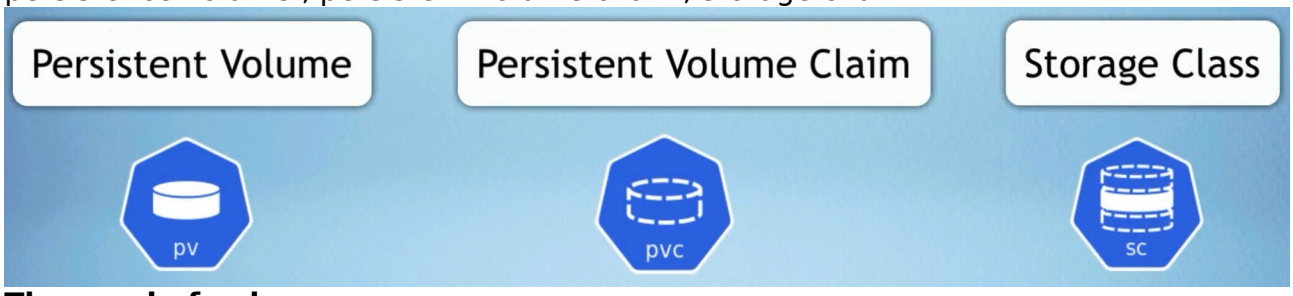
Persisting data with Volumes :

How to persist data in k8s using volumes ?

Three components of k8s storage !

persistence volume , persistent volume chain , storage chain .



The need of volumes :

Consider a case when u are using a mysql data base pod that your application uses . K8s don’t give u data persistence out off the box so when u restart the application or pod the data is lost .

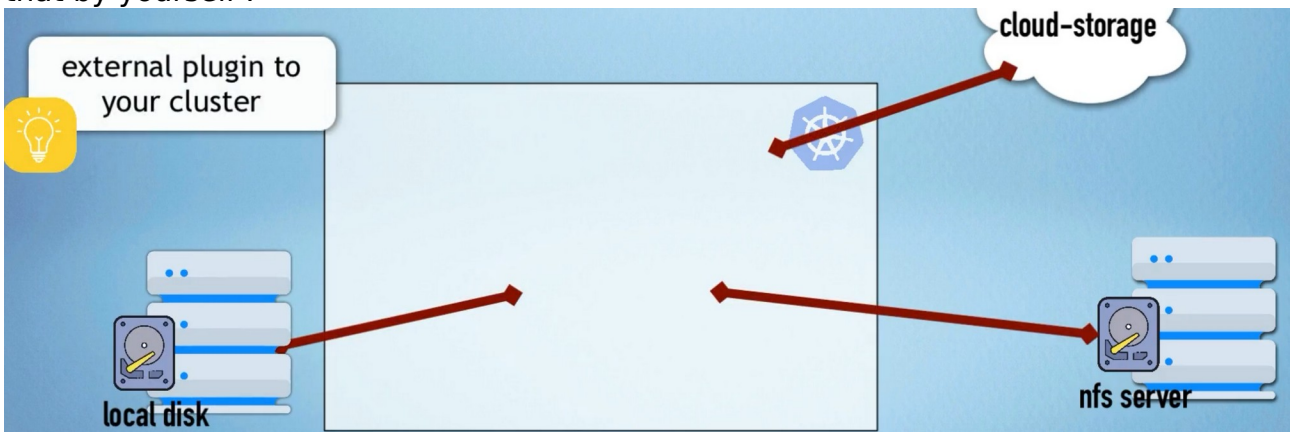
So u need a storage that does not depend on pod life cycle .

My sql will read the existing data from the storage .
How ever u don’t know on which pod the new node restart .
Storage must be available on all nodes . So that the up to data data is there on any node.
Storage needs to survive even if the cluster crashes .

Another Use case for persistent storage .

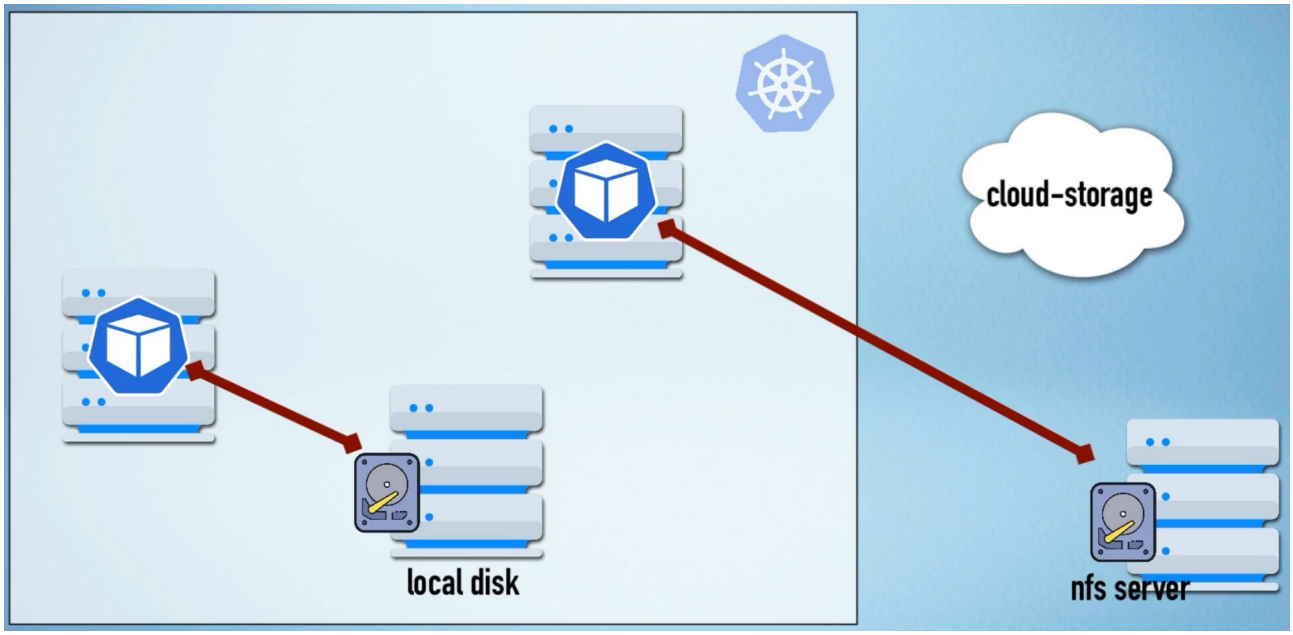
Persistent volume : is like a cluster resource that can be created via YAML file.
Persistent volume needs actual physical storage like local hard drives from the cluster nodes . Or external nfs servers or cloud storage like aws blob storage .

As an admin u decide which type of storage u need . You need to create and manage that by yourself .



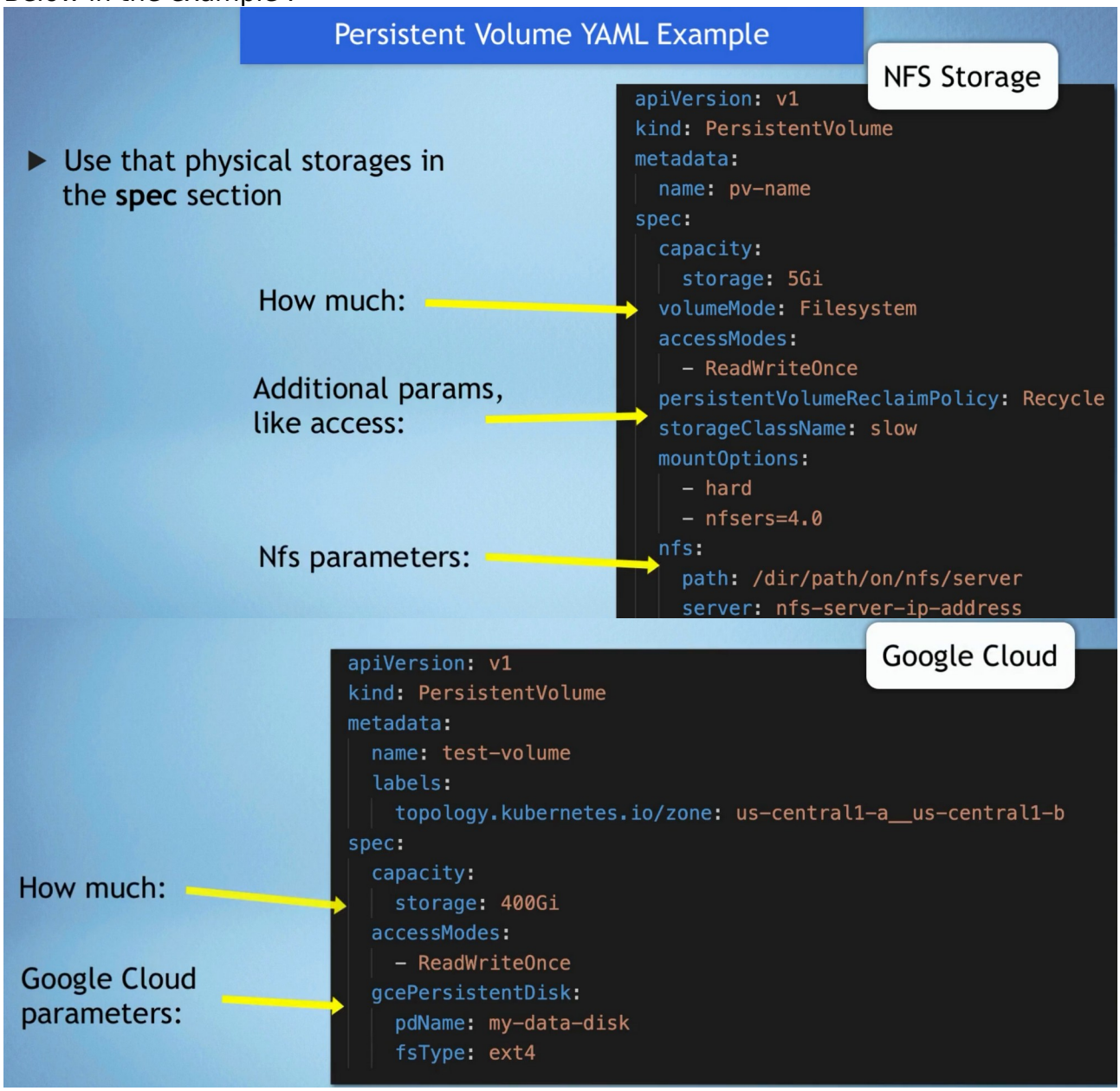
Weather internal or external storage .

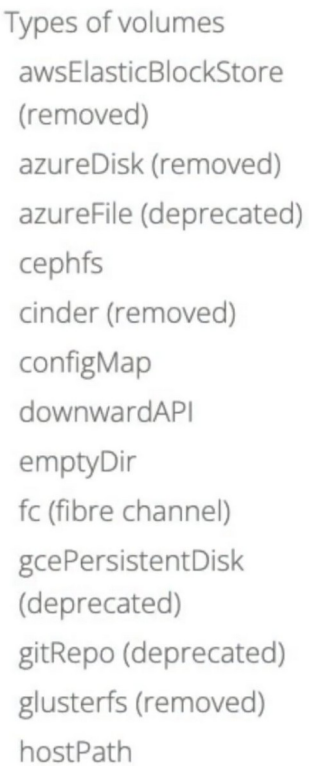
In the cluster multiple pods may use multiple storage location inside or outside the cluster.



Using persistence volumes u can access these storage .
Using spec section u will define which storage backend u need to refer to .

Below in the example :

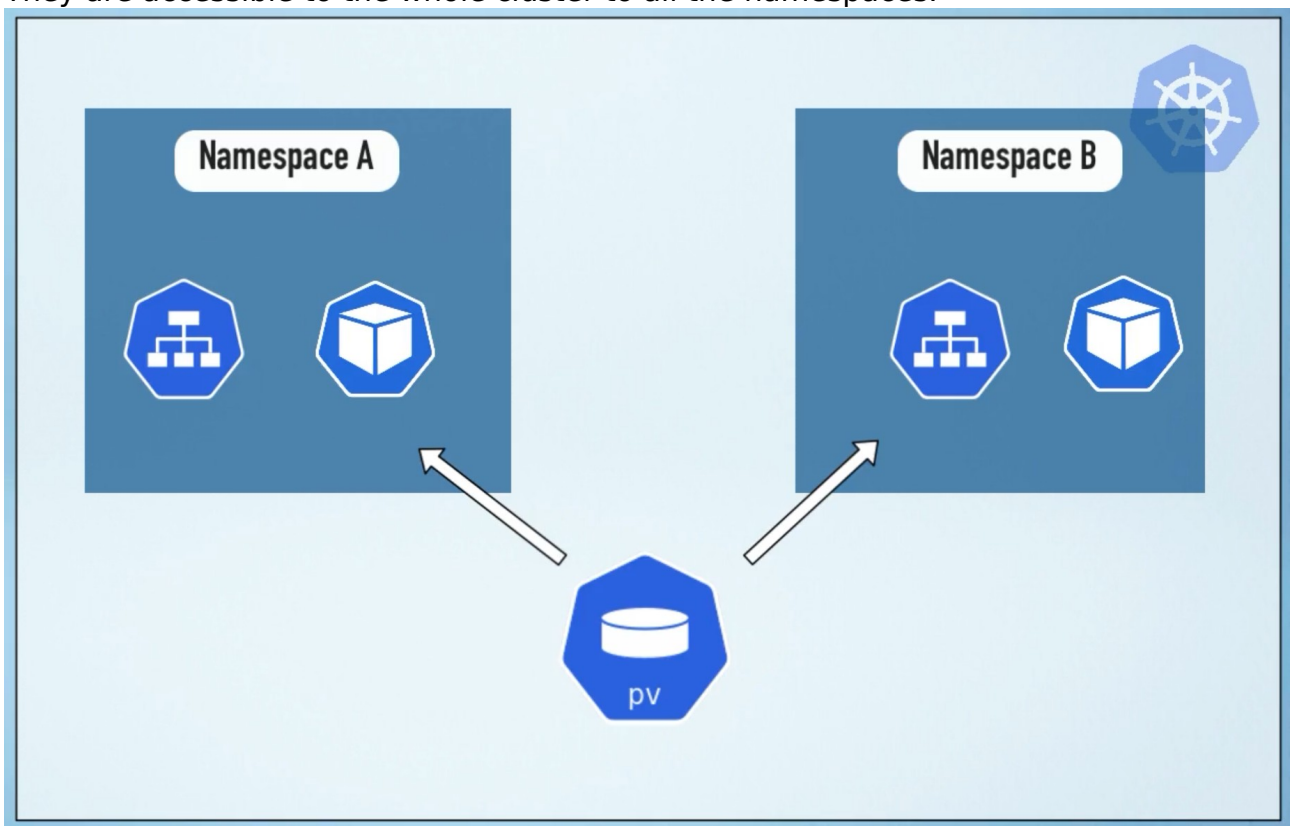




Depending of storage type spec attribute differs. :
Complete list of storage backends supported by k8s :

Persistence volumes are not nameSpaced means they are accessible to the whole Custer. Different from other like pods and services They are not in any namespace .

They are accessible to the whole cluster to all the namespaces.



Local vs remote volumes type :

Each volume type has its own case!
Other while they won't exist .

Local volume type violate 2) and 3) requirement for data persistence :
First : Being tied to one specific node ! Rather then tied to each node equally because u won't know which node will start .
Second : surviving cluster crashes .
There because of these two reasons for DB persistence u always prefer remote storage .

Who creates persistence volumes and when ? :
K8s administrator and k8s user .

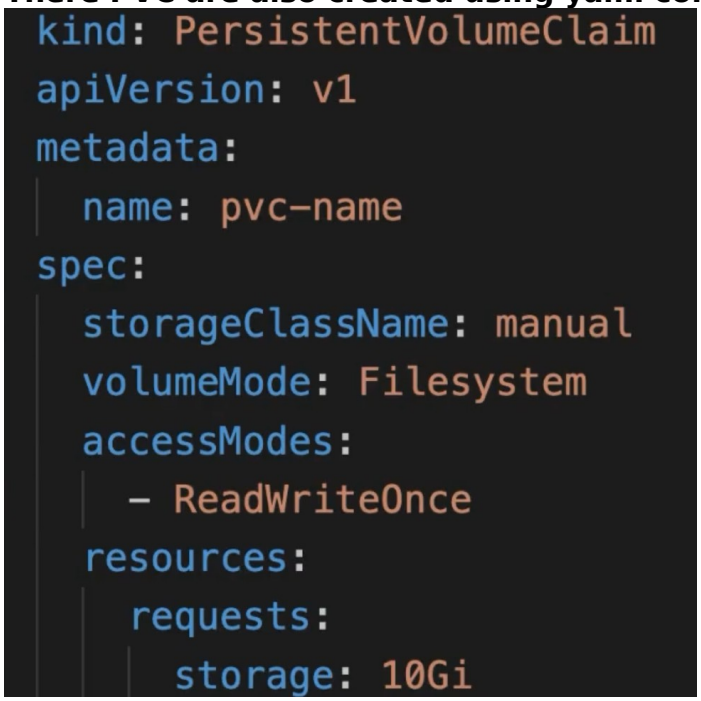
As we know persistent volumes are resources like CPU , so they have to be already there in the cluster the pod that depends on it or uses it .

K8s administrator sets up and maintains the cluster and k8s users deploys application in cluster.
So k8s administrator is the one who configure the storage resource eg: a cloud storage that will be available to the cluster . And create persistence volume components from these storage backends. Based on the information from dev team on what they of storage they need .

But now its the job of developers to explicitly configure the application yaml file to use those persistence volume components .

On other words application has to claim that persistence volume storage .

So u do that using other k8s component **persistence volume claim**
There PVC are also created using yaml configuration .

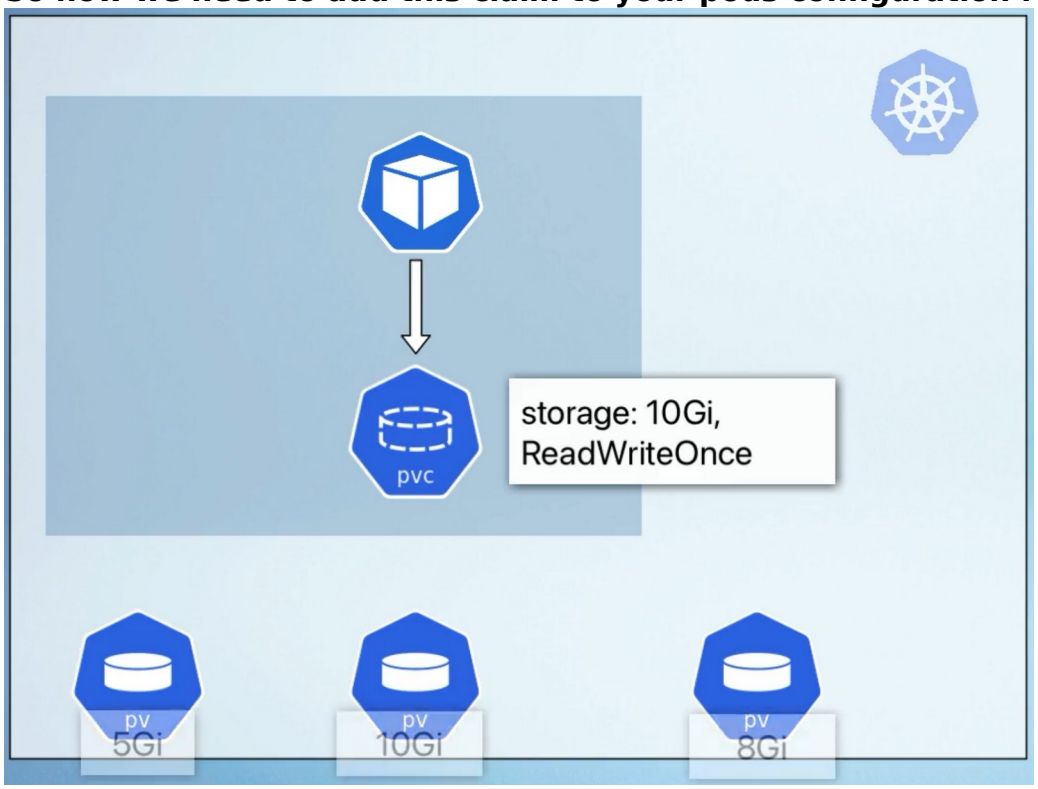


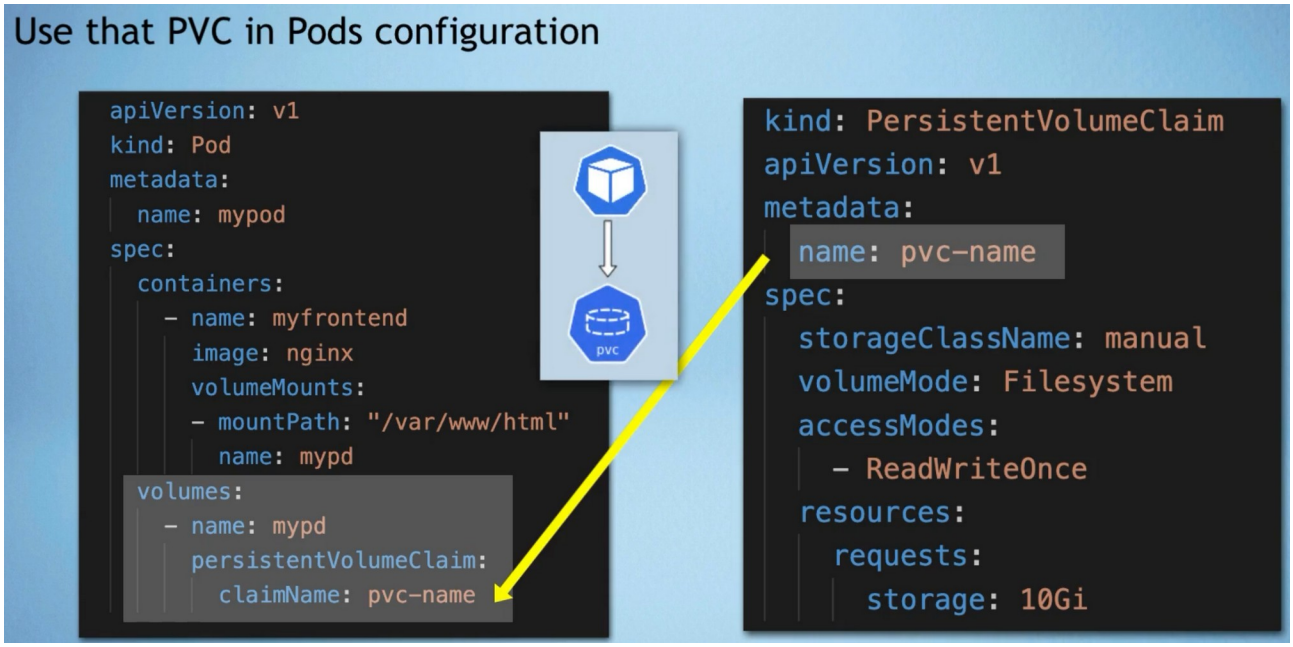
```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Pvc claim the volume which certain volume size of capacity and come additional characteristics like access mode etc.

And what every persistence volume matches this criterial or in other word satisfy this claim will be use for the application .

So now we need to add this claim to your pods configuration .



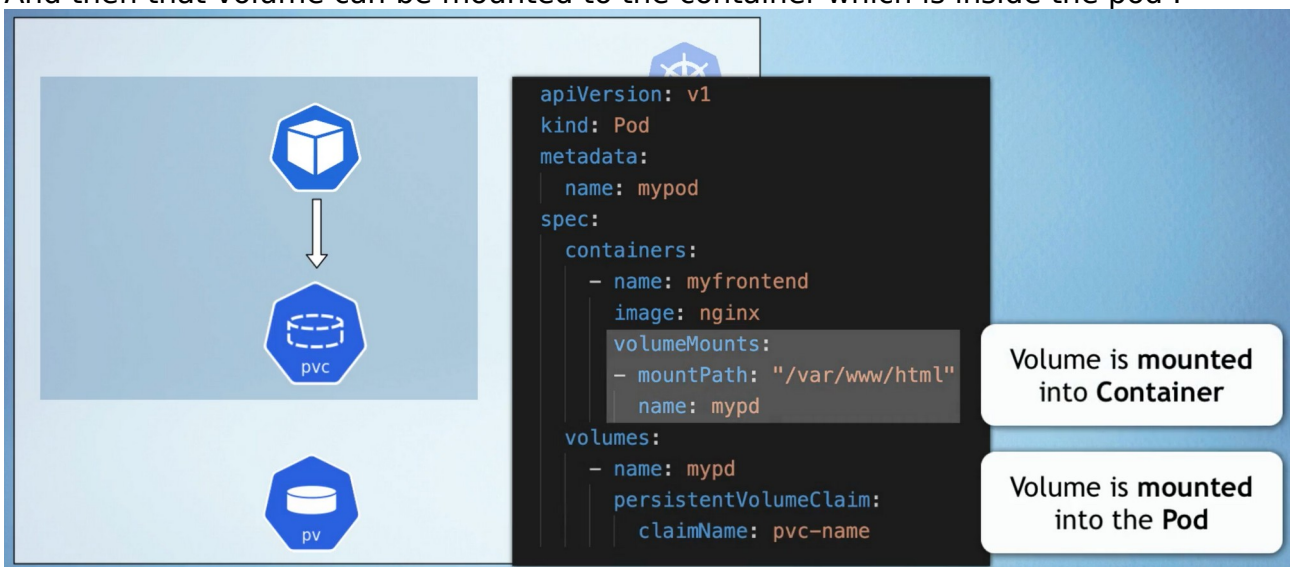


Level of volume abstractions :

Pod request the volume through the PV claim .
Then claim will go and try to find the volume in the cluster that satisfy the claim .
And finally the volume has the actual storage backend.
And thats how the pod will be able to use that actual storage backend in persistence way .

Persistence volumes Claim must be in the same namespace as the pod using the claim and persistence volumes are not namespaceD.

So one the pod finds the matching persistence volume .
One the pod find the volume through the PVC the volume is then mounted into the pod .
And then that volume can be mounted to the container which is inside the pod .



And not finally contender and the application insider the contender can read and write to that storage and now then the pods dies it contender or app will point to the same storage .

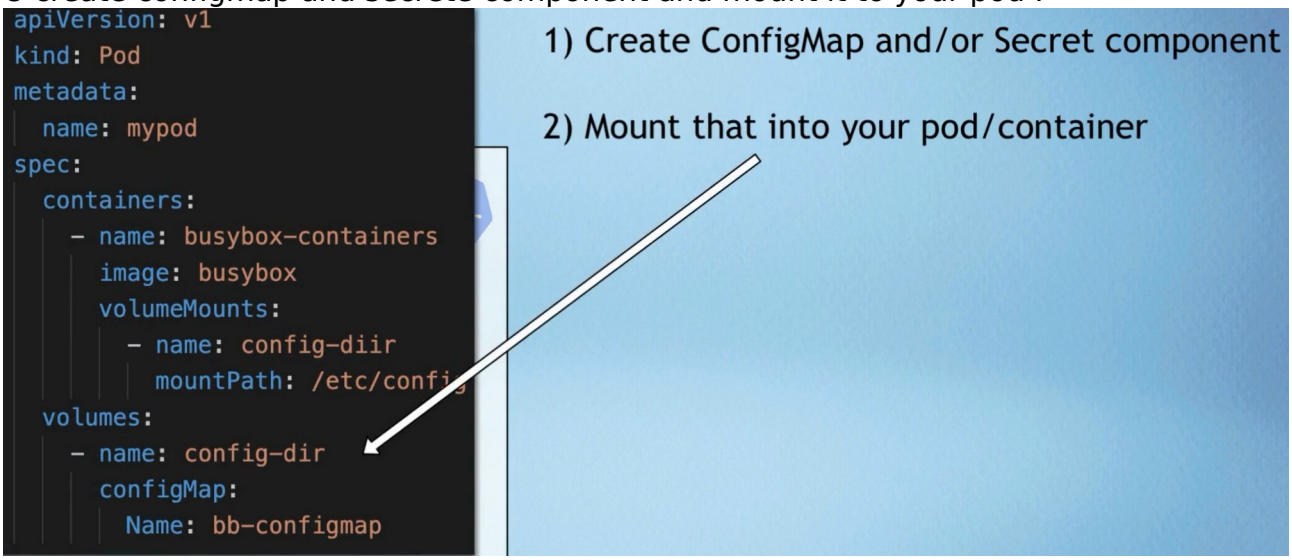
Why so many abstraction levels ?

Admin provisions storage resource
User creates claim to PV .

Benefit of this is as a developer u don’t care about where the actual storage is .

Config map and secret

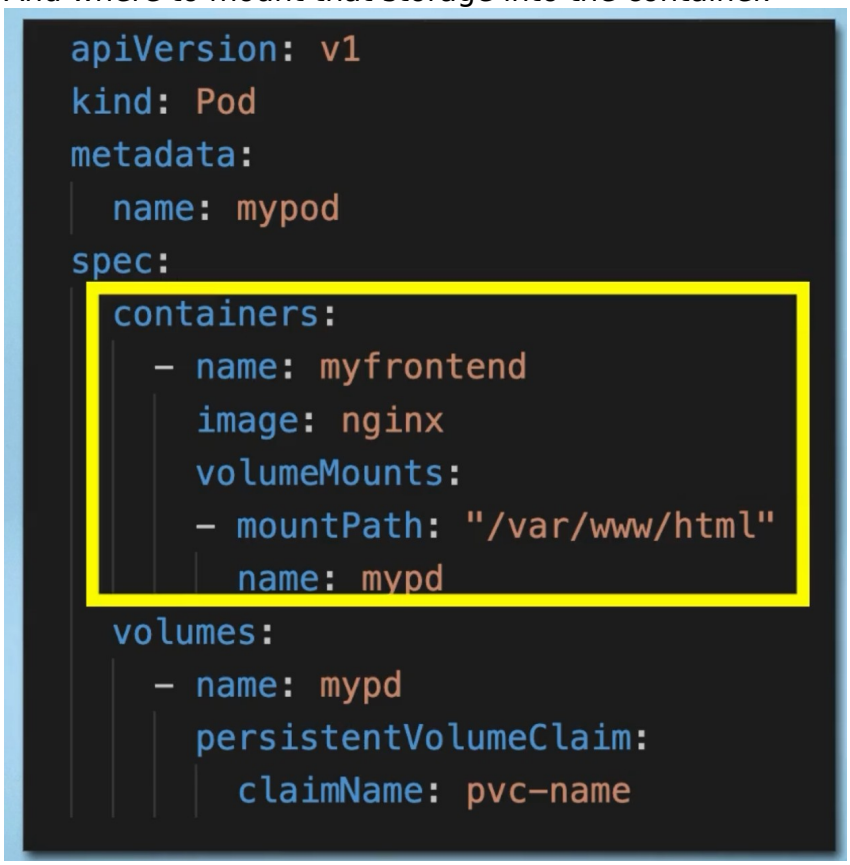
These two are local volumes but they are not create via persistence volume or PVC .
These are managed by k8s it self.
1. U create configmap and secrete component and mount it to your pod .



What we have learned so far :

Volumes is directory with some data
These volumes are accessible in a containers in a pod.
To use a volume pod specific what volume to provide to the pod .

And where to mount that storage into the container.



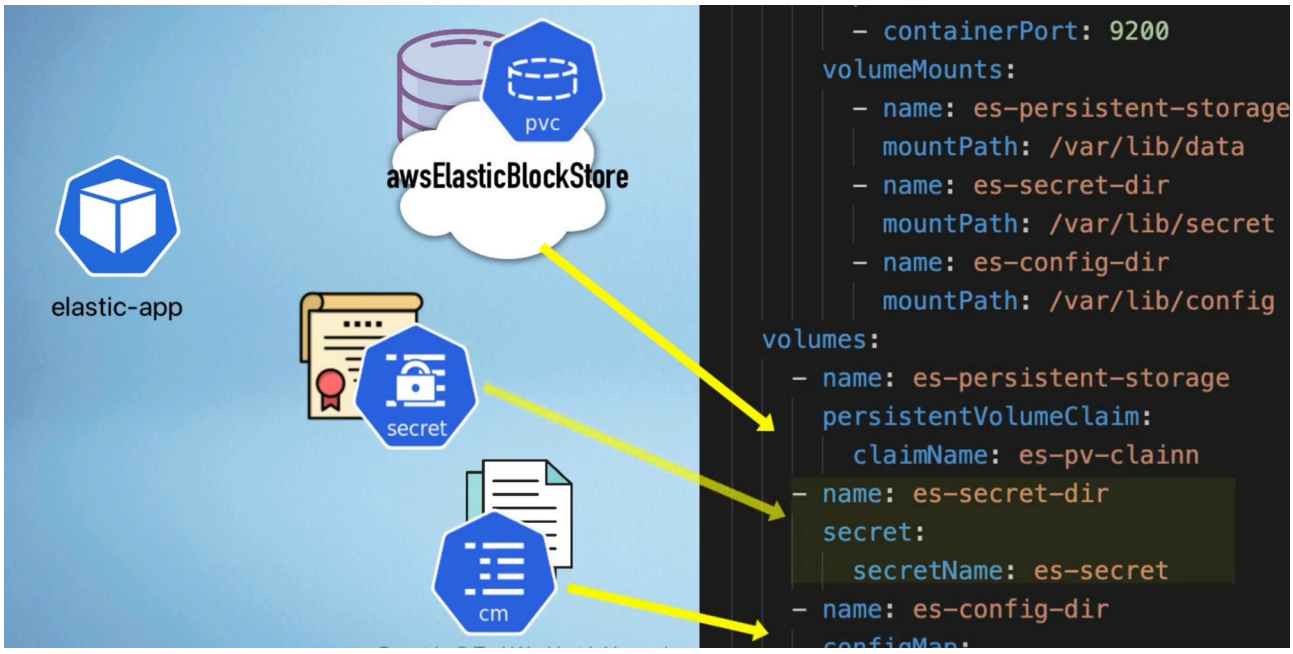
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

If u have multiple containers u can deice which container can use that storage .

Pod can use different volume type simultaneously .

```
volumes:
  - name: es-persistent-storage
    persistentVolumeClaim:
      claimName: es-pv-clainn
  - name: es-secret-dir
    secret:
      secretName: es-secret
  - name: es-config-dir
    configMap:
```

U can list all the volumes that u want to mount to your pod .



Above we see list of volumes we want in the pod to reference to and the volumes mounts where to mount the volume inside the container.

Storage class:

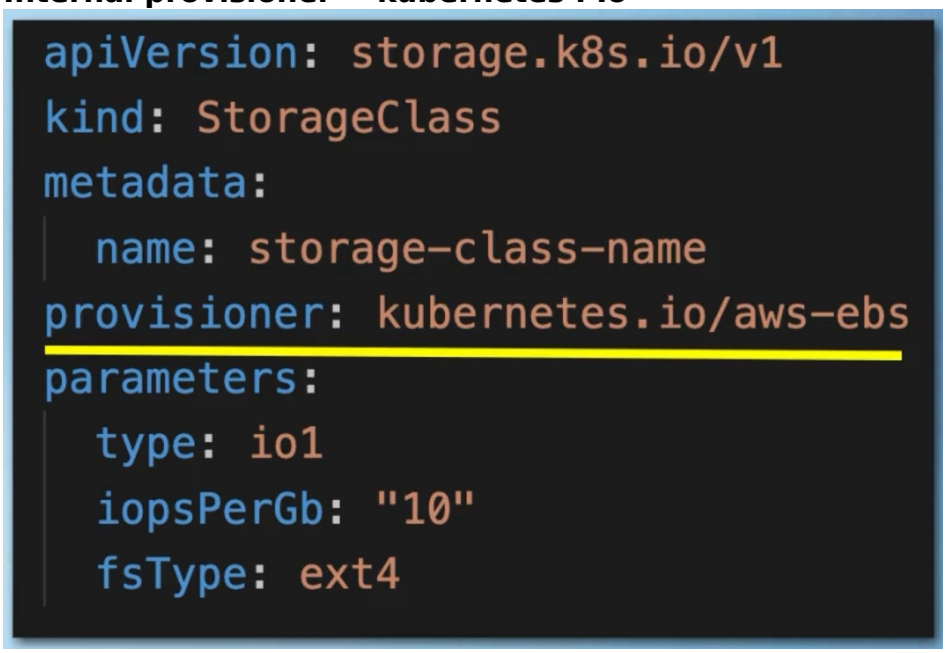
To persist data admin need to configure storage for the cluster create PV and developers can claim using PVC ’s .

Admin has to create all the PV for all the application inside the cluster manually which is time consuming so to make this process more efficient k8s has 3rd component for data persistence called storage class.
Storage class provisions persistent Volumes dynamically … when every PVC claims it .

Storage class is also created using yaml config file.
Storage class create PV dynamically in the background .

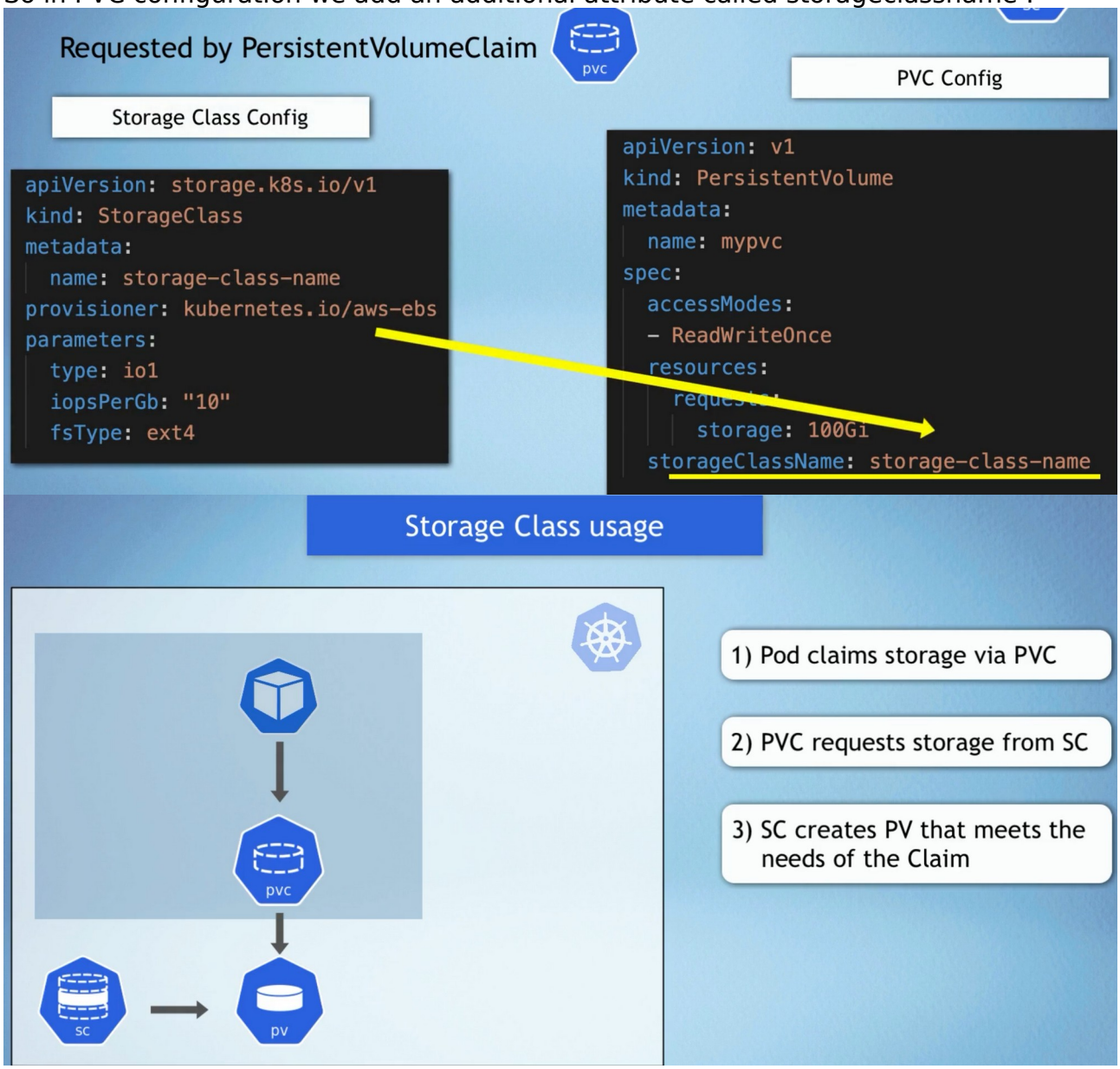
We defined storage backend in the persistent volume component (VPC) now we have to define it in the storage class component and we do that using a **provisioner attribute** .

Each storage has own provisioner
Internal provisioner = kubernetes . Io



We add parameters for storage we want to request for pv .

Same as PV , storage class is requested by persistent volume claim .
So in PVC configuration we add an additional attribute called storageclassname .



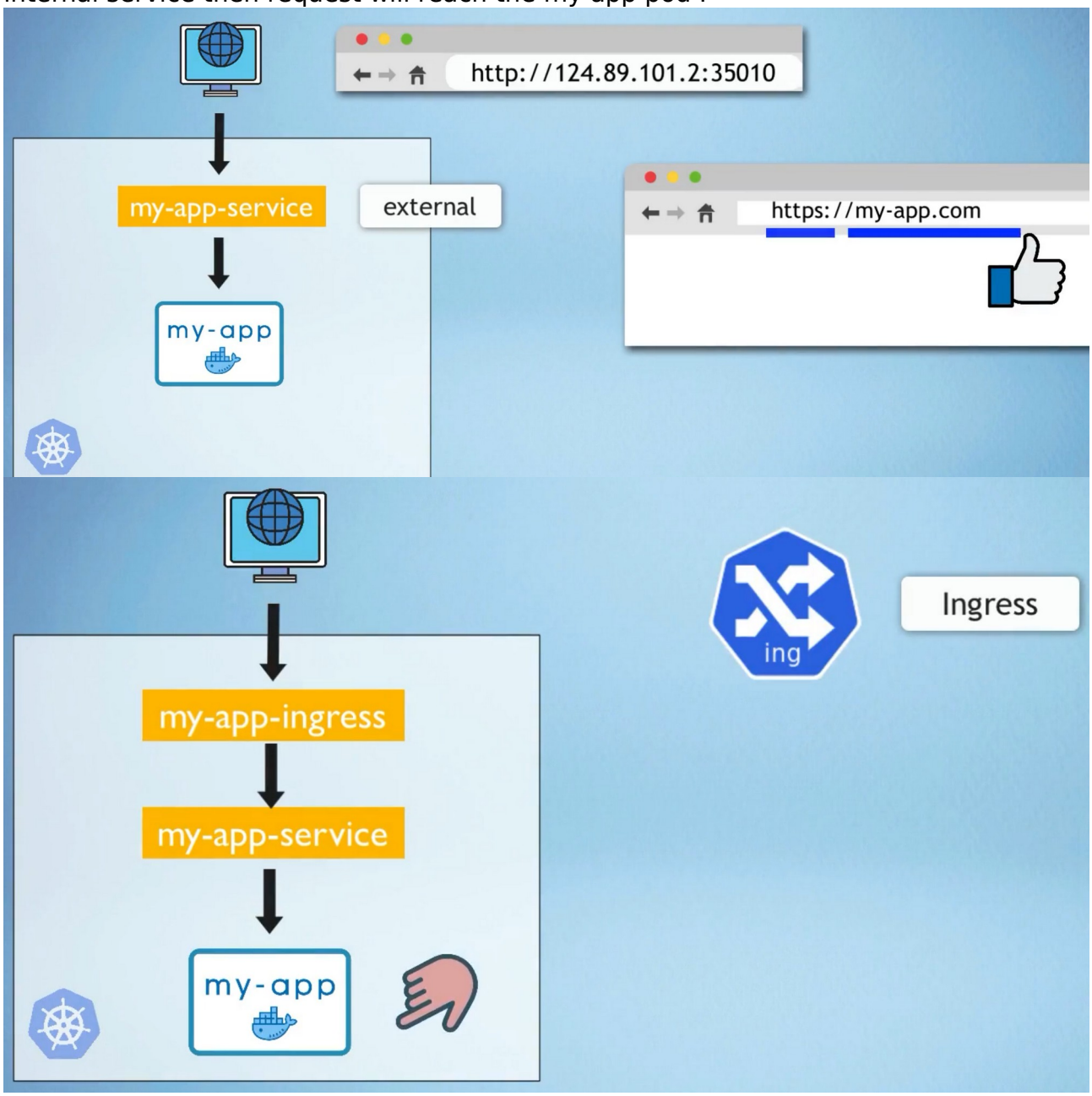
K8s Ingress

External services vs ingress !

Your app should be access able for external request to able to reach your application
One way to do this is u have a external service ip address of the node and port no.
Final product u want is a domain name for your application and a secure connection https .
U can do this via k8s component called ingress .

So now instead of external service u will have a internal service u will not open your app via ip address and port of for application .

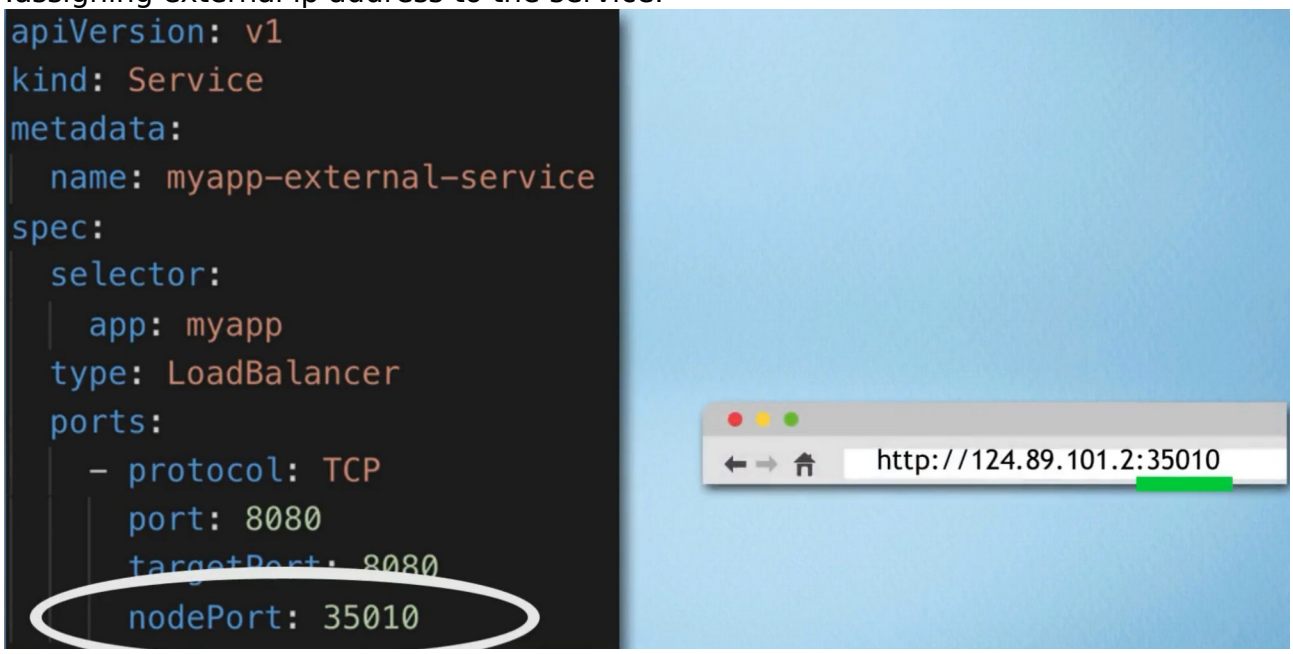
Now if the request comes to the browser it will first reach ingress then redirect to internal service then request will reach the my app pod .



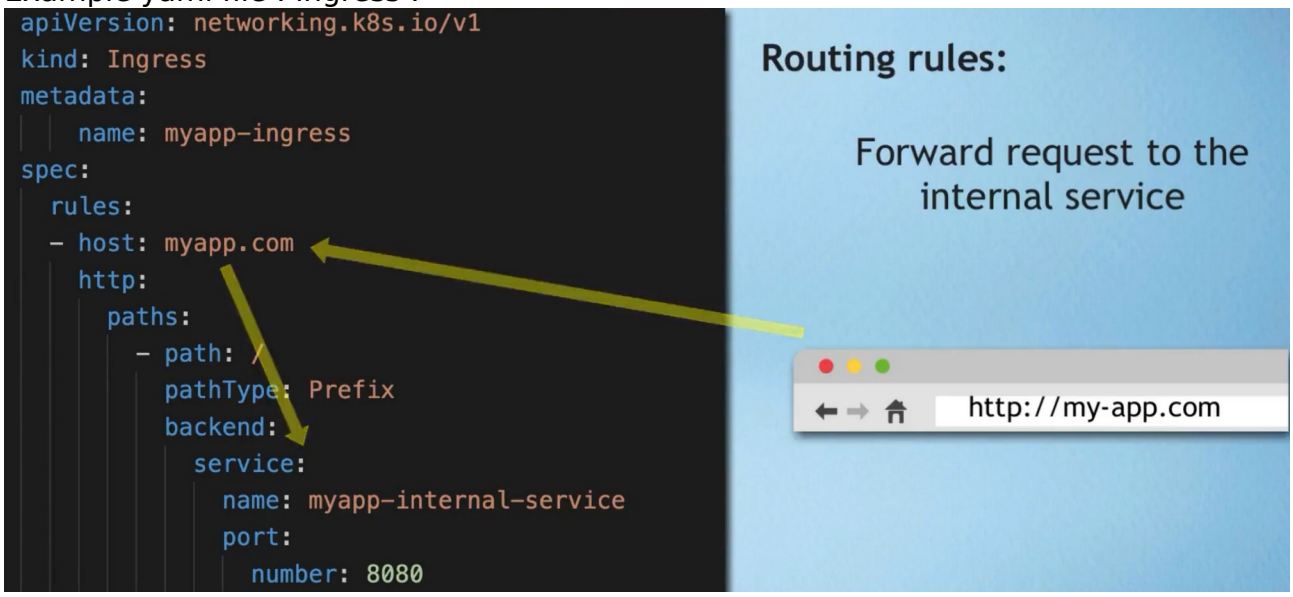
YAML configuration file : external service vs ingress .

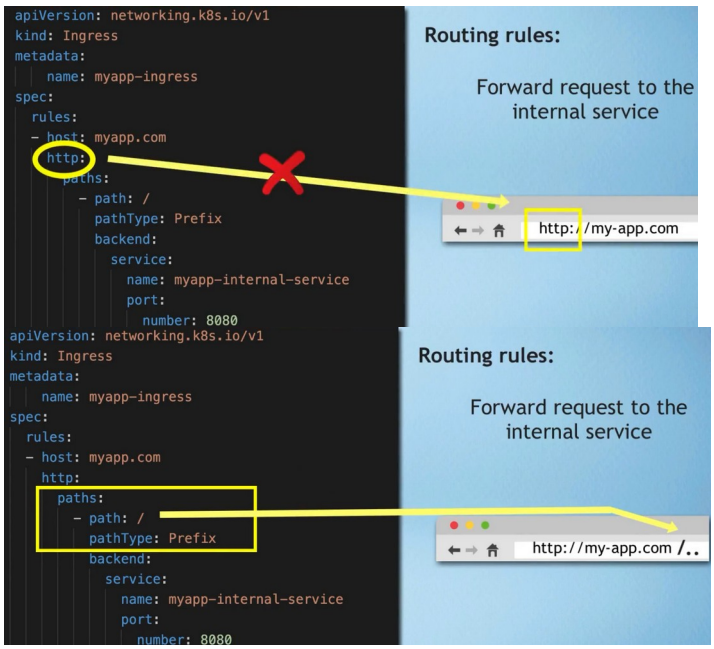
Example yaml file external service :

:assigning external ip address to the service.

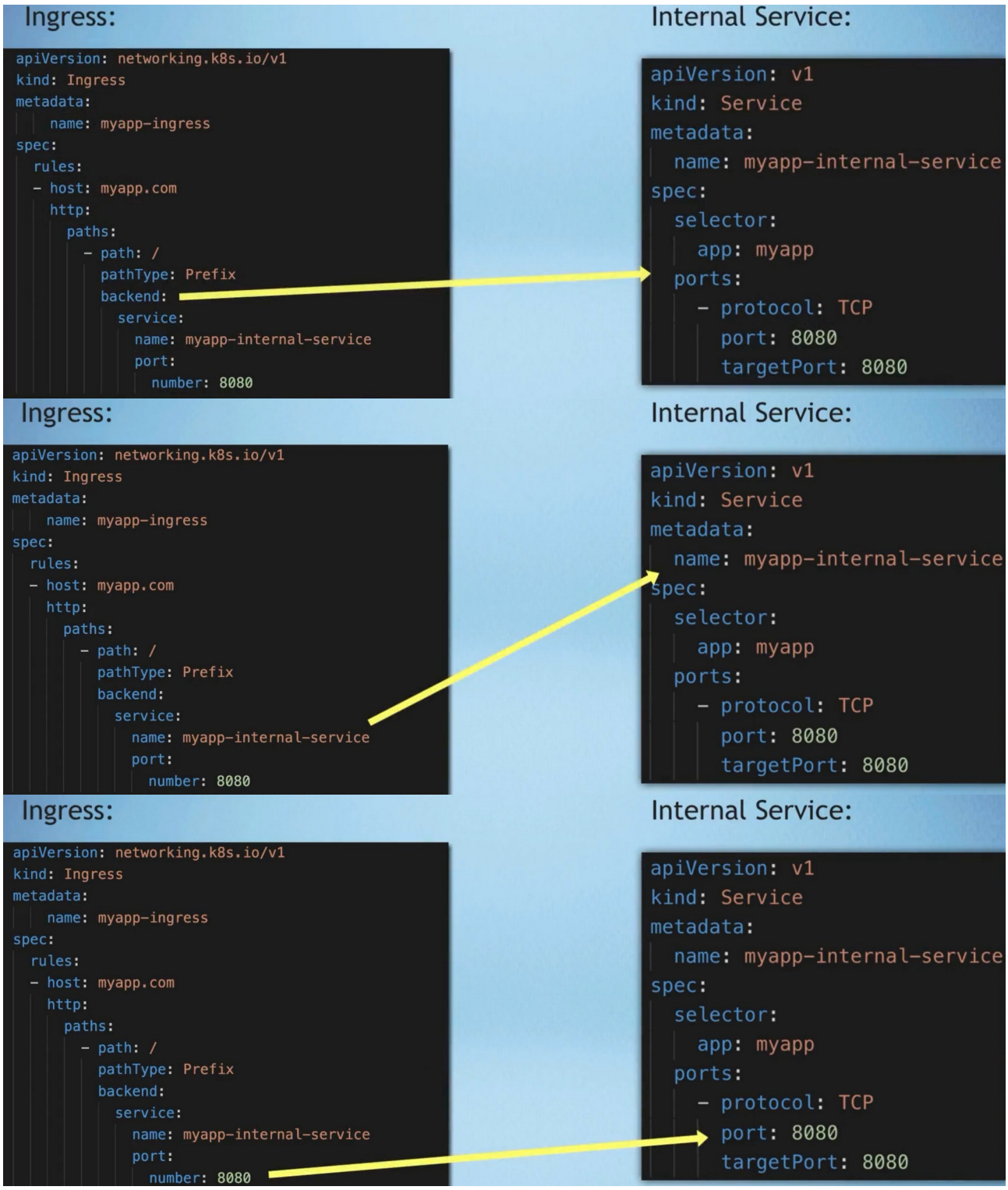


Example yaml file : ingress :





Ingress and internal service configuration



No node_port in internal service where as in external service there is a node_port
Default type is cluster ip of internal service instead of type load balancer in external service .

In ingress HOST : should be a valid domain address.
And u should map that domain name to the ip address, which is a entry point to your k8s cluster.

Eg: if u decide one of the one inside your k8s cluster is a entry point u should map that domain name to the ip address of that node .

How to configure ingress in the cluster ?

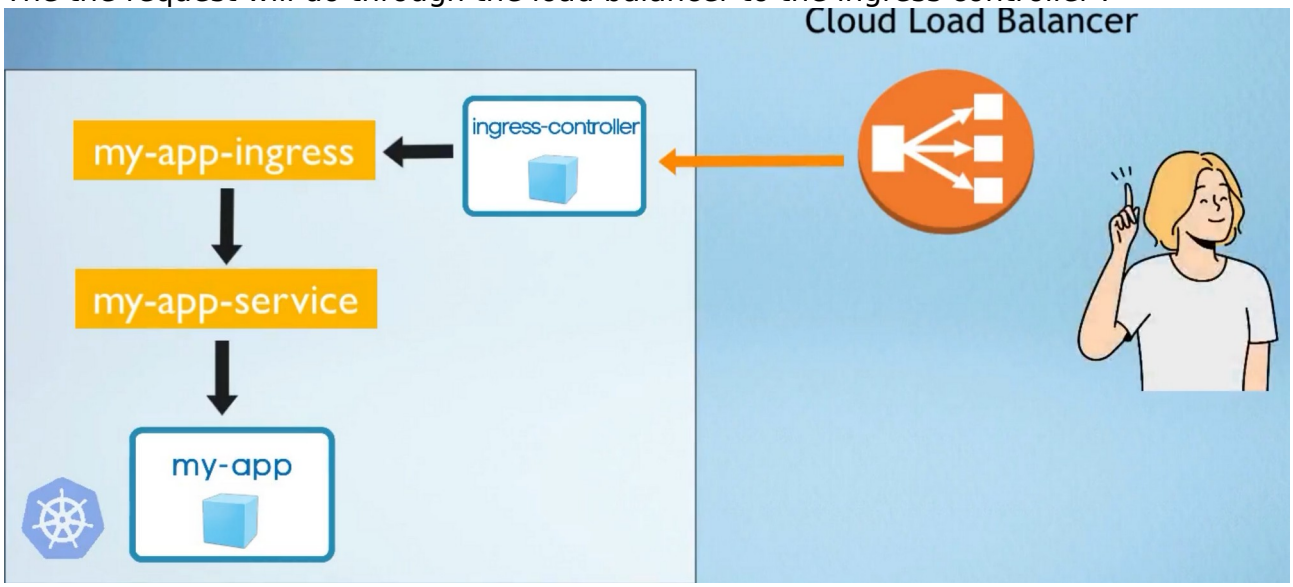
U need an implementation of the ingress and that implementation is called ingress controller .

Ingress controller is a pod of set of pod that is installed in your pod and does the evaluation and processes ingress rules , manage all the redirections .
Ingress controller will be the Entry point to the cluster .

There are many third party implementation of ingress controller ! one of them is k8s Nginx ingress controller .

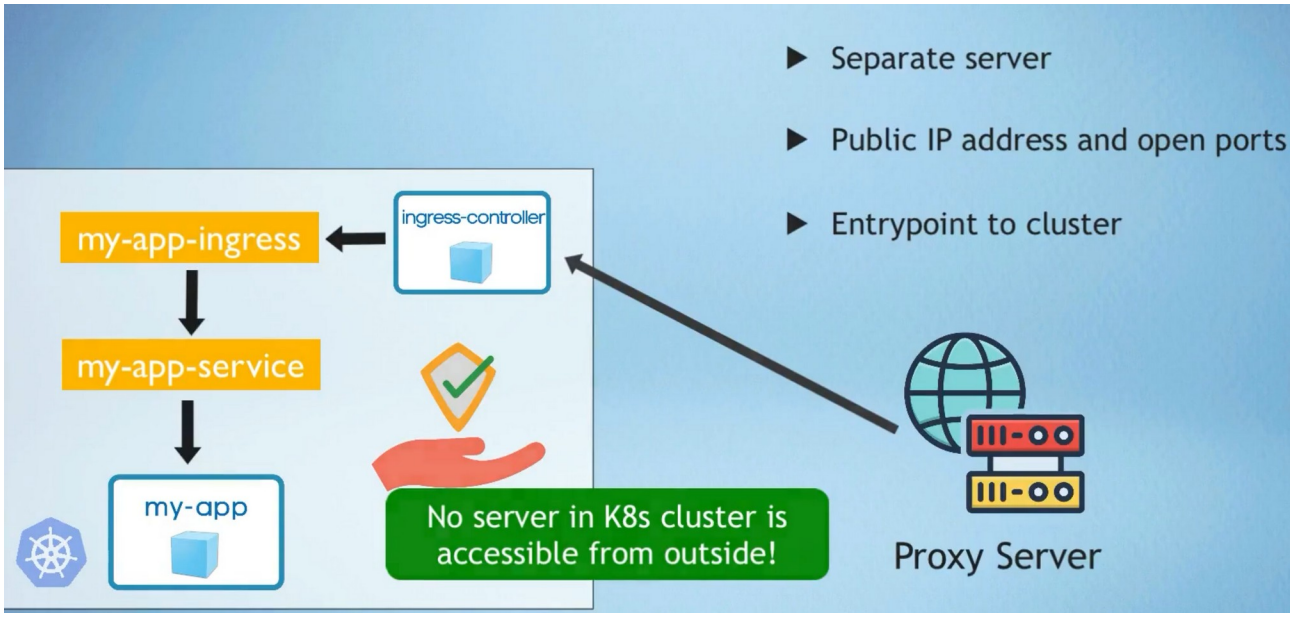
Environment is which your k8s cluster is running :

If u are using a cloud provider like AWS u will have a cloud load balancer service by the AWS itself . U don't have to implement load balancer yourself .
The the request will do through the load balancer to the ingress controller .



On the other side if u r deploying your k8s cluster on a bare metal server then u would have to do that part your self.
Meaning u need to configure some kind of entry point either insider of the cluster of outside as separate server .

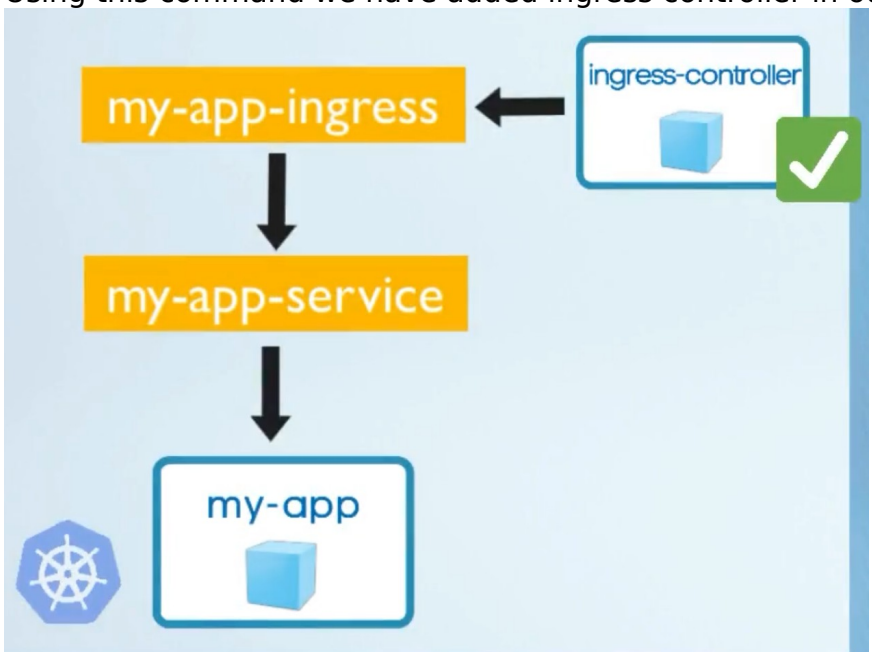
One of the type will be a external proxy server that will be a software or a hardware solution that will that the role of the load balancer .



So all the request will go on the proxy server and then it is redirect to the ingress controller , it will then check the ingress rules . And the whole request forwarding will happen.

Configure ingress in minikube

1. We need to install ingress controller in Minikube .
: commands to use : : minikube addons enable ingress
Using this command we have added ingress controller in our cluster.



Now we will create a ingress rule that a controller can evaluate.
:minikube start
:minikube dashboard -> this command will set up the dashboard in our environment and out up the UI we new browser tab that we can access internally when its ready

List of all the components that I have in my k8s dash board.

```
(base) vishaldwivedi@vishals-MacBook-Air-2 ~ % kubectl get all -n kubernetes-dashboard
NAME                                            READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-b5fc48f67-7tbv5   1/1     Running   0          7m
pod/kubernetes-dashboard-779776cb65-srhkj       1/1     Running   0          7m

NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/dashboard-metrics-scraper   ClusterIP   10.100.4.146     <none>        8000/TCP   7m3s
service/kubernetes-dashboard        ClusterIP   10.108.165.208   <none>        80/TCP     7m3s

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dashboard-metrics-scraper   1/1     1            1           7m3s
deployment.apps/kubernetes-dashboard        1/1     1            1           7m3s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/dashboard-metrics-scraper-b5fc48f67   1         1         1       7m
replicaset.apps/kubernetes-dashboard-779776cb65       1         1         1       7m
```

dashboard-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
spec:
  rules:
  - host: dashboard.com
    http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: kubernetes-dasboard
              port:
                number: 80
```

This is a ingress configuration for forwarding every request that is directed to dashboard . Com to internal Kubernetes-dashboard service

I have not map this dashboard . Com to any ip address .

Apply this ingress rule by :
: kubectl apply -f dashboard-ingress.yaml

```
% kubectl apply -f dashboard-ingress.yaml
ingress.networking.k8s.io/dashboard-ingress created
```

```
% kubectl get ingress -n kubernetes-dashboard
NAME                CLASS   HOSTS           ADDRESS        PORTS   AGE
dashboard-ingress   nginx   dashboard.com   192.168.49.2   80      2m17s
```

This above ip address will map to dashboard . Com host

```
% sudo vim /etc/hosts
Password:
```

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       dashboard.com
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
# Added by Docker Desktop
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

Add local host 127.0.0.1 to dashboard . Com means the request will come to the minikube cluster the request will be handed over to the ingress controller and the ingress controller will go and evaluate the rule that we define and finally forward that request to that service .

Finally we need to run command: minikube tunnel

```
% minikube tunnel
✅  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...

🏃  Starting tunnel for service mongo-express-service.
❗  The service/ingress dashboard-ingress requires privileged ports to be exposed: [80 443]
🔑  sudo permission will be asked for it.
🏃  Starting tunnel for service dashboard-ingress.
```

Now I can one dashboard . Com on my browser.

Minikube tunnel is a process that is gonna map this local address to the service for ingress .

Ingress default backend :

```
% kubectl describe ingress dashboard-ingress -n kubernetes-dashboard
Name:             dashboard-ingress
Labels:           <none>
Namespace:        kubernetes-dashboard
Address:          192.168.49.2
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host           Path  Backends
  ----           ----  --------
  dashboard.com
                 /   kubernetes-dashboard:80 (10.244.0.39:9090)
Annotations:     <none>
Events:
  Type    Reason  Age                  From                      Message
  ----    ------  ----                 ----                      -------
  Normal  Sync    5m24s (x2 over 6m12s)  nginx-ingress-controller  Scheduled for sync
```

If u can not find that page u need a error message for the users .
For that u need to create a internal service Default backend .

```
% kubectl describe ingress -n kubernetes-dashboard
Name:             dashboard-ingress
Labels:           <none>
Namespace:        kubernetes-dashboard
Address:          192.168.49.2
Ingress Class:    nginx
Default backend:  kubernetes-dashboard:80 (10.244.0.49:90
Rules:
  Host           Path  Backends
  ----           ----  --------
  dashboard.com
                 /   kubernetes-dashboard:80 (10.244.0.49
Annotations:     <none>
```

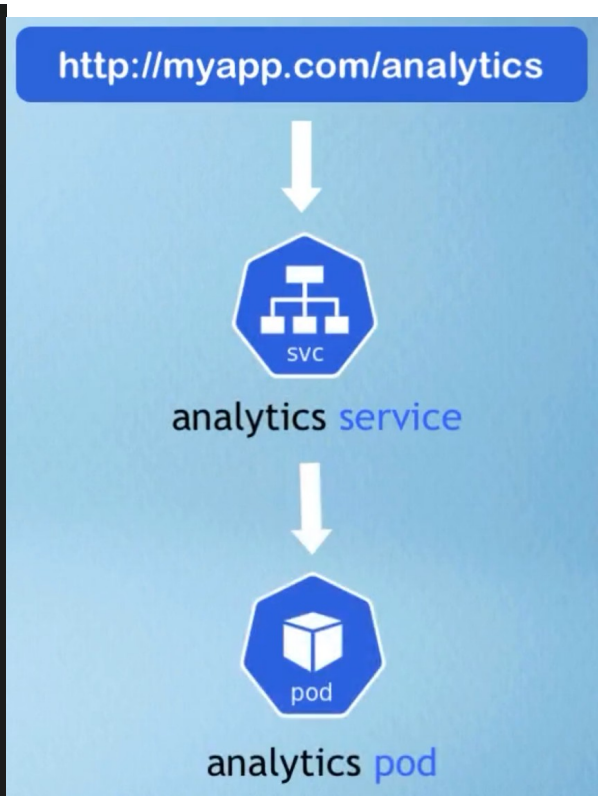
```
spec:
  defaultBackend:
    service:
      name: kubernetes-dashboard
      port:
        number: 80
```

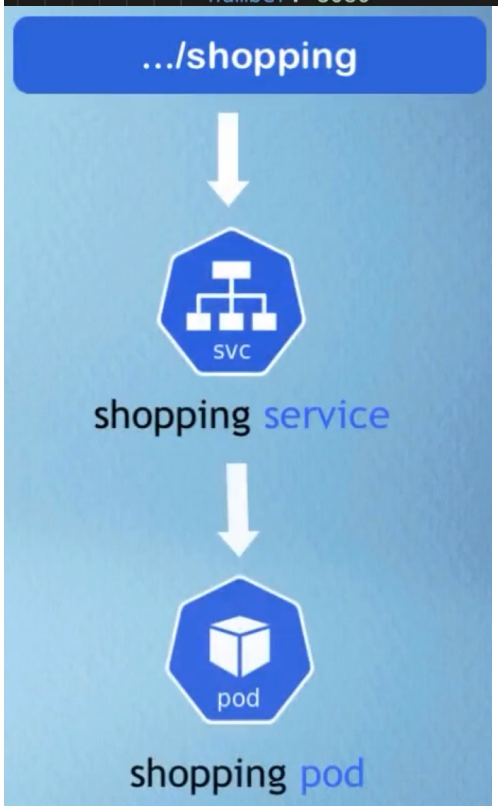
More use cases of ingress

1. Defining multiple paths form same host .

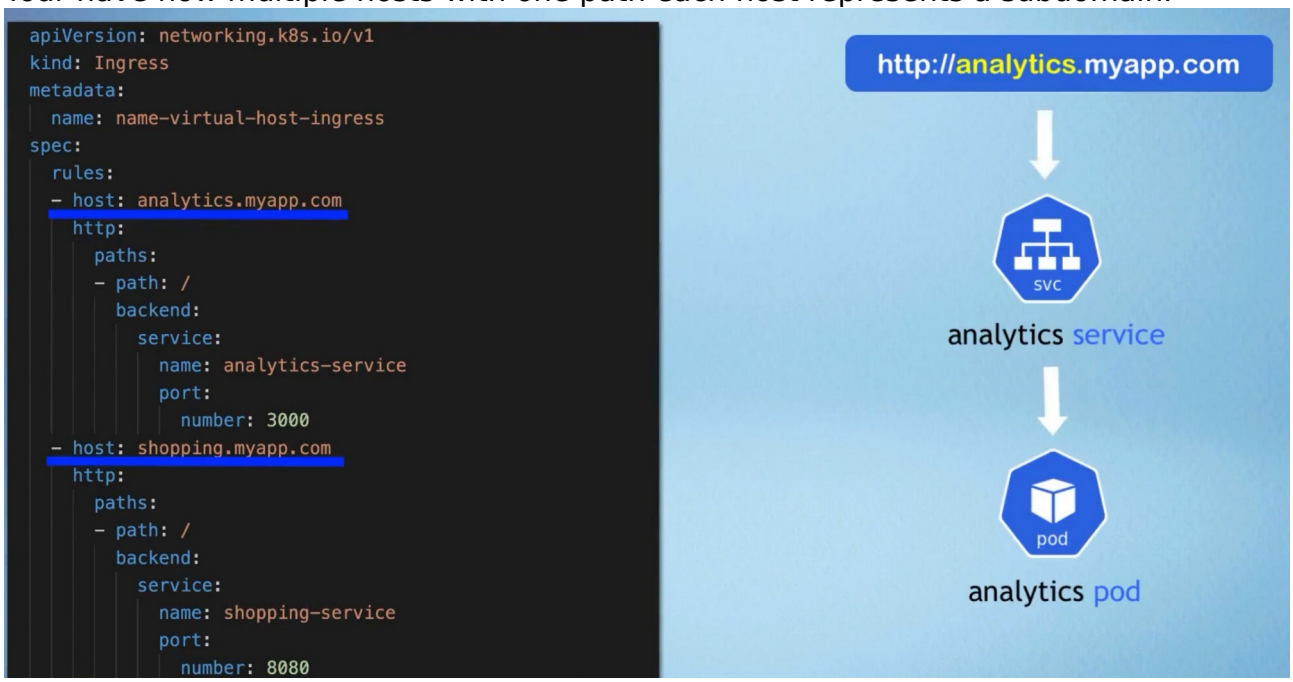
Eg: use case is google has one domain but have multiple services .

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp.com
    http:
      paths:
      - path: /analytics
        backend:
          service:
            name: analytics-service
            port:
              number: 3000
      - path: /shopping
        backend:
          service:
            name: shopping-service
            port:
              number: 8080
```





2. Use case is multiple subDomains or domains.
Your have now multiple hosts with one path each host represents a subdomain.



Configuration TLS certification - https://

Under spec: u need to define t l s : and a secretName: