template:
    backoffLimit: 5
    metadata:
      name: jobPod
    spec:
      restartPolicy: OnFailure.
      containers:
        - name: alpine
        • image: alpine
          command:
            - sleep
            - 10s
```

So, whenever you have a Controller object, generally, you wud see 'template' in its spec.

(In template you wud have some other Controller)

Command is a string array

↓

2 spaces + '-'

Here, Job is to sleep for 10 seconds.

(or)

Here you might write a shell script (which will download a file ...)

↓

Here restartPolicy – ?

we want the Job to run and finish.

Here it will not restart.

Bcoz, the basic idea over here is if this Job shud be finished.

---

multipletimes.yaml — CronJob:

↓

CronJob v1 batch. → apiVersion: batch/v1.

kind : CronJob.

Google: Cronjob syntax to run every minute

metadata:

name: periodicjob

spec → CronJobSpec (click on it) → jobTemplate (JobTemplateSpec) → schedule

↓ click on it.

metadata → of the Jobs

spec (JobSpec)

↓

template

↓

metadata

name → livedataPod

This template is Pod template

↓

name: getlivedata job3

CronJob doesn't directly contain Pod.

It contains Job.

Job will have Pod

Pod will have (0)^r

} these are controllers.

- runmultipletimes.yaml :—

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: periodic job
spec:
  schedule: '* * * * *'
  jobTemplate:
    metadata:
      name: getlivedata
    spec:
      backoffLimit: 2
      template:
        metadata:
          name: livedatapod
        spec:
          restartPolicy: OnFailure
          containers:
            - name: alpine
              image: alpine
              command:
                - sleep
                - 3s
```

Controller generally has reference to other K8S objects.

In previous example
↓
(we have written hellojob which refers to the Pod.)

Here, we have written a CronJob which has Job in its template.
Job spec has one more template which basically speaks about running Pods.

Jobspec
↓
backoffLimit → Specifies the number of retries before marking this job failed.
(Limit based on no. of restarts)

activeDeadlineSeconds → Eg: 60 sec.
(Limit based on time)
In 60sec if your script is not finished it will stop.

$ kubectl apply -f hellojob.yaml

?/p: restartPolicy is reqd. (mandatory) (Job needs to be restarted only based on your script's execution)
⇓ changed in yaml files.

c

* For Jobs restartPolicy cannot be 'Always'. (as it needs to be finished)
For Pods → its OK (it can be always). (as job will never finish.)
(For Eg: you have written download file.
What will happen after command is done? → It will exit → Pod will restart now → Job can't finish.

* Jobs have backoffLimit to limit number of restarts and activeDeadline seconds to limit time period of execution.

$ kubectl apply -f hellojob.yaml (Running Job & waiting for completion)

o/p: job.batch/hellojob created.

$ kubectl get jobs -w

o/p :-	Name	Completions	Duration	Age
	hellojob	0/1	6s	6s
	hellojob	0/1	13s	13s
	hellojob	0/1	14s	14s
	hellojob	1/1	14s	14s ——> it is complete.

Jobs by default has completions bcoz it is expected to complete.

- In the Job template ——> we created Pod. (is a Controller)

$ kubectl get po

o/p-	Name	Ready	Status	Restarts	Age
	hellojob-rqmtm	0/1	Completed	0	63s.

As part of Job we have created Pod.

So, when you create Controllers you'll be seeing other objects also.

*Now, if you create Cronjob

↓

You'll be seeing 3 objects — one of each
1) CronJob object
2) Job object
3) Pod object.

Controller contains other k8s resources

Job
hellojob → led to creation of
Pod → hellojob-rqmtm

$ kubectl delete jobs.batch hellojob

o/p: deleted

$ kubectl get jobs

o/p: No Jobs (resources)

$ kubectl get Po

o/p: No resources...

So, when you delete the Controller, the child object which you created also gets deleted.

* CronJob manifest which we have written creates a job every minute and waits for completion.

$ kubectl apply -f runmultipletimes.yaml

O/p:- cronjob.batch/periodicjob created.

$ kubectl get cronjobs.batch

O/p:-	Name	Schedule	Suspend	Active	Last schedule	Age
	PeriodicJob	* * * * *	False	0	<none>	55

$ kubectl get cronjobs.batch -w

O/p:-				Active		(Bcoz we said every minute)
	periodicjob	* * * * *	False	0	<none>	24s
	periodicjob	"	"	1	0s	39s
	"	"	"	0	7s	46s
	"	"	"	0	7s	46s

Ctrl+c.

$ kubectl get jobs.batch

O/p:-	Name	Completions	Duration	Age
	periodicjob-28042755	1/1	7s	22s.

$ kubectl get po

O/p:-		Ready	Status	Restarts	Age
	periodicjob-28042755-n52q6	0/1	Completed	0	30s

So, we have created a Controller — which is a CronJob → and CronJob when it was active → which led to creation of Job → This Job has led to creation of Pod.

* It is the responsibility of the Controller to maintain the state of its child object → So CronJob maintains the state of Job and Job maintains the state of Pod.

* we created a CronJob which runs every minute.

$ kubectl get cronjobs.batch

O/p:- periodicjob * * * * * False 0 49s 2m28s (Right now 2 minutes Passed)

$ kubectl get jobs batch

we are having 3 Jobs that got create

	Completions	Duration	Age
O/p:- periodicjob-28042755	1/1	7s	2m6s
periodicjob-28042756	1/1	7s	66s
periodicjob-28042757	0/1	6s	6s

So Every 1 minute, what K8S is trying to do is → It is trying to create a new Job → that new Job creates a new Pod.

$ kubectl get po

	Ready	Status	Restarts	Age
O/p:- periodicjob-28042755-ns29b	0/1	Completed	0	2m23s
periodicjob-28042756-wcxkg	0/1	Completed	0	83s
periodijob-28042757-6q5sm	0/1	Completed	0	23s

A CronJob created 3 Jobs. These 3 Jobs created 3 Pods.

This will continue. For every 1 min → a new Job will be getting created which in turn creates new Pod.

* Generally we dont run a Job for every minute. we run it once in a day (or) once in 8 hrs . . .

$ kubectl delete -f runmultipletimes.yaml.

O/p: deleted

$ kubectl get cronjobs batch

O/p: No resources

$ kubectl get jobs batch

O/p: No resources

$ kubectl get po

O/p: No resources.

So, Controller maintains the state of Other K8S object.

Eg: Happy New Year mails.

Lets go back to Pods :-

- Pods are not controllers. So they don't maintain state of any other k8s object. The only thing Pod can do is Pod can run a Container. And if you put the restart Policy → depending on the restart Policy → if your ⓒ is not running → it will restart.

> But, ideally, when we want to run applications we always deal with controllers.
> Bcoz – we want desired state to be maintained.

→ Lets run a alpine pod.

$ kubectl apply -f alpine.yaml (It runs for 1 day)

o/p: Pod/alpine created

$ kubectl get po (or) pods. →o/p: alpine 1/1 Running 0 51s

$ kubectl api-resources

	short form
pods	Po
CronJobs	cj

(Jobs doesn't have a short name).

Now, I want to get inside this Container :- (Remember, we used to run Ex: docker container exec -it /bin/bash)

Now, if we want to execute a command in the container of alpine Pod :-

$ kubectl exec alpine -- pwd (pod's name → alpine, command → pwd)

o/p: /

$ kubectl exec alpine -- ls

Inside the alpine pod, I'm running basic commands

o/p : bin
dev
home
lib
:
usr
var

* Till now we ran commands. But we want to get inside :-
(ls, pwd)

TO access the terminal :-

(alpine → sh)

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

we are inside

/ # ls

o/p bin etc lib . . tmp var . .

dev home . . . usr . .

Give your Pod's name

Exercise :-

* Remember, I can have a Pod with 2 Containers.

How to execute a command on a specific Container.

(use --help)

(kubectl exec --help)

See if there is a way for you to pass a containers name.

* Lets run a Pod which exposes a port (or) which runs app^n on some port :-

/# Exit

$ kubectl delete pods alpine

The point is, there is some app^n running inside the Pod. How do I expose it to the outside world ?

$ kubectl apply -f httpd.yaml

o/p: pod/httpd-pod created.

$ kubectl get Pods

o/p: httpd-pod . . . .

This is definitely not the way how we are going to expose

↓

Infact, this is the worst way.

But it works.

Now, if we want to access the application in Container → we can do Port-forward (not recommended approach)

$ kubectl port-forward httpd-pod 8080:80.

we are doing port-forw. using kubectl.
kubectl is present in my master.

o/p: Forwarding from 127.0.0.1:8080 → 80.

Actually it is not K8s Port-forw. kubectl is doing port-for

we will learn something called as 'Services'

This is how you shud be exposing your applications.

This port-forwarding happens on 127.0.0.1.

This is happening only on Local host.

That means that → you can access your application whereever you have kubectl configured.

In my case, kubectl is in master node

Copy public ip of master

> ssh ubuntu@master's public-ip.

ubuntu $ curl http://localhost:8080

o/p: <html><body><h1> It works! </h1></body></html>

It works.

This might not be a fancy page. (This is html Equivalent of It works.)

But, if you want to expose it to the outside world.

$ kubectl port-forward --help

o/p: --address

I want to host it on all the ports. How do I do it?

$ kubectl port-forward --address "0.0.0.0" httpd-pod 8080:80

o/p: Forwarding from 0.0.0.0:8080 → 80.

Now, the app^n is running on 0.0.0.0.

Copy public ip of master.

http://54.244.41.170:8080.

o/p: It works!

But, if kubectl is configured on my laptop → you can access this app^n by accessing my laptop's ip-address. (whereever kubectl is configured → you have to access that IP address → as kubectl did port-forw-)

* There is a Controller called as Replica (Replication Controllers) and Replica set

we will try to tell pod → we will try to tell the no. of pods which we want to run.

& k8s will try to maintain that state.

→ I will try to tell that → I want 10 nginx pods.

what k8s will do is → it will try to create 10 nginx pods in whatever nodes which you have.

* Similarly there is a controller called as Daemonset which will ensure that there is one pod that is running on every node in your cluster.

* All we need to understand now is Controller's functionality.

Controller is a k8s object which controls other k8s resources.

Controllers — ReplicaSet, Labels, Selectors

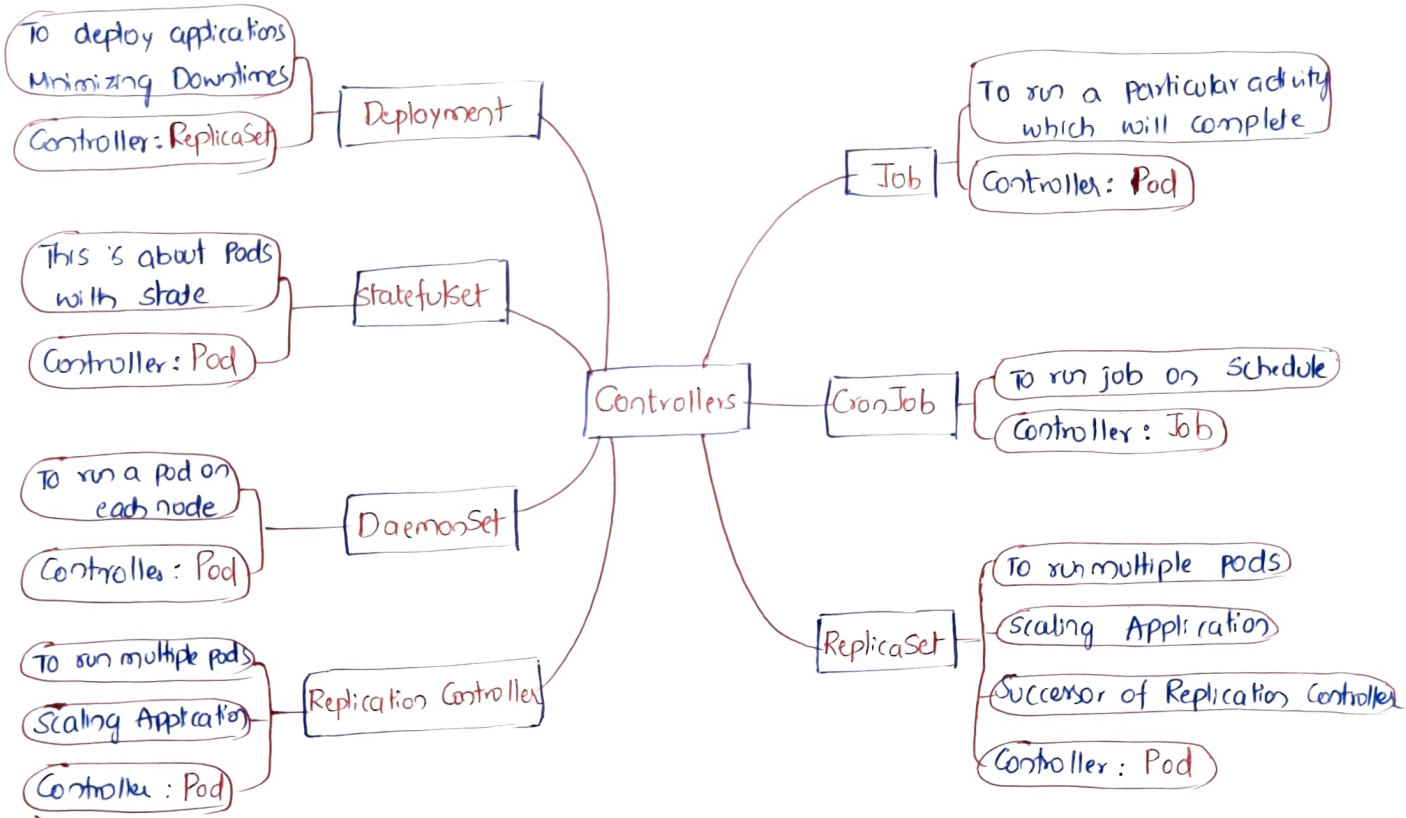
Replication Controllers

* Resources are objects exposed over api's
  Everything is an object.
  Workloads are something where you run your app^n in K8S.

- **Controllers :-**

→ Controllers in K8S control/maintain state of K8S objects.

→ Lets look at controllers which we have :-



1) **Job :-**

* Job is to run a particular activity which will complete.

(Generally, when we run anything in Pod → we expect it to run continuously)

Job is something where you are basically doing an activity, which will take some time and then it will go to completion.

Over here, restart policy becomes mandatory bcoz → Once the Job is successfully completed there is no need to touch. But, if it has failed, then you attempt to restart.

2) CronJob :-

- To run Job on schedule.

3) Replication Controller :

→ Basic Idea of Repli.. contr. is to run multiple pods.

This is the way of scaling your application.

4) Replica Set :

→ It also does exactly the same thing, [Successor of Rep.. controller]

but it is like a successor.

→ we will be using only this.

5) Daemonset :

* To run a pod on each node.

* For some reason, in your K8S cluster, if you have 50 nodes and in all the 50 nodes you need to run one pod per node → we wud go for daemonset.

6) stateful set :

- This is about pods with state. (volume concept)

If you're dealing with app^os which maintain state and if you want to scale them stateful set is the way.

7) Deployment :

→ To deploy applications, Minimizing Downtimes.

This Controller controls Replicaset.

(Cronjob controls Job
↓
Job controls Pod

when we say 'controls'
↓
it means → maintaing the state of it)

* Each Controller controls other object.

ReplicaSet :-

Google: Kubernetes Replicaset

→ ReplicaSet is a Controller which maintains count of pods as Desired state.

→ Let us try to write min 3 simple nginx pods.

Desired state is the combination of Pod and no. of pods which you want

Replicaset-Activity-1 - Create 3 nginx Pods :-

Folders ↓

Manifests → ~~k8s~~ → new file.

nginx-rs-yaml :-

I want to write a Replica Set

↓ API reference

Replicaset v1 apps

↓

apiVersion: apps/v1

kind → ReplicaSet.

metadata → name: (we only know name so far).

nginx-rs.

spec ↳ type: ReplicaSetSpec (Give 2 spaces + next line)

↓ click on it.

* minReadySeconds: 1 → For every pod how much time do you think it will take to get into Ready state. By default it is zero.

* replicas: 3.

Lets not write selector & see what happens.

* template → PodTemplateSpec. (click on it).

↓ — metadata: name: nginx-pod.

↳ spec (PodSpec) → containers

- name: nginx

image: nginx:1.23.

ports:

- containerport: 80

In Kubernetes

command will go & overwrite Entrypoint

Args will go and overwrite Command.

Lets try to create a pod (without selector)

master

$ kubectl apply -f nginx-rs.yaml

o/p:- spec.selector: Required value. (error)

It's trying to tell → it needs a selector.

* We will be getting into Label in a detailed way. For now, temporary fix.

nginx-rs.yaml :-

Temporary workaround for adding selectors

We need to understand the need & necessity for Labels.

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
spec:
  minReadySeconds: 1
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.23
          ports:
            - containerPort: 80
```

$ kubectl apply -f nginx-rs.yaml

O/p :- replicaset.apps/nginx-rs created.

$ kubectl get replicaset.apps

O/p :-

Name	Desired	Current	Ready	Age
nginx-rs	3	3	3	135

Remember we said, we want 3 nginx pods.

$ kubectl get po

O/p :- nginx-rs-lwb2f 1/1 Running 0

nginx-rs-qrq2w 1/1 " 0

nginx-rs-rqxv4 1/1 " 0

we can see 3 pods.

In the Replicaset → the Desired state was → you wanted 3 pods

→ Current state is - you have 3.

so, it is trying to maintain the desired state.

This Controller (Replicaset) has created 3 pods.

we can give count dynamically also. (count = 3)

For now, learn static no. of pods

$ kubectl api-resources

O/p :- — — —

— — —

(rs) → short-name for replicaset →

$ kubectl get rs

$ kubectl describe rs nginx-rs

O/p :- Name : nginx-rs

Namespace : default

Selector : app = nginx

Replicas : 3 current / 3 desired ✓

Pod Template.

Labels : app = nginx

Port : 80/TCP

$ kubectl get po

O/p. nginx- -

" } 3 pods running.

"

Now delete pods.

$ kubectl delete pods nginx-rs-dwzf nginx-rs-qrq2w nginx-rs-xqxw4

O/p: deleted 3 pods.

I have deleted all the three pods.

But ↓

$ kubectl get po

O/p: nginx-rs-87.. - — 3s

" " - 5f.- — - 3s

" " - 2m... - - 3s

I got pods with new names. (I have deleted the pods previously).

The moment I deleted the pods → new Pods got created. → why?

Bcoz the desired state was 3 pods. So, it will try to maintain the count of 3. → This is basically what ReplicaSet does → maintain desired state.

Replication Controller also does exactly the same thing.

* Lets change the replica count

↓

For everything you have declarative way & Imperative way.

Google: kubectl cheat sheet.

↓

scaling resources → kubectl scale --replicas=3 (or) 4...

↙ not a good way. (Always go with changes in manifest — (not by commands)

$ kubectl scale --replicas=5 rs/nginx-rs

bcoz commands don't have history. If you have made the changes in the file → that can be added to your version control → you will have history of each & every change which you have done)

O/p: --- scaled.

$ kubectl get po

O/p: (5) nginx-... 2m1s
" 2m1s
" 2m1s
" - 2s ago
" — — 2s ago } latest created.

So whether you want to increase the count (or) decrease the count manually → there is a way.

$ kubectl scale --replicas=1 rs/nginx-rs

o/p: scaled.

I'm reducing count to 1.

$ kubectl get po

o/p. nginx-rs-s1fz6 1/1 . . . 3m12s.

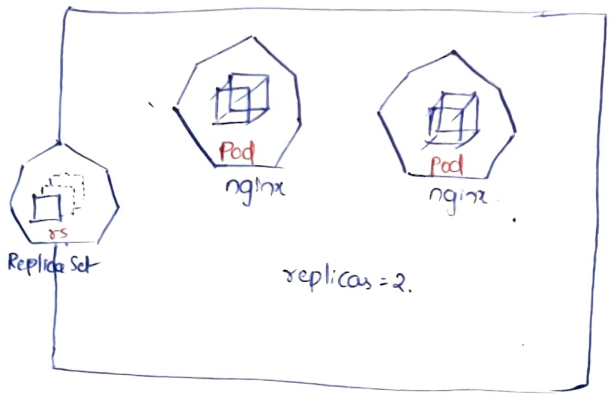
So increasing is scaling-out → no. of pods

decreasing is scaling-in.

(scaling-up → ↑ size
-down → ↓ size)

So, I can ↑ the no. of replicas (or) ↓ the no. instantaneously.

* When we containerize our application → we have written Docker files and we created image → we were trying to run Docker ⓒ to run ⓒ on a single node. From there — we came to the state — where we write a manifest file and our application can be scaled to even 100 no. of pods — running on multiple machines. The thing which we have not done so far is — access the application.



Any replication controller is a reconciliation loop.
It always acts like a watchdog
It always look for
-any change that needs to be done to match desired state.
It informs state back to etcd.
(All comm^n is via api server)

ReplicaSet - Activity-2 - Create 5 Pods with Jenkins and alpine in One Pod :-

Scaling out & in → means increasing count (Eg- 1 Pod → 5 pods)

Scaling up & down → means increasing size. (1GB RAM → 2 GB RAM)

jenkins-rs.yaml :-

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: jenkins-rs
spec:
  minReadySeconds: 5
  replicas: 5
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      name: jenkins
      labels:
        app: jenkins
    spec:
      containers:
        - name: jenkins
          image: jenkins/jenkins:lts-jdk11
          ports:
            - containerPort: 8080
        - name: alpine
          image: alpine:3
          args:
            - sleep
            - 1d
```

2 containers < jenkins, alpine.

$ Kubectl delete rs nginx-rs

$ Kubectl apply -f jenkins-rs.yaml

$ kubectl get rs

		Desired	Current
O/p:	jenkins-rs	5	5

$ kubectl get po

		status	
o/p.	jenkins-rs-7r..	containercreating	15s

we gave minReady seconds = 5s

$ kubectl get po

		status	Age
o/p:	jenkins-rs-7r..	Running	48s

They are in running state bcoz 5 seconds are finished.

$ kubectl describe rs jenkins-rs ← Get Events from describe rs.

Now lets delete one pod manually

↓

$ kubectl delete po jenkins-rs-s76t4

o/p: deleted.

$ kubectl describe rs jenkins-rs

o/p:

— —	
— —	
— —	2m39

In the Events, it speaks about creating a new pod which is 135ec ago.

successful create 13s → Pod created.

↓

Now, count is maintained. $ kubectl get rs

Exercise: Write a manifest to create

- nginx replication controller with 3 Pods.

$ kubectl delete rs jenkins-rs

* If I have necessary nodes with me, I can set Pods count to 100 also.

for Eg: If I have enough CPU & RAM to run 100 pods → My cluster can create it

But, how do I access it from outside world?

for us to do that, we need to understand imp. topic of k8s → Labels.

Labels :-

* In K8S, everything is stored in etcd cluster.

In etcd cluster – it stores name-value pairs (or) key-value pairs.

* Google : kubernetes labels.

* Labels are key-value pairs that can be attached as metadata to K8S objects.

↓
why shud I do this ?

→ Labels help in selecting/querying/filtering objects.

For Eg: I want to get all the nginx pods → K8S cannot do that.

✱ I want to get all the pods with this label → K8S can do this.

So, K8S cannot know what is running inside your pod.
The only way for us to query the information is by attaching labels.

Label Activity 1 – Create a nginx pod with label :-

→ Lets create a nginx pod with label → 'app : nginx'
↓ Key ↓ value.

From now onwards,
Anything & Everything which we create in K8S → Lets have Labels

→ New folder : labels → new file → labeldemo1.yaml

Google → API ref → pod v1 core
↓

```
api version : v1
kind → Pod.
metadata : → click on object metadata. → labels
    name :
    labels
    { type: object }
Eg:- app: nginx.
spec :
    con-
```

→ Map of string keys & values that can be used to organize (scope & select) objects.
May match selectors of repln controllers and Services

* labeldemo1.yaml :-

```
---
apiVersion: v1
kind: Pod
metadata:
  name: labeldemo
  labels:                    → (we added labels)
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.23
      ports:
        - containerPort: 80
```

$ kubectl apply -f labeldemo1.yaml

o/p: pod/labeldemo created.

$ kubectl get po --show-labels

o/p:

Name	Ready	Status	Restarts	Age	LABELS
labeldemo	1/1	Running	0	10s	app=nginx

Now I have given a label → app=nginx.

Lets run some other Pods using declarative

Google: kubectl cheat sheet → se

$ kubectl create --help

Google: create pod declarative. (We use manifest file). Imperative → we use commands directly

$ kubectl run n1 --image=nginx

o/p: pod/n1 created.

$ kubectl run n2 --image=nginx

o/p: pod/n2 created.

$ kubectl run n3 --image=ngin.

o/p: pod/n3 created.

$ kubectl get po --show-labels

O/p:

			Labels
labeldemo	1/1	Running	app=nginx
n1	1/1	"	run=n1
n2	1/1	"	run=n2
n3	1/1	"	run=n3

Now, I have 4 pods. All the 4 are running nginx. But can you make out whether they are running nginx (or) not ? → NO.

K8S also cannot make out.

In K8S, if you want to query something → you can do so only o.b.o labels.

* Now what I can try to tell is —

Get me all the pods which have a label called as app=nginx. Then I'll get this (labeldemo)

Get me all the pods which have label — run=n2.

Then I will get n2.

So, I can add labels, delete labels.

* whenever you are basically querying the items, there are 2 kinds of things which you can query on

→ Equality matching
Eg: app=nginx.
(when app is nginx)

→ One is called as Equality → that means when label matches

→ If you want to write some conditions, that is called as Set-based.

Eg:- Get me labels in such a way that I can write conditions where I'll try to tell. app can be nginx (or) httpd

There are 2 kinds of conditions which you can write

1) Equality-based requirement → means exactly Equal

2) set-based requirement → Eg: app is web (or) nginx (or) db (anyone of them, if it is present, then consider that pod)

Using these things, you can query the information.

$ kubectl delete po n1 n2 n3

* Now, I'm trying to create pods (Imperatively) and giving labels:-

$ kubectl run n1 --image=nginx --labels "app=nginx, component=web" (no space here)

$ kubectl get po --show-labels

					LABELS
o/p:	labeldemo	1/1	Running	28 min	app=nginx
	n1	1/1	"	7s	app=nginx, component=web

I have 2 Pods now.

One pod has → app=nginx

other pod has → app=nginx & component=web.

* Now, lets create one more pod.

$ kubectl run n2 --image=nginx --labels "app=web, component=web"

o/p: pod/n2 created.

$ kubectl get po --show-labels

o/p:	labeldemo	-	-	app=nginx
	n1	-	-	app=nginx, component=web
	n2	-	-	app=web, component=web.

I can apply these labels to any K8s objects.

- Labels
  - Equality-based → you will have 3 simple operations → =, ==, != } both are same here.
  - Set-based → you wud have in / -notin
    - Partition.
    - ! partition → means I don't have a label called as partition.

* Lets try to write some selectors.

* 1st I want to get all the pods → which have label → app=nginx.
  we have 2 of 3 (in our prev. example)

$ kubelctl get po --selector "app=nginx" --show-labels

o/p:-	labeldemo	1/1	-	-	-	app=nginx
	n1	1/1	-	-	-	app=nginx, component=web

I got only 2 → ✓

$ kubectl get po --selector "app != nginx" --show-labels

O/p: n2 1/1 Running 0 2m42s app=web, component=web.

I have only one pod.

So, you can try to write equals (=) or not equals (!=)

* Now, I want to write a selector in such a way that

⇩ app in (nginx, web)

$ kubectl get po --selector "app in (nginx, web)" --show-labels
(space)

O/p: I get all 3 pods.

So, when app is either nginx (or) web.

$ kubectl get po --selector "component in (web)" --show-labels

O/p: n1 1/1 Running 0 app=nginx, component=web
n2 1/1 Running 0 app=web, component=web

So, I have 2 labels, where we have component = web.

* I can try to write either component (or) app in web

↓

$ kubectl get po --selector "component, app in (web)" --show-labels

O/p: n2 1/1 . . . . . app=web, component=web.

~~So, there is only one pod → where I have app =~~

$ kubectl get po --selector "component, app in (web, nginx)" --show-labels

O/p: n1 - - - app=nginx, component=web
n2 - - - app=web, component=web.

(3rd one didn't occur bcoz there was no label called as component)

Now, you can update label by using kubectl command,
but that is dangerous → Don't do that.
Changing labels from cmd line is good for experimentation, but disastrous when you are trying to run them on production.

* So, can I select K8S objects now — ? → Yes (using labels, selectors)

* Go to previous file: jenkins-rs.yaml

```
ap-
kind -
metadata:
  name:
Spec:
  minReadySeconds:
  replicas:
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      name:
      labels:
        app: jenkins
```

I'm trying to tell that
I have created a pod with label app=jenkins.

Here I'm trying to that
Give me a selector with app=jenkins.

$kubectl label --help

$ kubectl get po

o/p: labeldemo
n1
n2

I'm trying to give labels

$ kubectl label po labeldemo app=jenkins
(labeldemo: pod name)

o/p: error: 'app' already has a value (nginx), and --overwrite is false.

$ kubectl label po labeldemo app=jenkins --overwrite.

o/p: pod/labeldemo labeled.

I can give labels to multiple pods.

$ kubectl label po labeldemo n1 n2 app=jenkins --overwrite

o/p: pod/labeldemo not labeled.
pod/n1 labeled
pod/n2 labeled

$ kubectl get po --show-labels

o/p labeldemo - - - app=jenkins

n1 - - - app=jenkins, component=web

n2 - - - app=jenkins, Component=web

So, I have 3 pods with the label - app=jenkins.

$ kubectl run n3 --image=nginx --labels "app=jenkins"

o/p: pod/n3 created

$ kubectl run n4 --image=nginx --labels "app=jenkins"

o/p: pod/n4 created

Create 5 Pods with label app=jenkins

↓

Now run the replicaset with 5 replicas

$ kubectl get po --show-labels

o/p: labeldemo - - - app=jenkins

n1 - - " " , Component=web

n2 - - " "

n3 - - " "

n4 - - " "

So, I have 5 pods which have label → app=jenkins.

Am I running Jenkins inside them? → No

But my label was app=jenkins.

* Now run the replica set with 5 replicas. Lets see what happens.

$ kubectl apply -f jenkins-rs.yaml.

o/p: replicaset.apps/jenkins-rs created.

$ kubectl get rs jenkins-rs

$ kubectl get po

o/p: labeldemo

n1

n2

n3

n4

Are you seeing atleast one extra pod getting created?

NO.

What is happening is? → It is not creating Pods. It knows about the Pods which are there. It will only try to maintain desired state.

How it will try to maintain desired state? → It needs 5 pods which matches the label description → app=jenkins.

So, I already have 5 Pods whose description is (label) app = jenkins

So, it has not created even one extra pod as desired state is already present

$ kubectl delete po n4

O/p: n4 deleted.

Now, I deleted one pod.

$ kubectl get po

O/p: jenkins-rs-vzvbq - - - 3s ⟶ we see one pod is added.

labeldemo

n1

n2

n3

* Selector is not only to create new stuff

If there is already a pod that matches this label, replicaset will not create one more.

So K8s tries to maintain desired state with the help of these labels.

K8s as a Service (Azure-AKS), Internal & External Communication

* So, in Replication Controller, there is no concept called as MatchLabels. In Selector, you have to directly write a Label.

- **Kubernetes as a Service (Managed Kubernetes)**

→ Every cloud Provider gives you some flavor of k8s.

* Azure : AKS
* AWS : EKS
* GCP : GKE

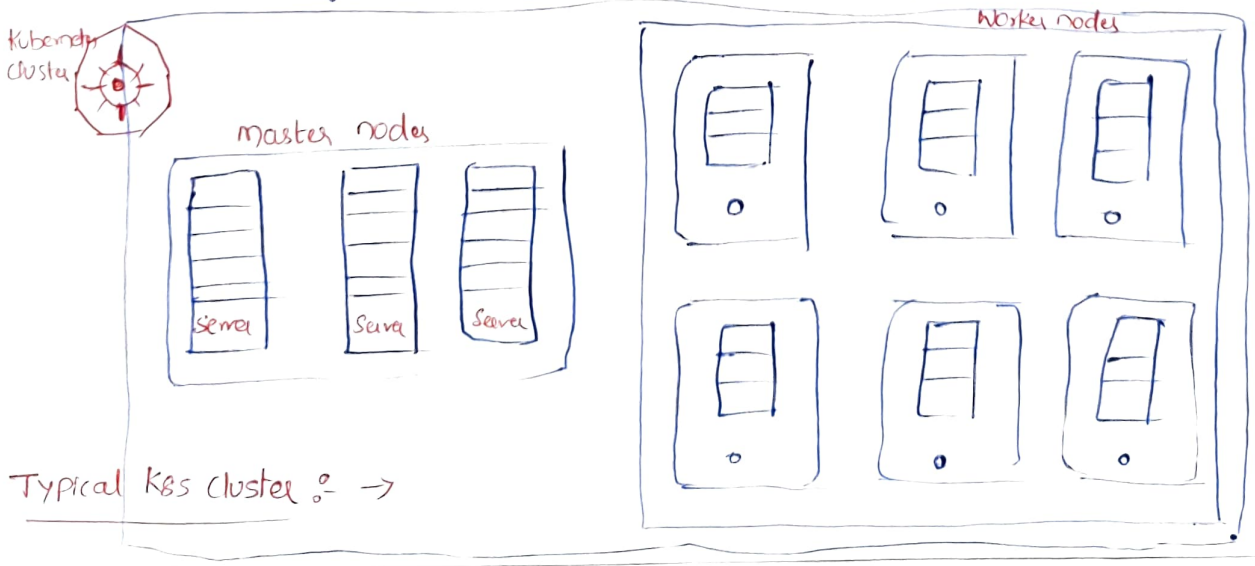
→ k8s as a service basically means

the master nodes will be managed by Cloud Provider.

(In kubeadm, we had 2 Node ~~cluster~~ plus 1 Master ~~node~~ Node). (Master node was necessary for us to run the cluster.) (and actually we were running all of our application pods on our worker nodes)

When you go to k8s as a Service, let it be AWS, Azure or GCP, master nodes will be completely managed by them. There is no need for you to create master nodes.

→ You can have 1 or more components for master nodes. So if you have multiple master nodes, that becomes High-Availability (HA) cluster. And then you wud have worker nodes.



Typical k8s cluster :- →

* When you are installing K8s on your own servers → you need to have master nodes as well as worker nodes.

So, we have chosen 1 master node & 2 worker node cluster. Whenever we wanted to interact with K8s cluster → our Kubectl was configured within the server. we were logging in and doing that.

But ideally in production, we'll generally not do this.

* When you go to cloud → what would happen? → Lets look at the transformation.

⇓

In our previous case → we are managing Everything.

But, once you go to the cloud → you no longer have access to master nodes.

These are created by your cloud providers. You won't be managing master nodes.

Bcoz cloud will do it for you.

For this they will have hourly charges. (approx - $0.2/hr)

Azure has a basic cluster → where your ... for cluster will be zero. (If you want advanced features → Pay for it)
for basic features ↗

The worker nodes are nothing but virtual machines.

In case of AWS → it is EC2 M/C, In case of Azure → it is Azure VMs.

* Generally, these nodes can be autoscaled.

Generally, when you go with on-premise cluster, if you create a 3 node cluster, & if you want to make it 4 nodes → again you have to add one node → & use kubectl join command (or) kubeadm join command ....

But once you go with managed cluster, if you want to ↑ no. of nodes, all you have to do is just tell the cloud Provider. There is no need for you to do extra configs

- So, in on-premise cluster, we can scale pods but we cannot scale nodes.

Once you starts using K8s as Service, since cl. pr. supports scaling → you can also scale nodes.

• Advantages :-

1) Less administration (compared to what I do on on-premise)

2) Nodes can be scaled.

3) inbuilt support for cloud integrations. (Cloud Controller Manager (CCM) will bring in everything necessary to interact with the cloud). which has intelligence of the cloud.

For Eg: If you want to interact with some db (Rds) → which is running in a Private network in AWS → You can integrate that with K8s which is running on AWS.

Eg: You can attach load-balancers. etc.

* Till now, cluster which we were using had only 4 architectural components

- api Server
- Kube-scheduler
- etcd
- controller-manager

Now, in addition to that, the moment you go to k8s as a Service we have CCM.

* In this course, we will be using EKS & AKS.

we will mostly use → AKS → easy to set up.

• Setting up basic K8s Cluster in Azure (AKS) :-

*difference :-

Now you wud be having a Laptop. In this Laptop, you wud be having Kubectl. (bcoz you no longer have access to the master nodes).

So, in your laptop, install Kubectl. Using Kubectl on laptop, → you interact with K8s cluster.

* In these setups, we configure Kubectl on

→ dev systems

→ build servers

wherever you are creating a CI/CD pipeline, this kubectl will be configured in Jenkins Agent (or) Azure-devops Agent. From there you will be doing CI/CD deployment

Google:- Kubectl Installation

↓

Install tools (or)

we can use Azure CLI to set it up.

* Installing AKS requires → Installing Azure cli

bcoz, we need to get kubectl configured.

(If you remember, when you were creating kubeadm cluster, there was something called as Kubeconfig file which you might have copied to the home directory.)

So to get that kubeconfig file → we need Azure cli.

Google: azure cli install

↓

How to install (microsoft.com)

— Install on windows

— Install on others.

(We will setup kubectl on our laptop. But we will also set it up on Linux M/c)

Laptop

Kubectl from our local M/c will have access to the dev clusters of Kss, not really prod.

dev

Ideally, we will be working with Kubectl configured on a build server

Build server (Linux)

prod

So we need to know both.

* Once you have installed azure cli,

1st thing you need to do is → 'az login'. (command) (Enter Azure credentials)

* After this ↓,

Google: azure aks quickstart. (Deploy aks using Azure cli)

You need to execute certain commands.

* Lets create a Linux M/c (Azure) :-

You can also use Docker container called Azure cli

Resource Groups

↓

+ Create

↓

Resource Group: kubectl

Region: East US

↓

review + Create.

↓

Virtual M/c → + Create.

(So, in CI/CD pipeline, this M/c will be your build server

Bcoz it can connect to Kss → this is where you'll be using kubectl apply commands → which your jenkins (or) azure-devops will basically call.

Azure virtual M/c.

↓

Resource Group: kubectl.

VM name: gtKubectl.

Image: ubuntu 20.04

B1s.

User name: Dell.

Key pair: use existing →

stored keys: my_id_rsa.

Publi_inbound ports: Allow.

SSH (22).

Review + Create.

(Not really worried about ports as we are not using it for port-forwarding) → Create.

→ we will set up kubectl in Linux M/c

But
you can set it up on windows Laptop also

* Since it is a Ubuntu M/c. → Azure-cli Install-on linux. (microsoft.com)

↓

I have one script.

```
curl -L https://aka.ms/InstallAzureCli | bash
```

The concepts which we are learning → we will be able to run it on any cloud.
(till spoken about specific cloud)

(copy public-ip).

```
Dell> ssh 172.190.121.203
```

'yes.

Git Bash
(rename: kubectl on linux).

1st step → I want to install Azure cli.

(or) curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

```
$ curl -L https://aka.ms/InstallAzureCli | bash
```

Enter
Enter . . . . Run the CLI with /home/Dell/bin/az --help.

This will set up azure cli for me.

tab completion - to be activated.

```
$ source ~/.bashrc
```

$ az --help

$ az login

To sign in, use a web... https://... and enter code

click on ↙ and enter code. ✓

Pick an Account. ✓

Now, lets go with K8S Installation :-

Google: azure aks quickstart → Deploy aks using Azure cli

1st Command → Create a resource group.

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

O/p: resource group created

next

Create AKS cluster: (This is a K8S cluster with one node)

If you want you can ↑ the count (for now, 1 is enough)

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

→ will take sometime → 3-4 min for Azure (AKS)
20 min for EKS
13 min → Terraform

↓

This Command will create a K8S Cluster

Once I have this cluster, I have to install kubectl on this VC and then get Kubeconfig file.

I can get kubeconfig on multiple systems if I want.

* Azure CLI is a way to connect to Azure using CLInterface.

But, even in Azure, if we go with Advanced K8S cluster → it will also take 20 min

Here we are creating a very basic K8S cluster.

* If you want kubectl → $ az aks install-cli

I dont have Kubectl in this M/C.

$ Kubectl --version

O/p. command not found

$ az aks install-cli

Error: Permission denied.

$ sudo az aks install-cli

Sudo not found. (or I have to add the Path.

Path of az → /home/Dell/bin/

root: # /home/Dell/bin/az aks install-cli

Exit.

$ Kube ctl version.

To get your Kube config file :-

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

O/p: Merged "myAKSCluster" as current context in /home/Dell/.kube/config.

$ Kubectl get nodes (we selected → count: 2)

O/p	Name	Status	Roles	Age	Version
	aks-nodepool1-...-vmss...000	Ready	agent	4m11s	v1.25.6
	" " " 001	Ready	agent	4m32s	v1.25.6

we are using K8s version of 1.25.

Google: Kubectl cheat sheet

↓

echo... (kubectl autocomplete)

$ echo "source <(kubectl completion bash)" >> ~/.bashrc

$ Kubectl get api-resources

So, we have set up the K8s cluster. We have 2 nodes.

* Now, go to Azure → Resource Groups → my Resource Group → my AKS cluster.

Nodepools → nodepool1 (will have 2 nodes).

nodepool1 → represents similar kind of nodes

Nodes

(we can always scale this → Upgrade kubernets
Update Image
Scale node pool
(we can select Auto-scaling)

* Now, in my Laptop,

(azure cli
kube config
kube cluster) steps (do this)

So,

* Over here, you are not creating VM separately.

You are just trying to tell that — you want a cluster with how many nodes.

It has created a 2-node cluster with AKS (which is managed).

Since it is managed → there is no kubectl around → I have to configure →

for that I need a kubeconfig file → To get this, there is need for azure CLI.

(Every kubectl command needs a kubeconfig file to connect to the cluster)

In a kube config file, you will have api-server's details as well as some credential details.
↓
they wud be some keys.

* (Now, we have a K8s cluster.)

- Exposing Applications running in Cluster to External as well as Internally when scaled :—

We created 3 pods. I want to have — a way to access these.

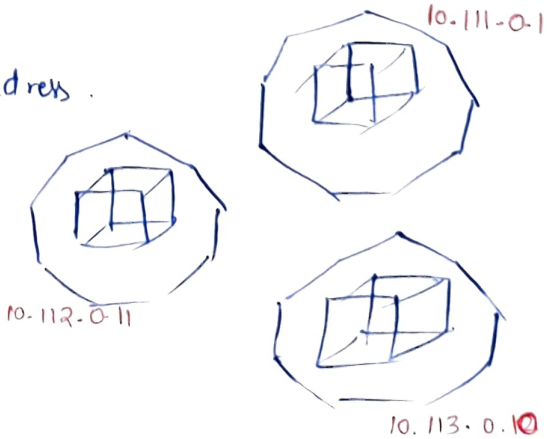
But what is the problem?

Every Pod comes up with its own IP-Address.

Let's assume IP addresses are → 10.111.0.10
10.112.0.11
10.113.0.10.

On this my nginx is running.

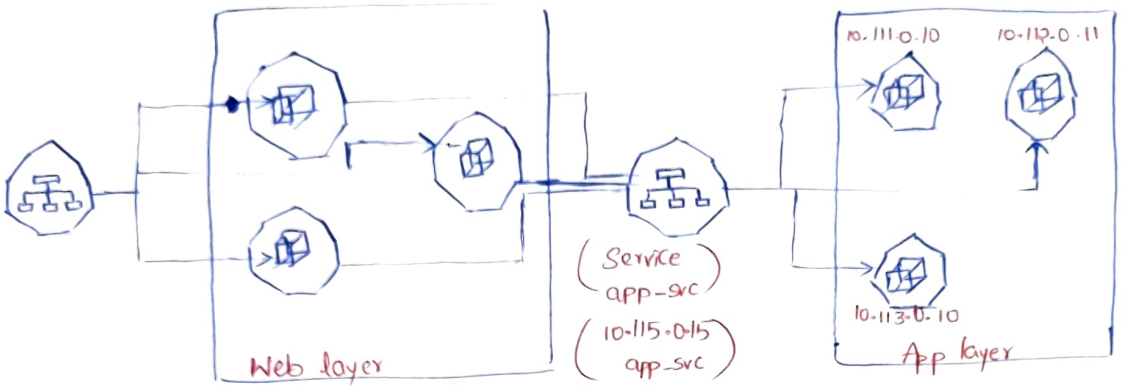
So what shud I basically give.



Let us assume that this represents web layer of your application.

we also have a app layer.

So, we agree that web servers need to deal with app servers.



The problem is how will web server speak with app server.

which IP address shud it use?

If it is one pod → there is no issue. (You can directly communicate from web server to app server).

The same problem arises in db also →. In db, lets assume that I have 3 pods running my database. which pod's IP-address shud I give?.

→ Generally, the basic thing which we do, in normal servers,

for eg lets assume that these are all not pods, rather these are VMs,

So what we do is → We basically create a load balancer.

And load-balancers are of 2 types. 1) Internal LoadBalancer → work within inside your Organization.

2) External Loadbalancer.

↓

is to take traffic from Internet (or) from External sources

So

* How can web layer communicate with App layer?

↓

In K8S, there is something called as Service

Here we'll be creating → app-svc. The best part of app-svc is → It recieves an IP.

↓

Eg:- 10.115.0.15.

and also it has a name → app-svc.

(c)^S can be resolved by using names as well as IP addresses.

- Any web server who wants to communicate with app → will not directly communicate by the pod's IP-Address, but they wud be interacting with app-svc

- What app-svc might do is → it might interact with 10.19.0.10, 10.112.0.11, 10.113.0.10
  This is what k8s service does.
  Service does some kind of loadbalance.

But there is one interesting fact about this service.

↓

Every service gets an IP-Address.

(But understand one thing → service is not a workload)

Every service gets a virtual IP-address which will never change.

- So, whenever you are interacting with the service, there is networking that is happening. (i.e. k8s networking happens).

So Service gets a virtual IP-address — which forwards to one of the Pods.

But, how will service select the pods?

So, what is only way of selecting resources in k8s?

If I have to get some items → ?

Service will know to which pod it has to forward the traffic
o.b.o Selectors
and it knows only
Equality-based selectors
So you shud match labels.

For Eg:- we will be trying to tell that → Service → whenever someone hits you → forward the request to all the pods which have a label called as Eg: layer is app.

So any Pods which have layer as app → k8s Service will forward the request to one of the pods.

* So, k8s load-balancing works out only o.b.o Labels.

This is Internal.

* Now, let us assume that, I want to make it to the External world.

So, Here also we will have a service.

To expose it to the outside world, what are the different things, which we wud have?

The point is, K8S is running on nodes. It takes the help of your nodes.

So what are these ways & how can we do that?.

But,
the basic idea of service is → rather than accessing individual pods.
So, you can't connect from one app to other app 0.b.o IP addresses (or) Pod names bcoz they are going to change all the time. The only thing is we need something which is fixed → Fixed is → K8S Service.

Eg: replicaset → if we delete one pod, other gets created which has new name & new IP

K8S Service can know which Pod to forward the traffic only 0.b.o. labels.
(Internal communication happens like this) (External comm? → we will figure out).

- Every Pod gets a Unique IP and name.
- Connecting from one pod to other 0.b.o. name/ip might not be a good idea as pods are controlled by replicasets (or) other controllers (Any pod might die & you might get a new pod altogether)
- K8s has a Service which helps us in connecting to Pods with similar behaviour but by using labels.
- Each service gets a ip address and this is virtual ip which helps in forwarding traffic to one of the pod based on the labels. This ip is called as cluster ip.
- Services can be exposed to External world.
- Service is similar to Layer 4 load balancer (that means, it understands protocol, ip address, ports)

Remember, Service is not a component.
So what will happen if Service goes down? = Nothing.
(It cannot do URL-based routing)
Remember → it is just a virtual ip-address. It is not a pod.
The only case → service can go down is → when your cluster is down. (or) 1st case
2nd case: kubectl delete service

* Remember kube proxy → which is responsible for networking of k8s cluster, which will have networking components. And these networking components are controlled by k8s master.

So, this service's details are basically present in your master nodes — where it says that whenever someone accesses this service → just forward it to one of the pods

* Lets assume we have created a k8s cluster.

This k8s service internally

(Depending on what you want to run you can 1^x no. of nodes.

↓

2 nodes < Node 1 / Node 2.

(Lets assume we have 2 nginx pods)

nginx-1 (Label → app: nginx)

nginx-2 (Label → app: nginx)

Then two pods are created with the help of replicaset (nginx-rs)

Now,

I'll be trying to create one more pod → alpine-pod.

Now alpine-pod wants to speak with nginx-pod.

(the point is I have 2 nginx pods → so which pod's ip-address shud I use) → Bcoz I don't know whether nginx-1 is alive or not.

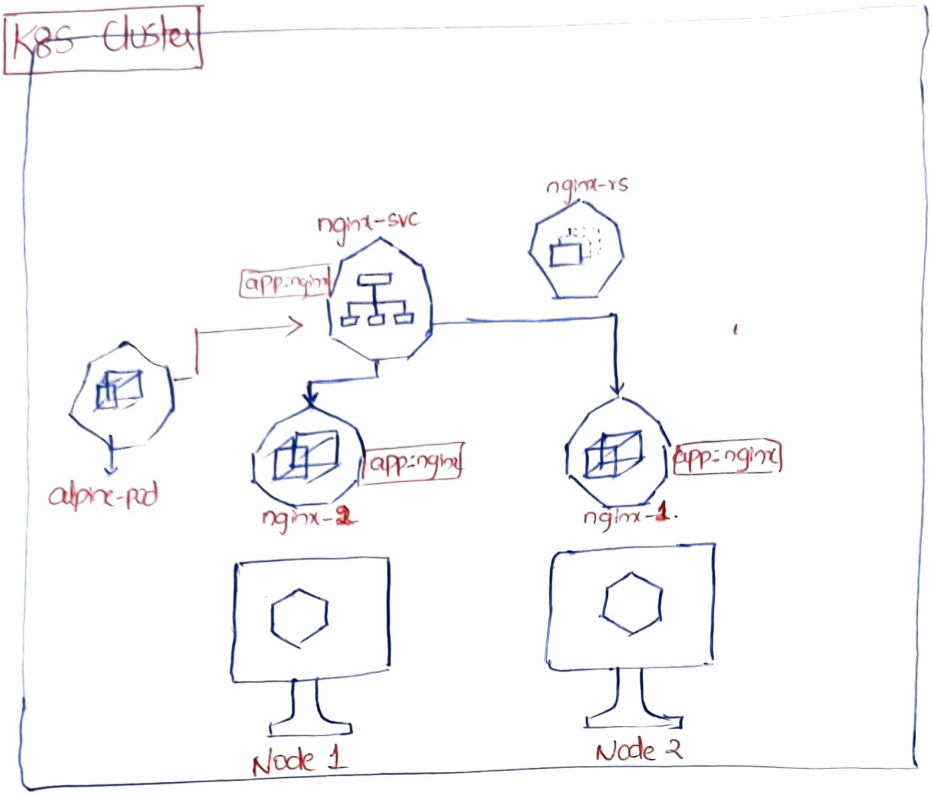
For that we are creating → (nginx-svc)

Now, when I want to communicate with nginx-pods → what I'll do is → I'll communicate with nginx-svc. & So, nginx-svc → I'll be creating it with Label → app: nginx. → So, this will obviously match either pod (nginx-1) (or) pod (nginx-2)

But I'm not trying to access it from outside. I'm accessing it from inside the k8s cluster.

So, this is my Internal communication workflow.

k8s Cluster – Internal Communication using k8s Service



Consider the following :

- We have an alpine-pod which needs to consume nginx
- but nginx is a replica-set and there can be 'n' replicas (so, how many replicas does nginx-pod have is something which alpine-pod is not interested in)

alpine-pod wants to communicate with nginx-pod.

(Like this is web-app? wants to speak with middle-ware)

* Lets create a nginx-rs

New folder → src → internal-demo → nginx-rs.yaml.

api-reference → apps/v1 (Replicaset v1)

kind → ReplicaSet

Metadata name → nginx-rs

labels: → These are labels for replicaset (not for pods)

app : nginx.

key : web

version: "1.23". (Label - key-value pair - String)

If I don't put " " → it will become a number.

Spec:

minR - 1

replicas = 2

selector:

matchLabels :

app: nginx.

nginx-rs.yaml :-

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
  labels:
    app: nginx
    layer: web
    version: "1.23"
Spec:
  minReadySeconds: 1
  replicas: 2
  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
```

We shud use API-ref
↓
Use 1.25 (depending on K8S version)

I can write one more manifest by putting 3 hypens over here.

Now lets create :→ nginx-svc.yaml

Google : kubernetes service.

API Reference.
↓
Service APIs
↓
Service v1 core.
↓
type: core, v1

→ apiVersion: v1

→ kind : Service.

→ metadata :
name : nginx-svc.

→ Spec
↓
ServiceSpec.
↳ ~~clusterIP~~ .

Short name of Service in api-resources
↓
(svc)

selector: (I'm interested in all pods with label app:nginx)
app: nginx.
type: ClusterIP (Same we have done in replica set)

means.

means

ClusterIP — An IP Address which works only within K8S cluster.

Use Case: I want an alpine-pod to speak with nginx Service (Internal Comm) (which can happen thru clusterIP)

Next Create alpine-pod.yaml

nginx-svc-yaml :-

```
---
apiVersion : v1
kind : Service
metadata :
  name : nginx-svc
spec :
  selector:
    app: nginx
    type : ClusterIP
    ports :
```

alpine-pod-yaml :-

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

- Kubectl is configured on my laptop now.

> cd \svc\internal-demo

> kubectl apply -f .\nginx-rs.yaml

o/p: created

> kubectl apply -f .\nginx-svc.yaml

o/p: invalid. spec.ports: Required.

ports :
service

So, port that will be exposed by the service =>

it is 80.

(or)

You can put 8080 → so whenever you access 8080 port, it will try to access 80 port on your ©r.

targetPort : → where your ©r is running.

what is inside your ©r → nginx ↓ 80 port

protocol → Default: TCP.

name → must be a DNS_LABEL.

```
ports :
  - name : nginx-svc
    port : 80
    targetPort : 80
    protocol : TCP.
```

> kubectl apply -f .\nginx-svc.yaml

✓ ✓

> kubectl get rs

o/p: nginx-rs 2 2 2

> kubectl get po

o/p: nginx-rs-6g 1/1 Running - -

" " -29 1/1 Running - -

> kubectl get po --show-labels

o/p					Labels
	nginx-rs-6g	1/1	-	-	app=nginx, version=1.23
	nginx-rs-29	1/1			app=nginx, version=1.23

> kubectl get svc -o wide.

o/p:-

Name	Type	Cluster-IP	External-IP	Port(S)	Age	Selector
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP		<none>
nginx-svc	ClusterIP	10.0.242.134	<none>	80/TCP		app=nginx.

So, there is already a service which is running in k8s which is called as kubernetes service. (For now I ignore it).

Our area of interest is → nginx-svc → It got some unique IP address.

> kubectl apply -f .\alpine-pod.yaml (Create an alpine Pod and login into that)

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

/ # → we are inside the pod.

→ Ping nginx-svc by its IP-address :-

/ # ping -c 4 10-0-242-134

-c → 4 means ping 4 times.

It is working.

o/p: 4 packets transmitted, 0 packets received, 100% packet loss.

Lets try to access 10.0.242.134

/ # curl http://10.0.242.134

/bin/sh:

o/p: curl: not found

/ # apk add curl

/ # ping -c 4 10.0.242.134

/ # curl http://10.0.242.134

o/p:- we are getting web-page.

```
<html>
<head>
-
-
<h1> Welcome to nginx! </h1>
-
-
```

and Accessing the web-Page using curl :-

for getting IP-address I'll take the help of Linux M/c → where I have kubectl. (Here also I have kubectl)

↓

$ kubectl get svc.

o/p: nginx-svc — 10.0.242.134.

ping is not always the command to check whether the connectivity works (or) not

bcoz → ping works on protocol called as ICMP and we have not written any rules around it.

The only rule which we have written is for 80 port

↓

80 port is working.

- For ping to work, your clusters need to open ICMP traffic.