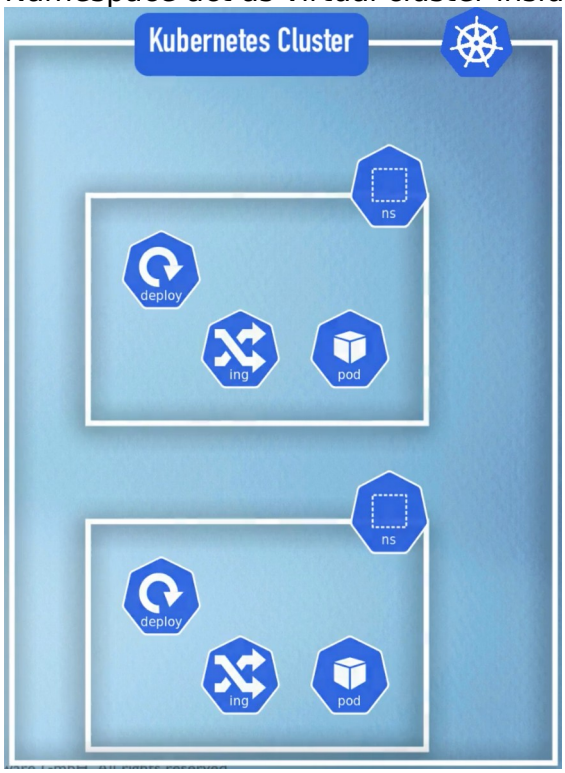
Kubernetes NameSpace :

What is a NameSpace : In k8s cluster u organise recourse in namespaces .
Namespace act as virtual cluster inside a cluster .




```
% kubectl get namespace
NAME              STATUS   AGE
default           Active   7d2h
kube-node-lease   Active   7d2h
kube-public       Active   7d2h
kube-system       Active   7d2h
```

When u create a cluster k8s by default gives u namespaces .
4 namespaces per Default .
: command line -> kublctl get namespace
1. Kube-system : not for your use , do not create of modify in kube-system
The components that are deployed in the name space are the system processes eg: control plane or kubectl processes .

2.kube-public : public accessible data , A configMap, which contains cluster information .

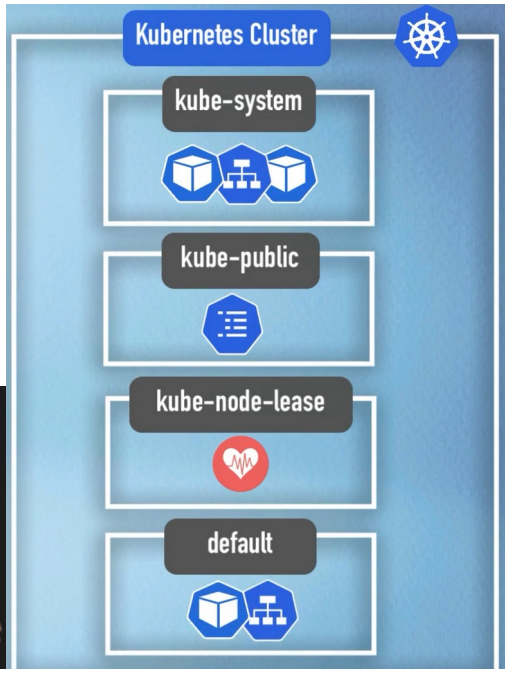
```
% kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:50836
CoreDNS is running at https://127.0.0.1:50836/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

3. Kube-node-lease : Heart beats of nodes , Each node has associated lease object in namespace. It determines the availability of a node .
4. Default namespace : resources u create are located here ,
U can create your own namespace using : kubectl create namespace vishal_namespace : command.

```
% kubectl create namespace my-namespace
namespace/my-namespace created
```

Other way to create a namespace is using a configuration file .

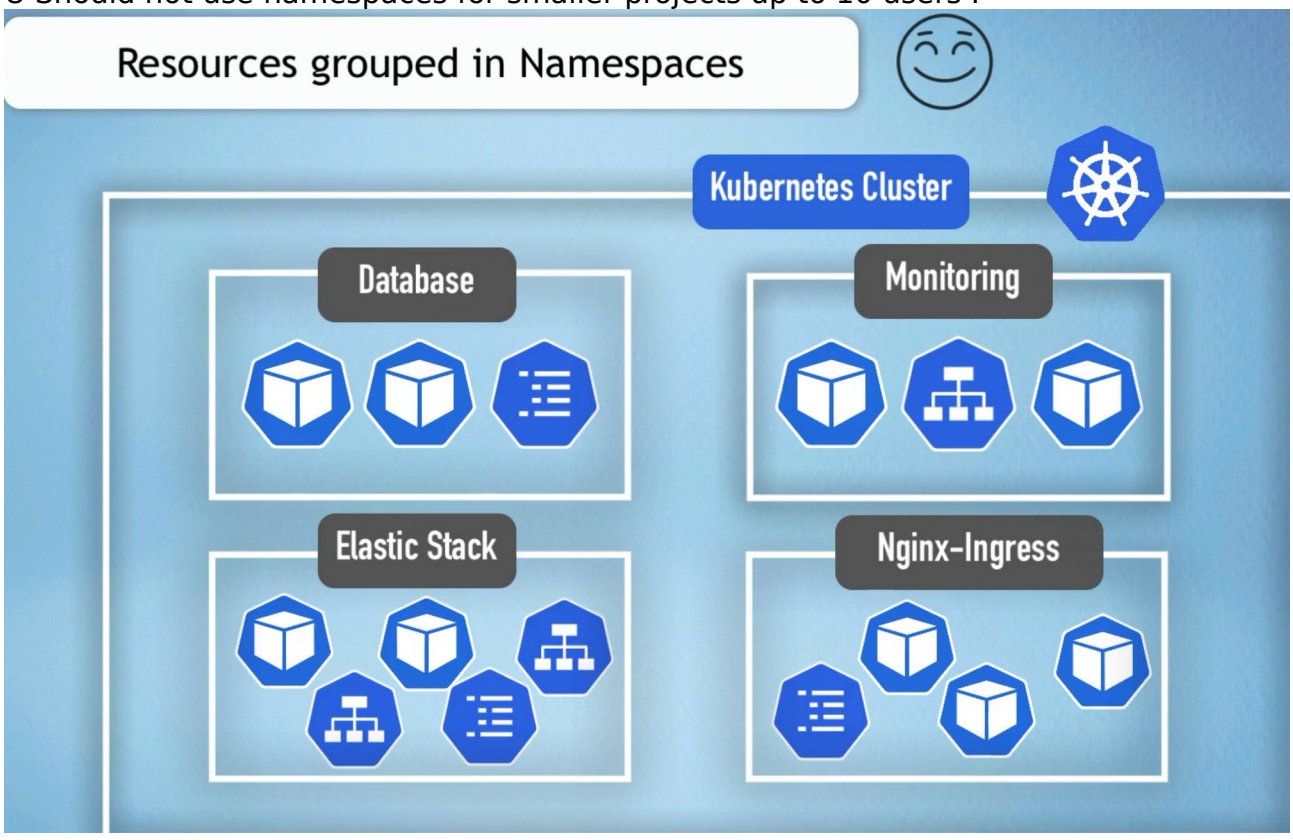
```
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
```



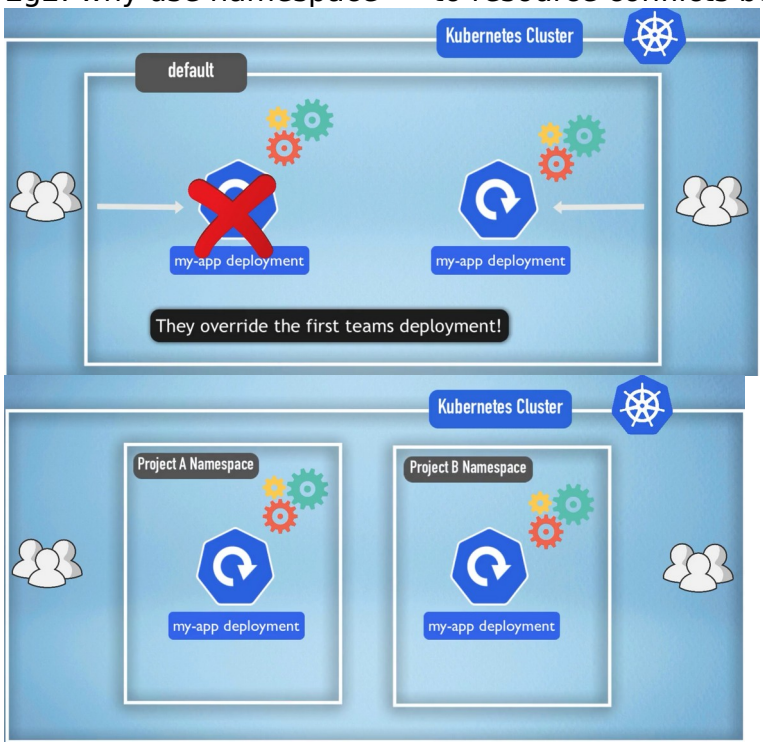
Why use Namespaces ?

If u have every thing in one namespace it will be very difficult to mange every thing different resource of different components .
Eg: u can group different resources on based on namespace like database name space contains all the resource for database.
Or monitions namespace where u deploy monotoning tools etc.

U Should not use namespaces for smaller projects up to 10 users .

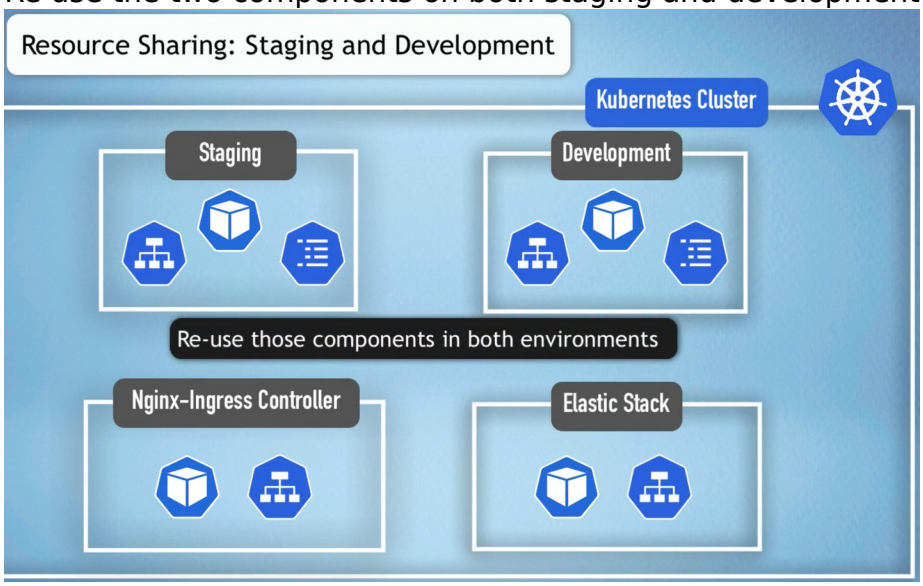


Eg2: why use namespace -> to resource conflicts btw teams .

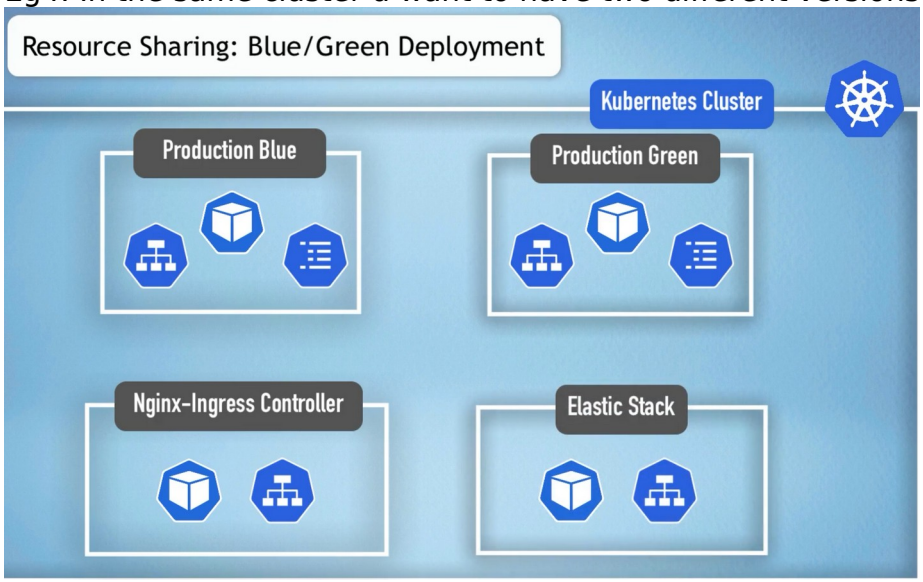


Eg3 : why use namespace -> resource sharing .

If u want to host same staging and development environment on the same cluster .
Re use the two components on both staging and development environment .

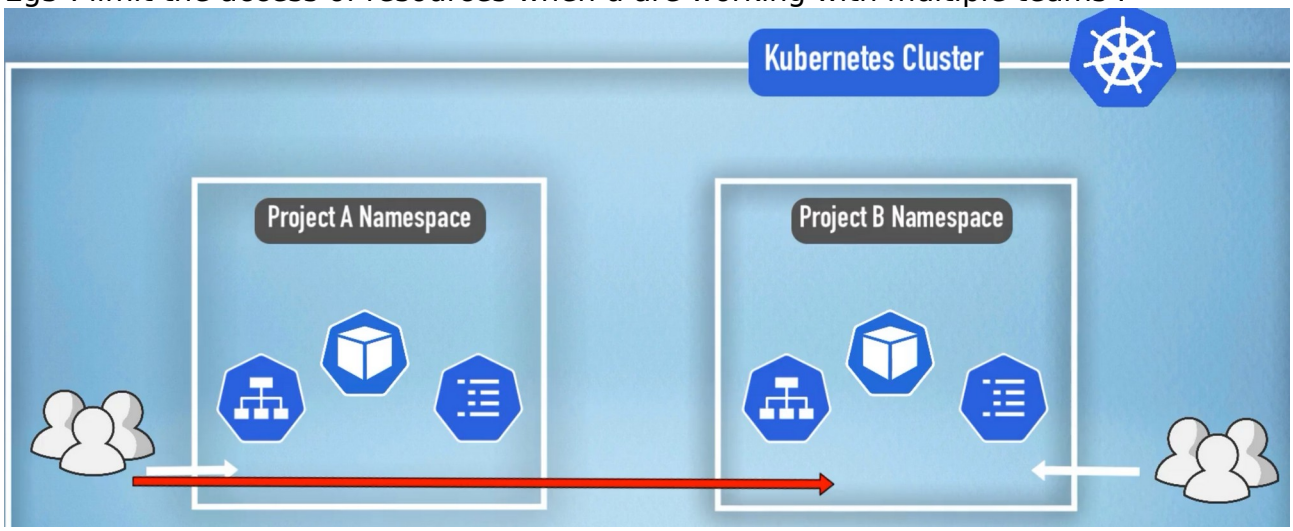


Eg4: In the same cluster u want to have two different versions of production .

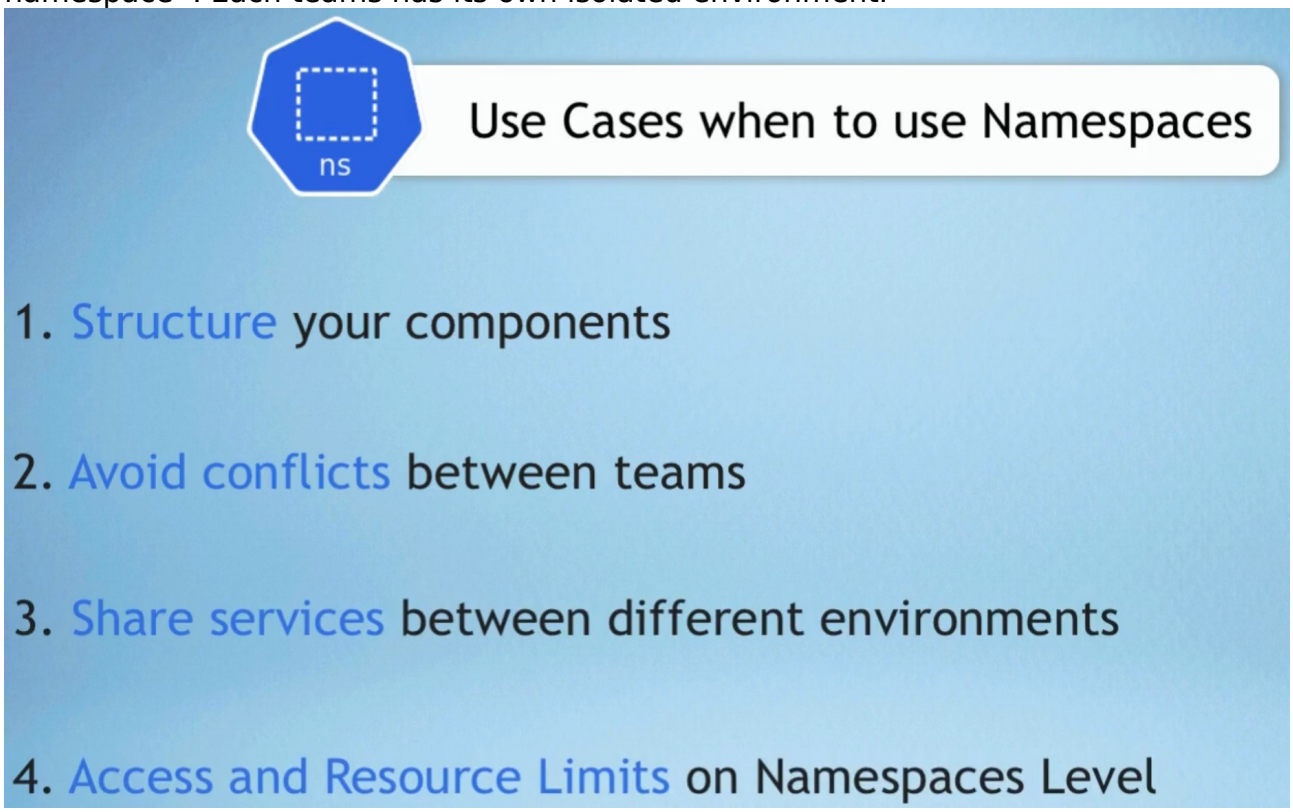


so no need of separate cluster.

Eg5 : limit the access of resources when u are working with multiple teams .

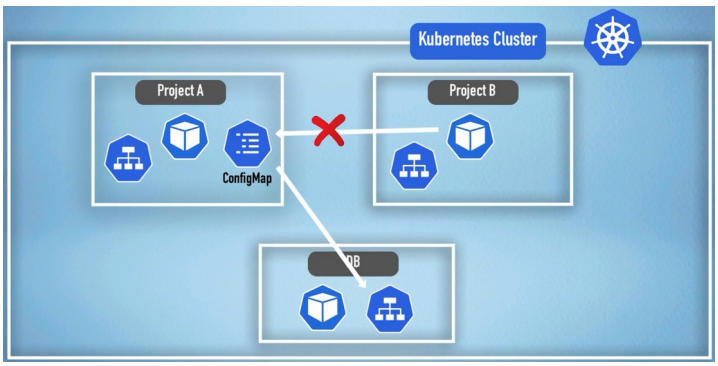


Limit the resources that each name space consumes eg: cpu , ram , storage per namespace . Each teams has its own isolated environment.

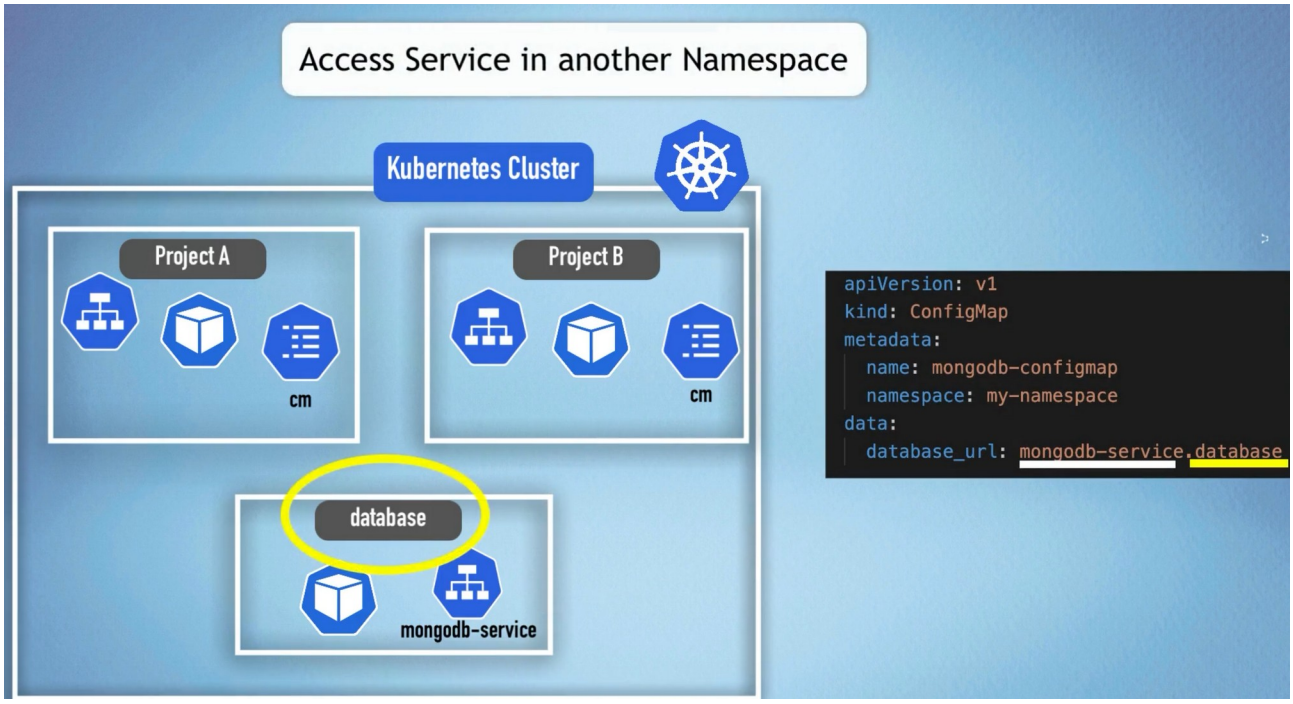


What are characteristics of Namespace:

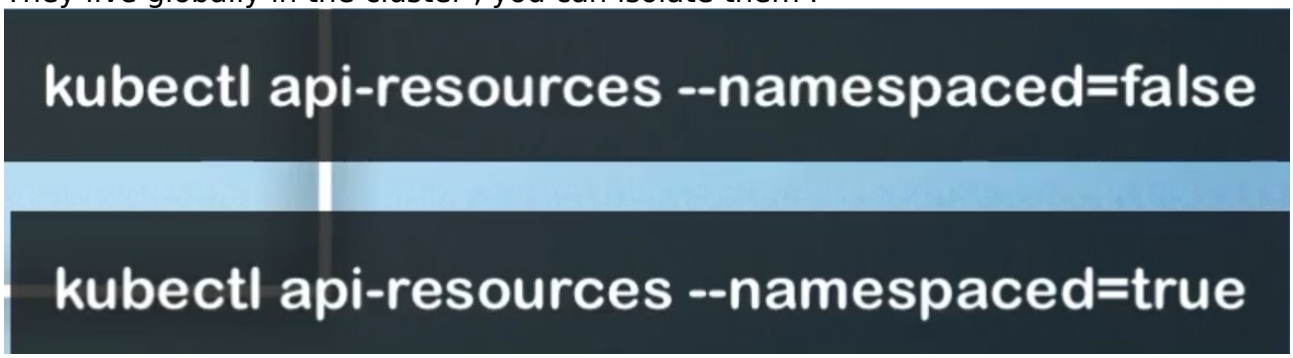
1. U cant access most of the resources from the other namespace .



Similar to secretes or configMap if project A and project B both are using same DB the confimap and secrets should be present on both project A and project B .
However the resource that u cant share is SERVICE .

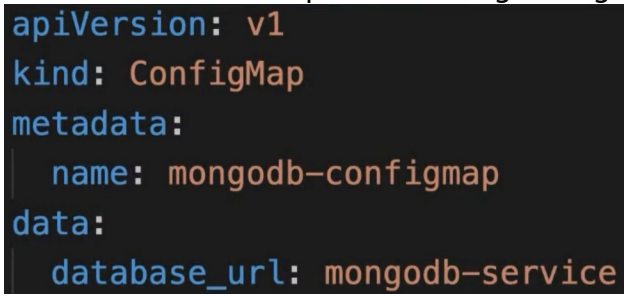


Component that cant be created in a namespace : are VOLUME AND NODE .
They live globally in the cluster , you can isolate them .



Create components in namespace :

We have create components using configuration file ->

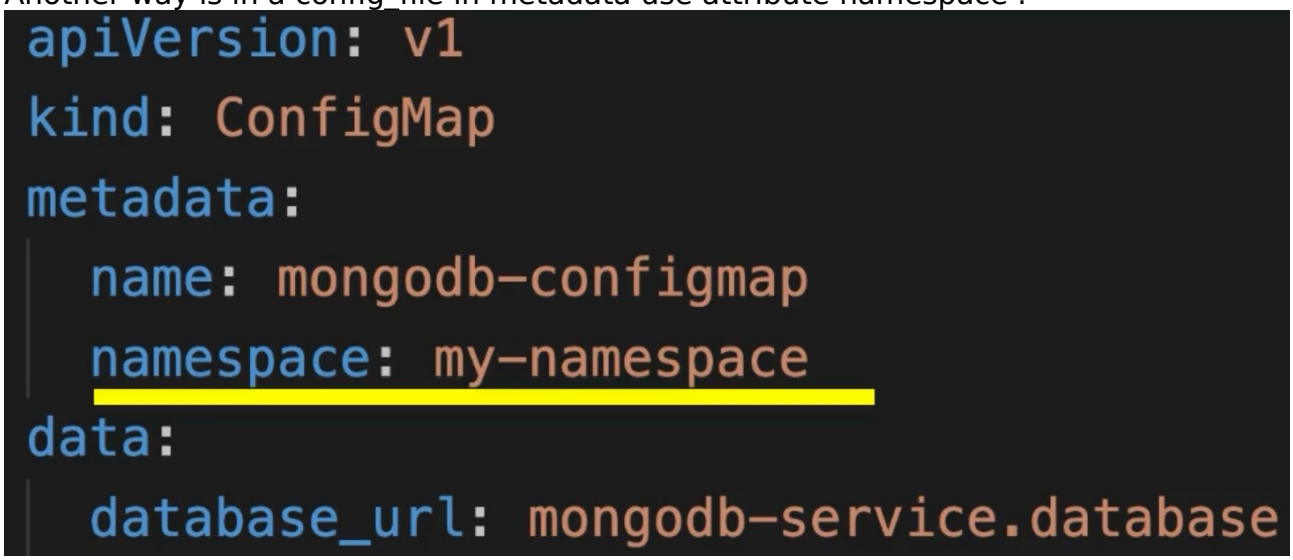


No namespace defined above , by default components are created in default namespace .

This below command will create config_map in myNamespace .

```
% kubectl apply  -f mongo-configmap.yaml --namespace=my-namespace
```

Another way is in a config_file in metadata use attribute namespace .

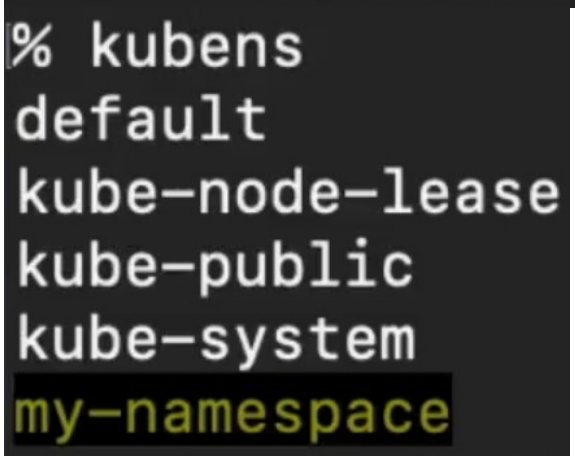


Use configuration file over kubectl command to add a component to a namespace.

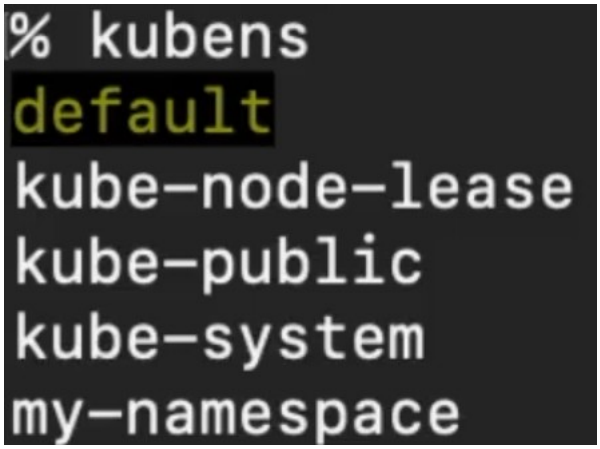
Change active namespace:

U can choose the default namespace to the other namespace .

```
% kubectl config set-context --current --namespace=my-namespace
```



: brew install kubectx -> this will install kubens .



Kubens helps easy to switch btw namespaces and see .

Kubernetes services :

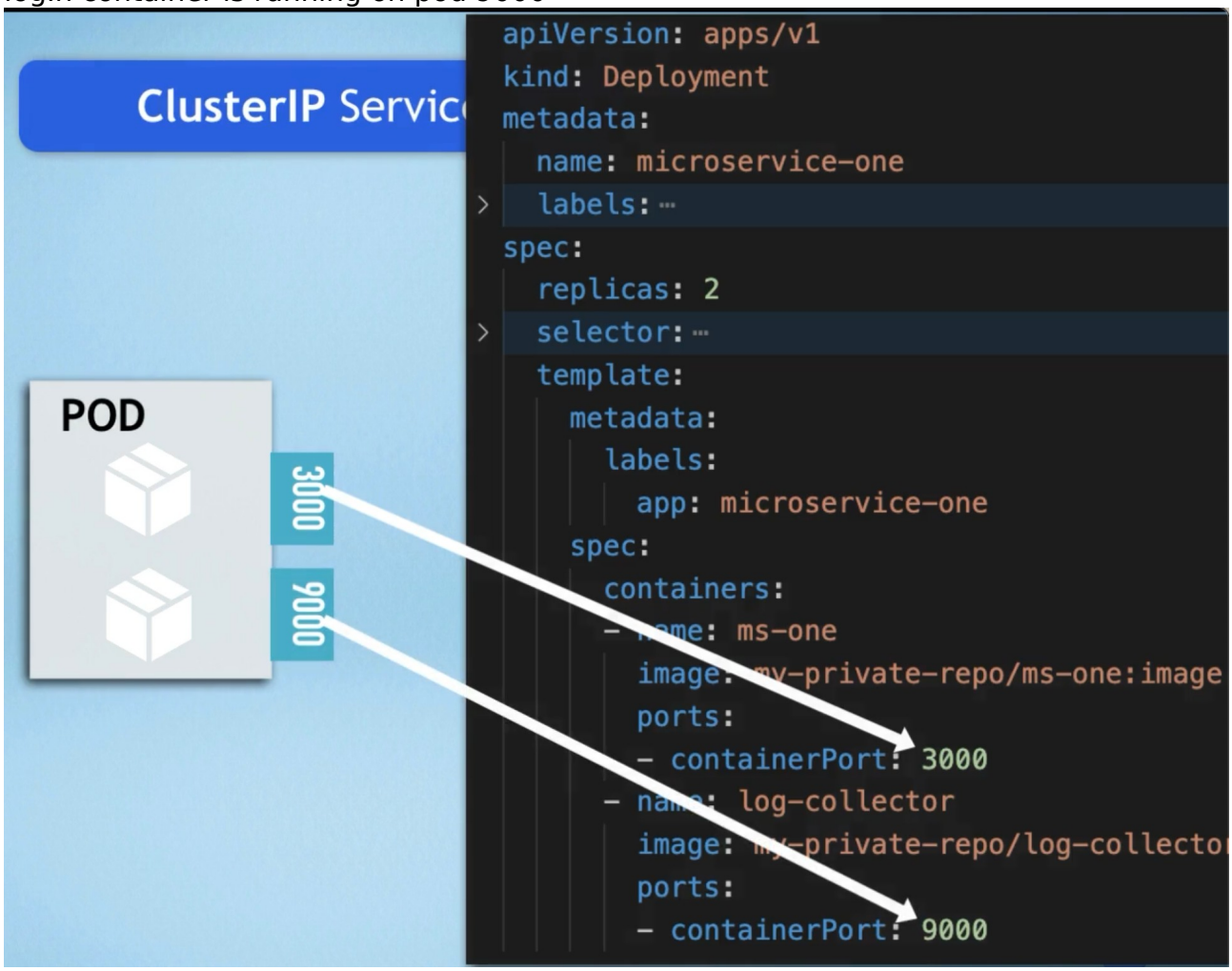
What is a service and why do we need it?
In k8s cluster each pod gets its own ip address but pods in k8s are ephemeral means they are destroyed frequently .
And when old one dies and new one restarted it gets new ip address this make u adjust pod ip address eery time u aecrest it .
With service u get a advantage of setting a static stable ip address which will remain there even if the pod dies .

Service also provide load balancing client will sent request to your cluser and services will forward it to the different pods of your micro_services application .
Loose coupling communication with in the cluster .
Components or pods insider the cluster also from external searches eg: browser request or DB communication .

1 . Cluster IP services .

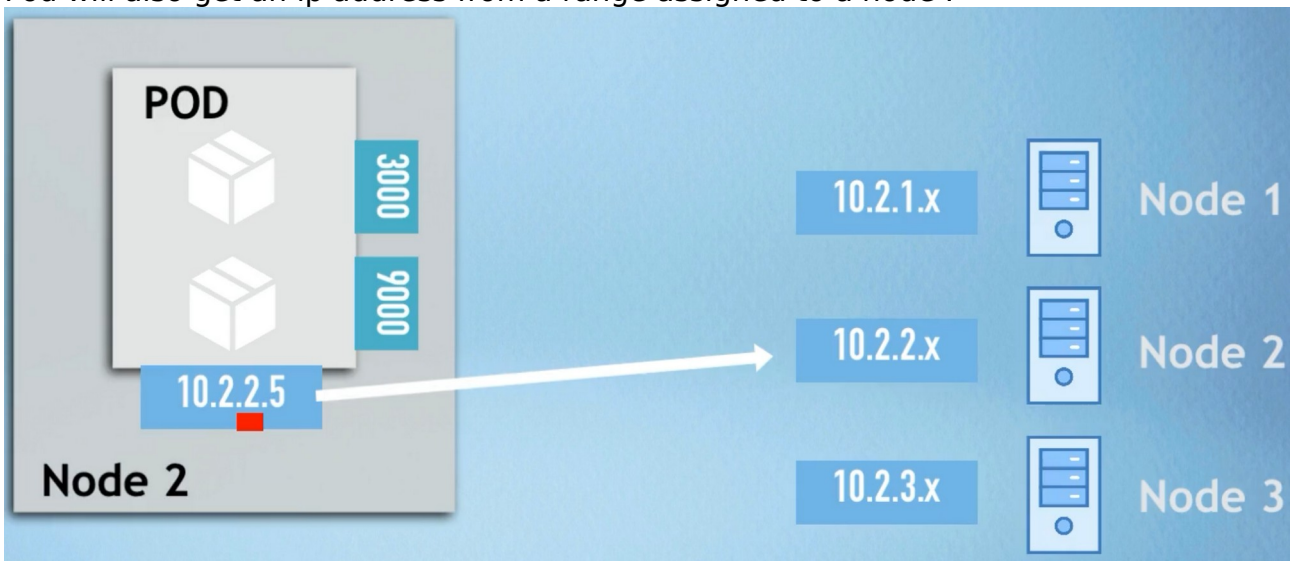
This is a default type of a service when u create a service and not specify a type .
Eg: we have micro_service application deployed in a cluster
We have a Pod with micro_service container running inside a pod
Beside micro_service container we have a side car container which collects the logs of the micro_service container .
Your micro_service container is running at pod 3000

logIn container is running on pod 9000



Those two pods will be now open and accessible inside the pod .

Pod will also get an ip address from a range assigned to a node .

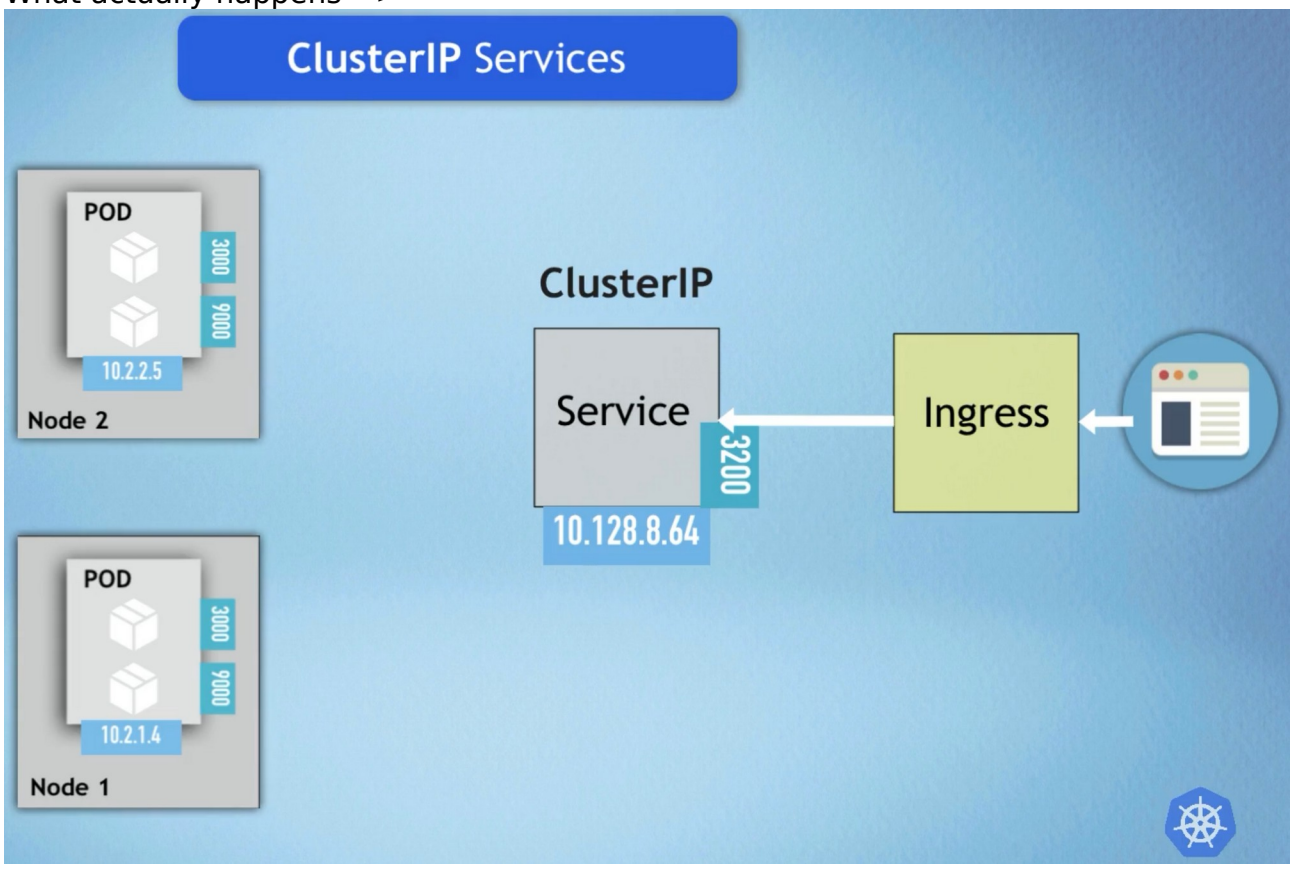


: to check the ip address of your pod inside your cluster : kubectl get pod -o wide

Now we can access those container at this ip_address and those ports .

If we set replica count to 2 it will create a new node with same pod and containers inside it with different ip address .

What actually happens =>

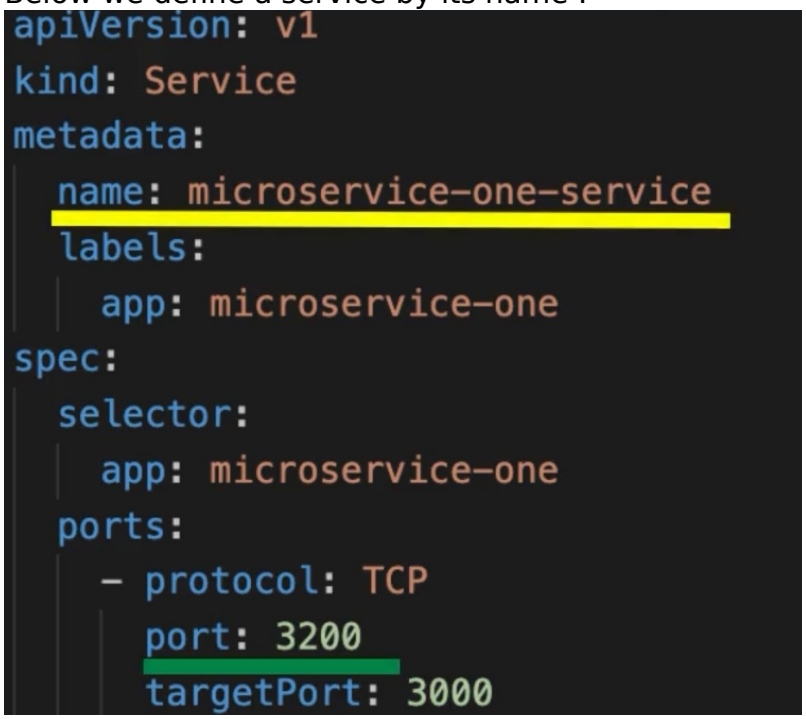


Ingress will forward the request based on request address to certain services we define the service by its name DNS resolution then maps that service to an ip_address Service in k8s is a component just like a pod but not a process its just an abstraction layer that represent the ipaddress and service will get an ip address where it is accessible at also it will be accessible at a certain port .

Below we have defined the ingress rule that will forward the request on the new address to certain services

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ms-one-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
    - host: microservice-one.com
      http:
        paths:
          - path: /
            backend:
              serviceName: microservice-one-service
              servicePort: 3200
```

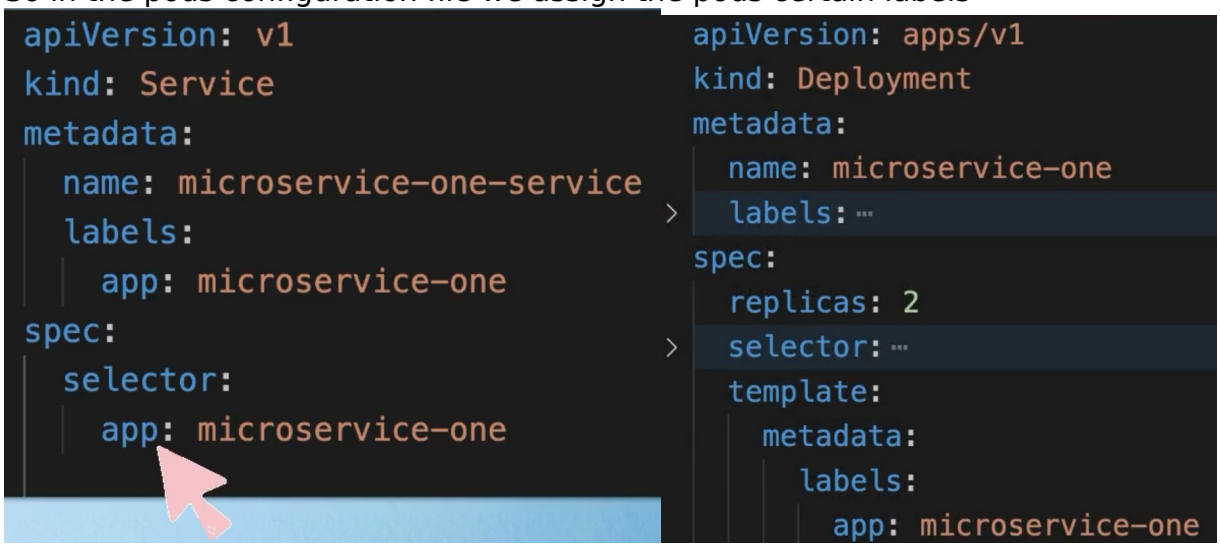
Below we define a service by its name :



One the request is handed over to the service than service will hand over the request to one of those pods that we have registered as a service end points .

How does service know ?
Which pods to forward the request to ?
Ans : selectors : pods are identified by selectors
In the service specification in the yaml file which will create the service will specify the selector attribute that has a key value pair defined as a list , key values pairs are the labels that pods should have to match that selector .

So in the pods configuration file we assign the pods certain labels

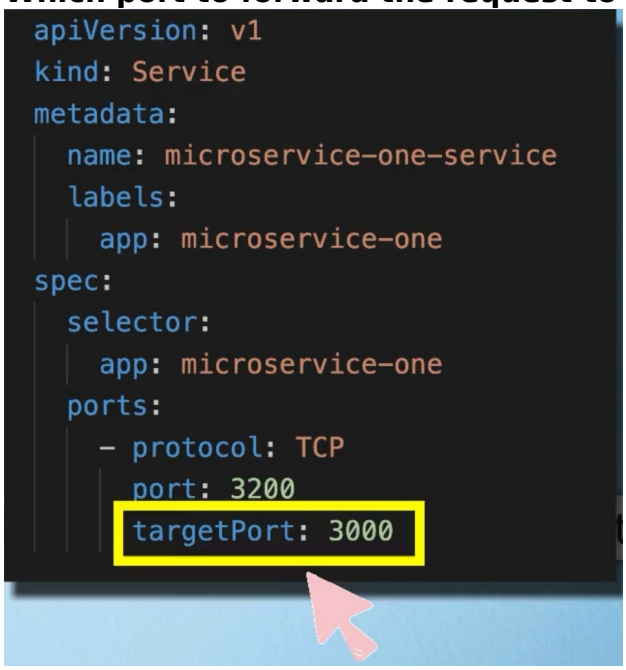


Eg: if we have 3 replicas of deployment pods or nodes each with labels
Service should match all 3 replica tables as selector attribute to register those deployment as a endpoint of that service . Must match all the selectors .

Then the service will know which pod belong to it and where to forward the request to .

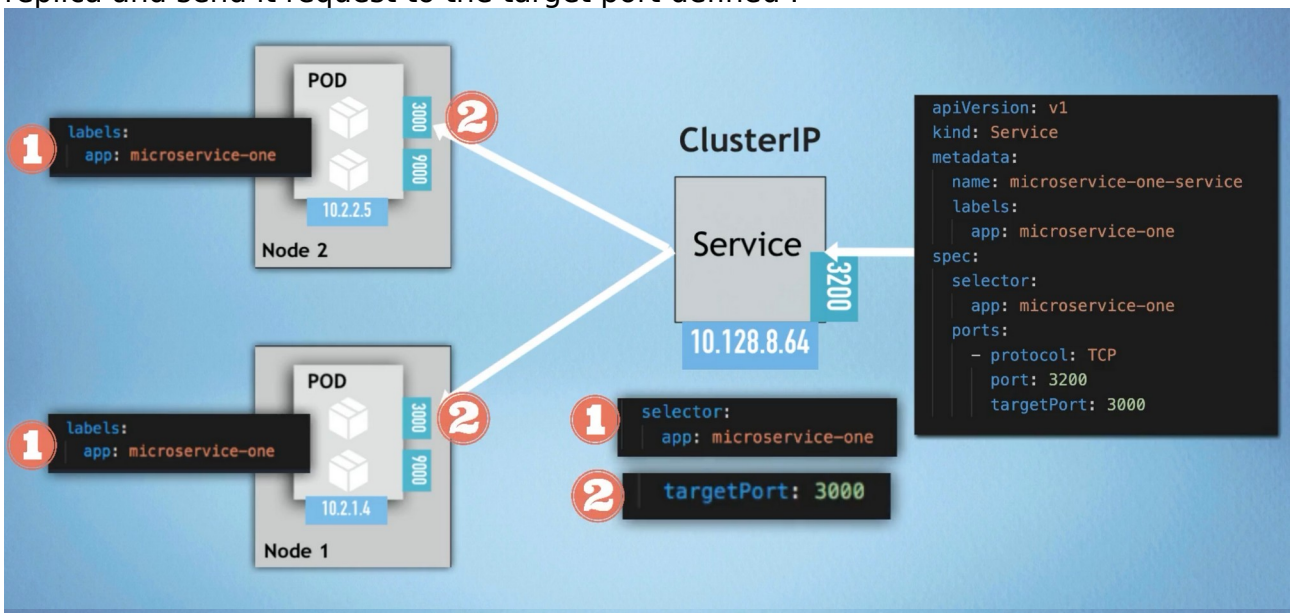
But the the pod have multiple pods open who does service know which port to forward the request to >?

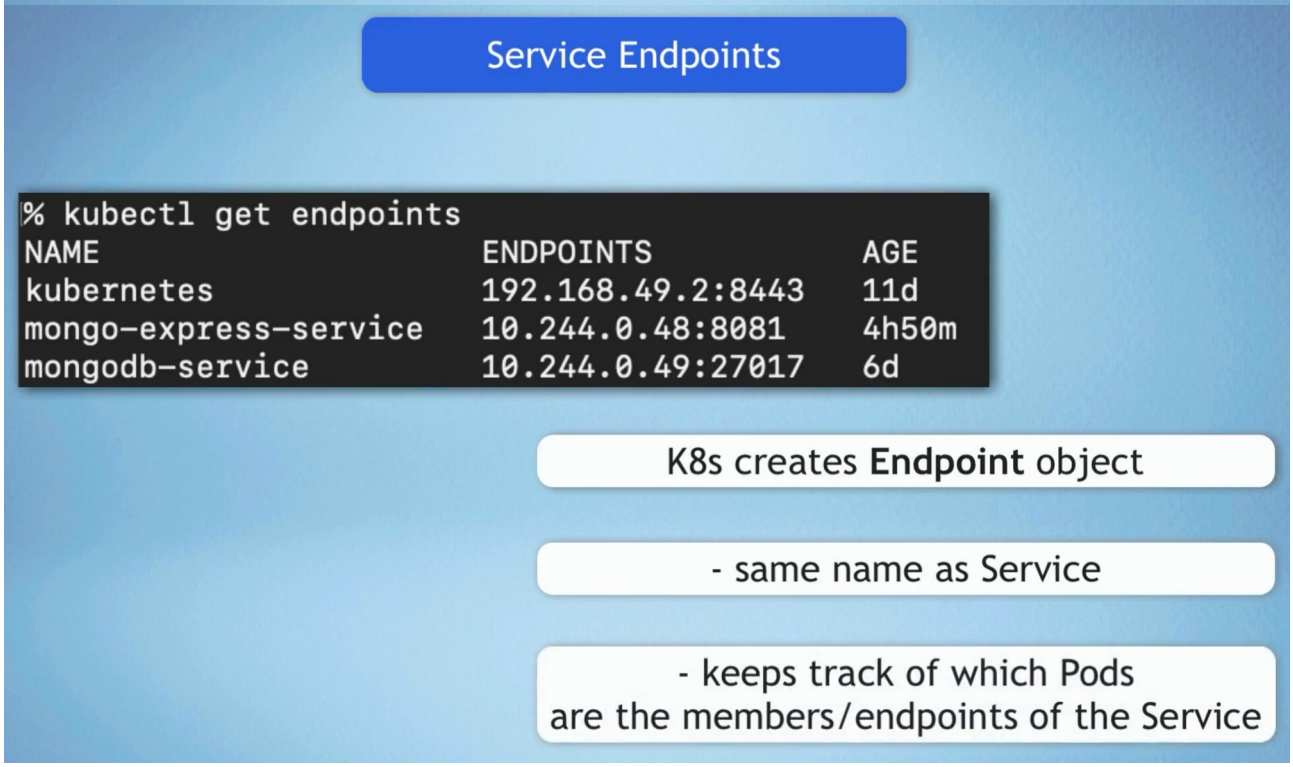
Which port to forward the request to on that specific pod?



And this is defined in the target port attribute inservice configuration .

When we create a service it will find a pod in selector of this pods will become the end point of the service when service will get a request it will pick one of those pods replica and send it request to the target port defined .





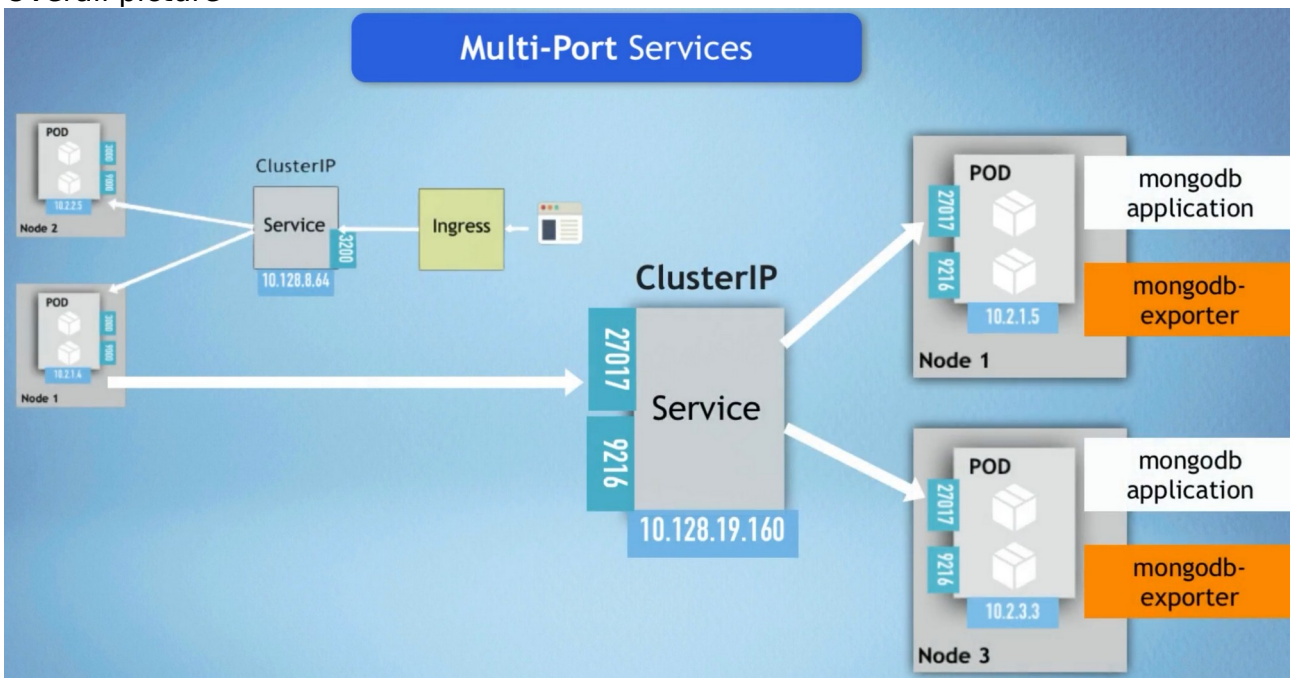
Service port is arbitrary , target port must match the port the container is listing at.

Now our container need to hand the request and need DB for that eg: micro_service uses mongoDB DB .

So we have two replica of mongoDB and cluster it have there own service end point so the microservice applicant inside the pod can talk to that mongoDB also using the service end point

Below is how the mongoDB service and deployment configuration files look like ->

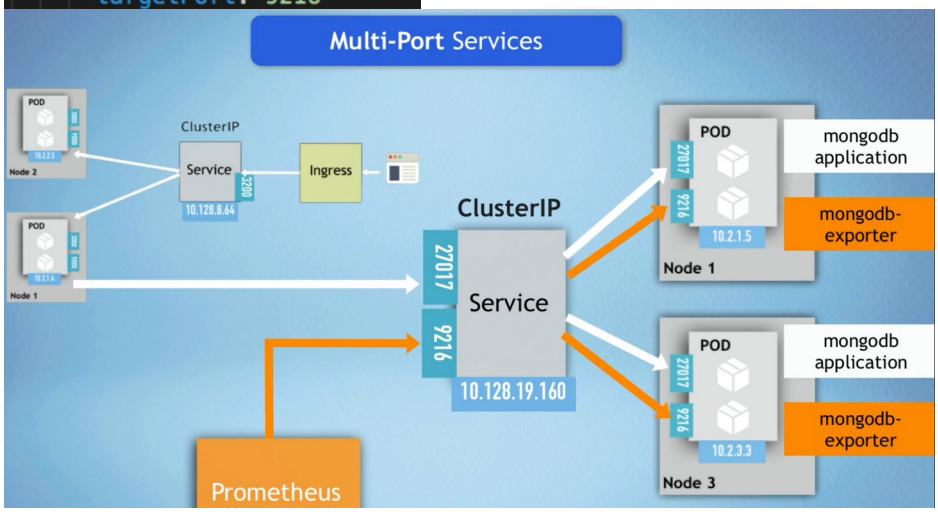
Overall picture ->



There is another client prometheus use to monitor which will take the logs data from mongodb exporter.

And when u have multiple ports defined in the service u need to name those ports .

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
  ...
spec:
  selector:
    app: mongodb
  ports:
    - name: mongodb
      protocol: TCP
      port: 27017
      targetPort: 27017
    - name: mongodb-exporter
      protocol: TCP
      port: 9216
      targetPort: 9216
```

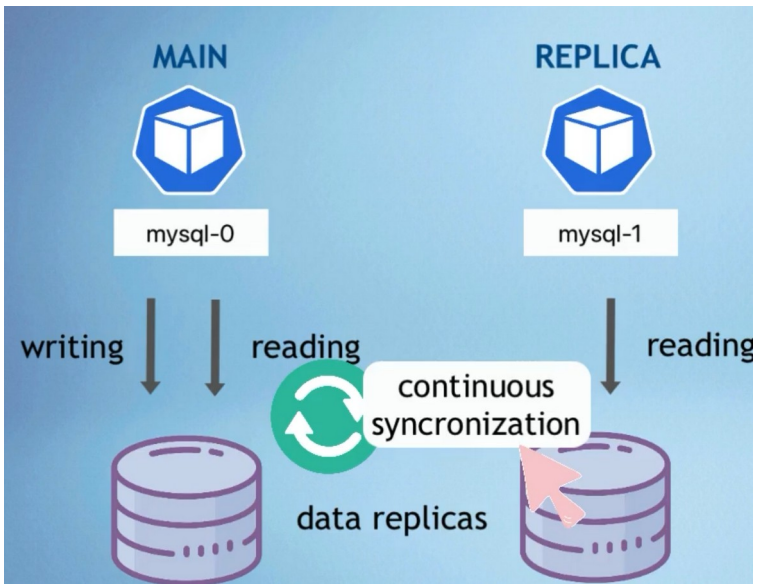


2. Headless service :

Client wants to communicate with one specific pod directly.

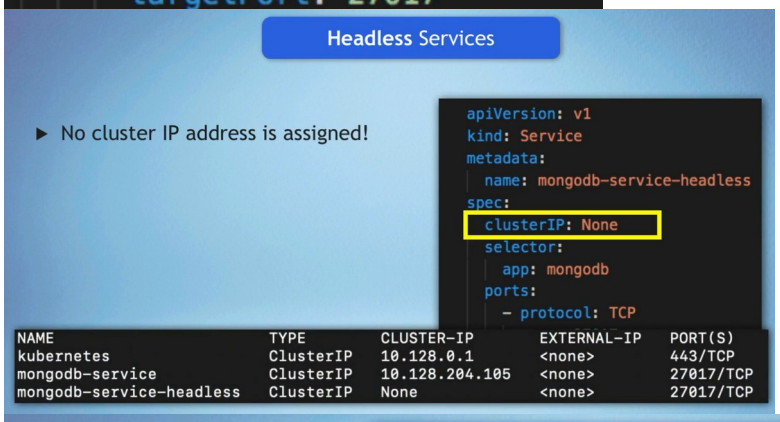
Pods wants to talk directly with a specific pod with out going to the service .

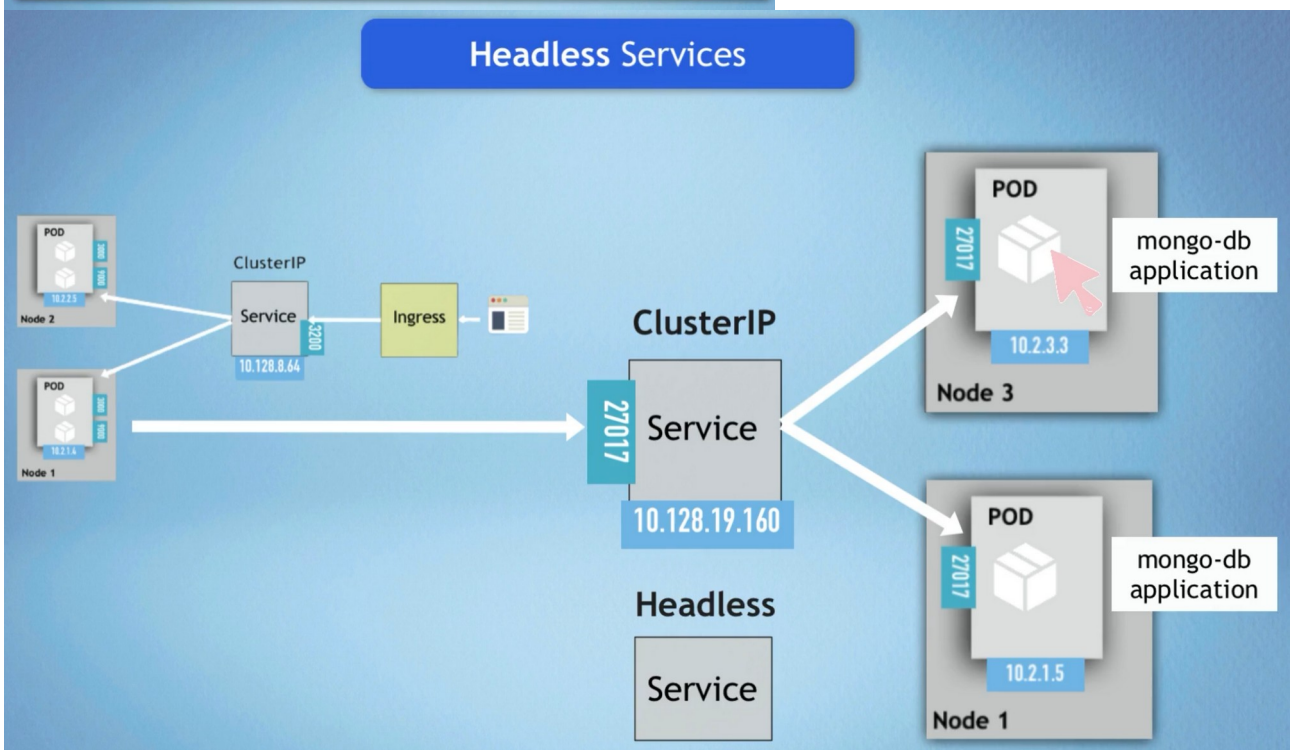
Use case of this requirements are : when we deploying stateful application to the k8s like database, elastic search .
On that cases pod replicas are not identical eg: ONLY MAIN instance of DB is allowed to write in the DB . And replica instance of DB will just read the DB



That the use can when u want the direct communication to the individual pods .
When client perform a DNS look up to a service DNS server returns a single ip address that belongs to that service .
This will be the services cluster ip address .
Set the cluster ip to NONE - return the pod ip address instead of the services ip address .

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service-headless
spec:
  clusterIP: None
  selector:
    app: mongodb
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```



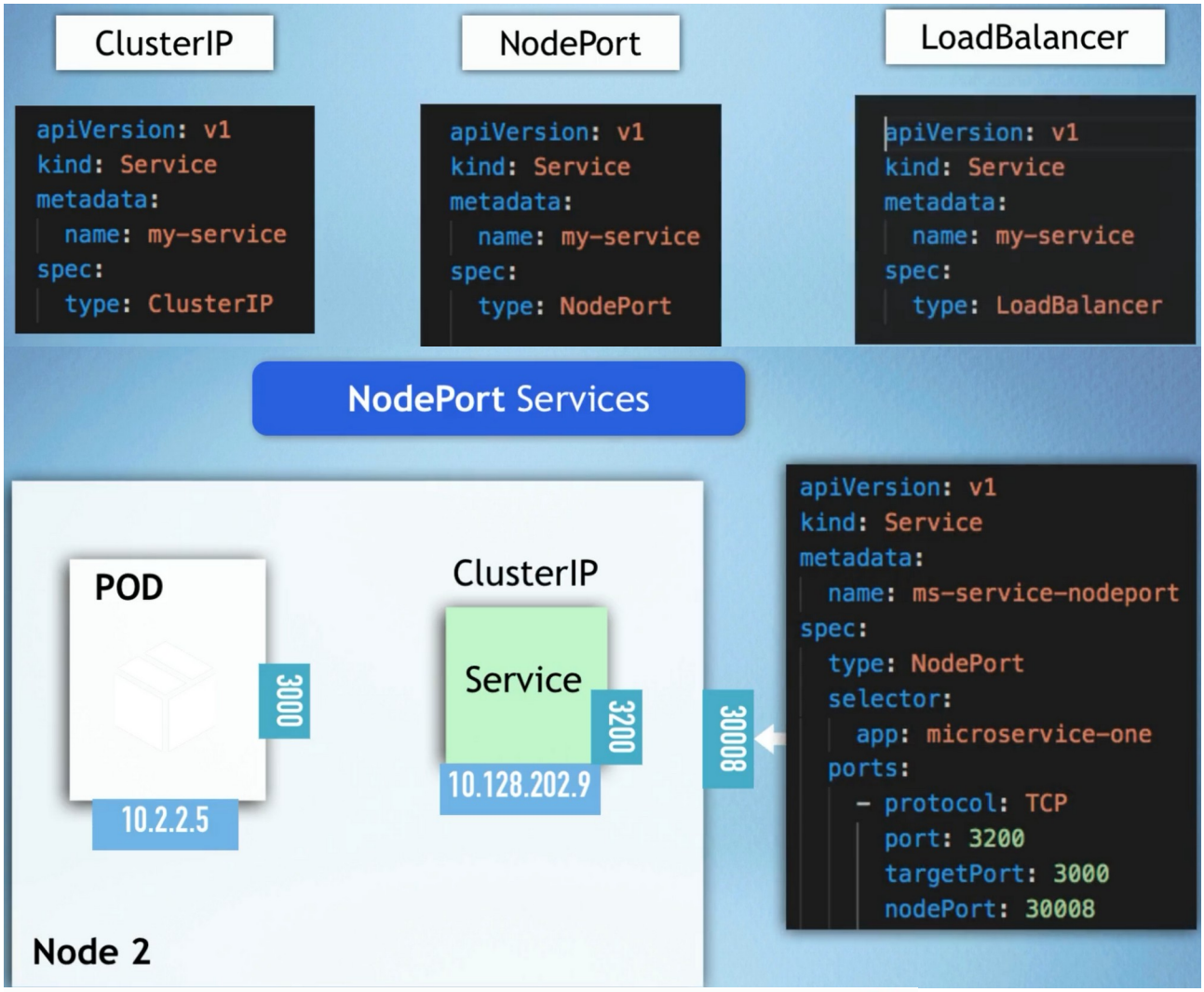


3. Node port service :

When we define a service there are three type we can define it :
=> clusterIP , NodePort , LoadBalancer

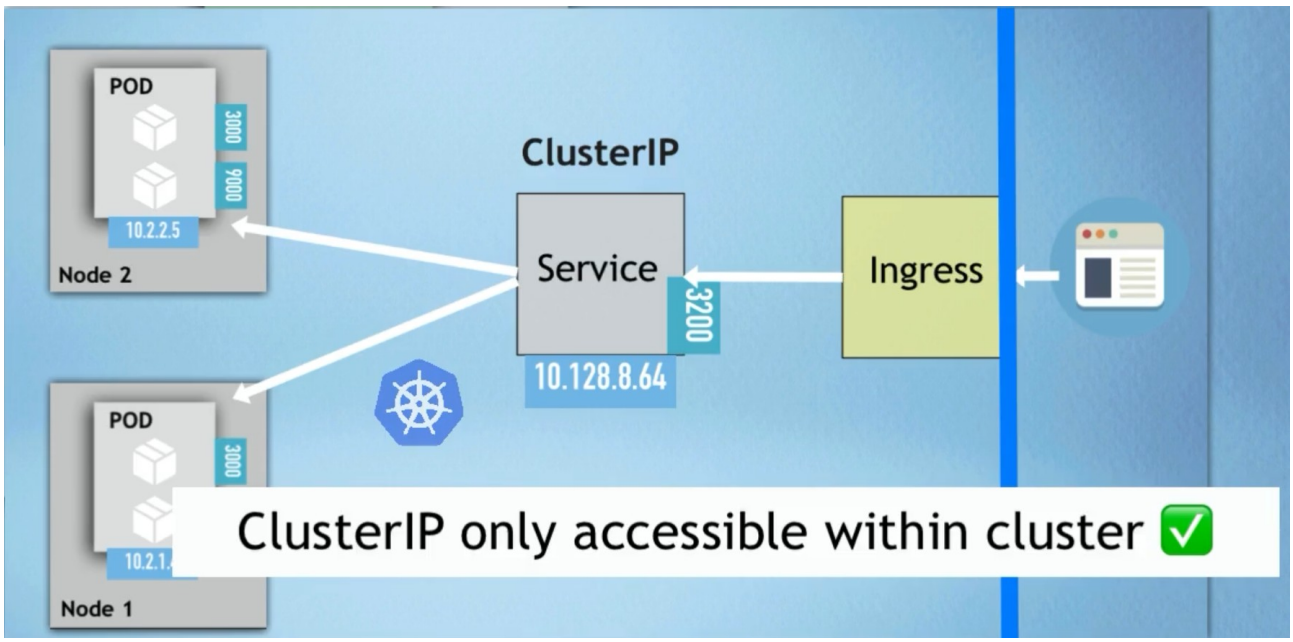
Default type not needed .

Internal service .



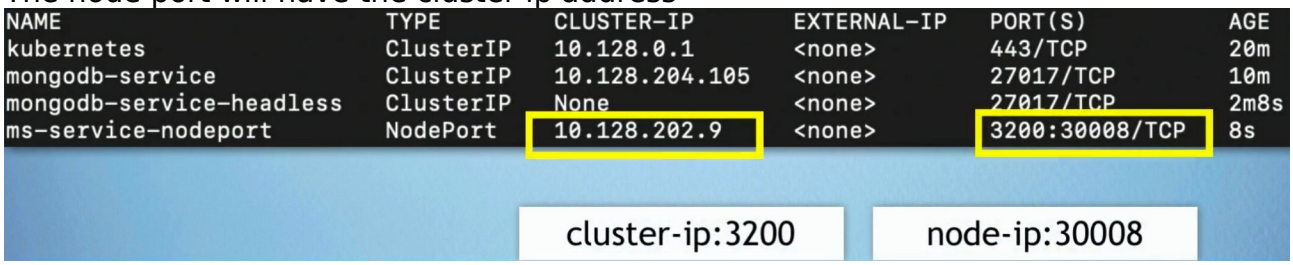
Range: 30000 - 32767 !

IN the previous example cluster ip service is only accessible with in the cluster .
So no external traffic can directly address the cluster ip address .
In node port service the external traffic has a access to fixed port on each worker node .



So in node port service in stead on Ingress the browser request will come directly on the on the nodeport attribute that the service specification defines .

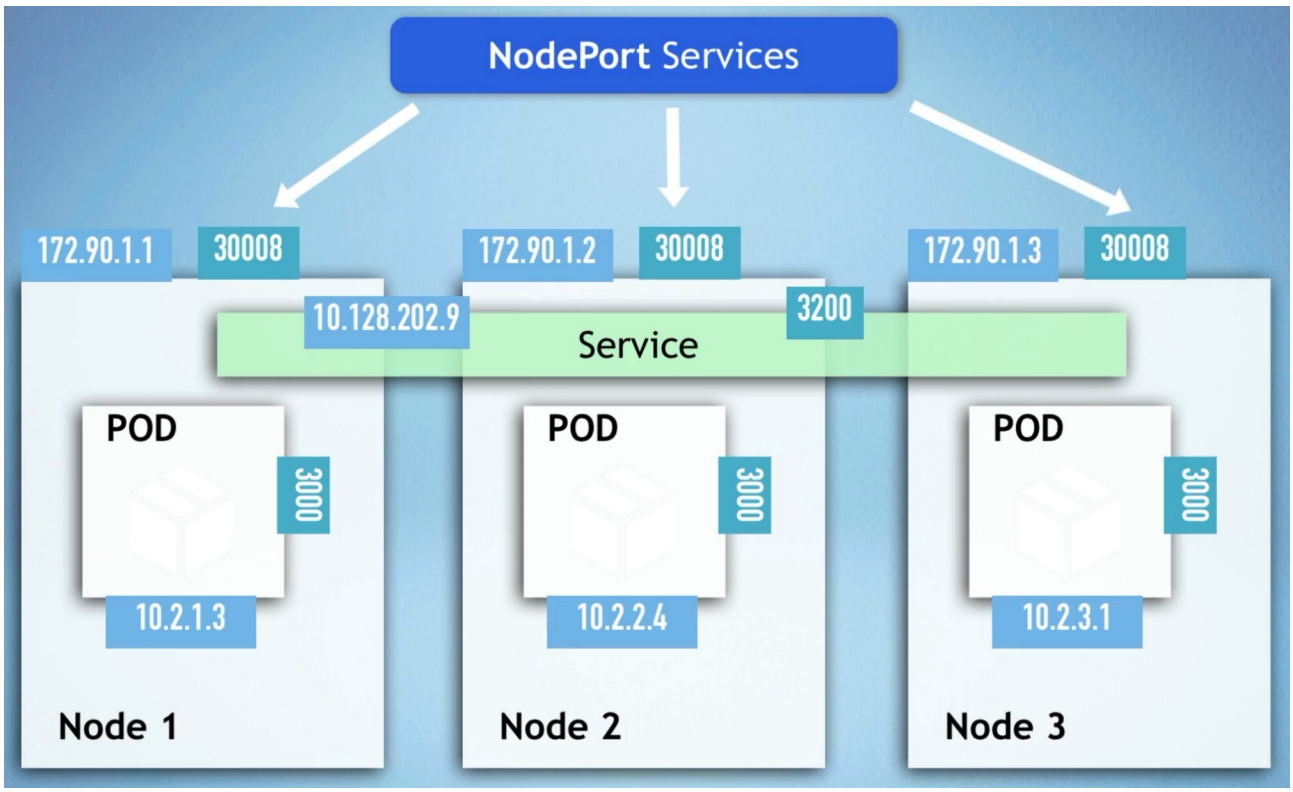
The node port will have the cluster ip address

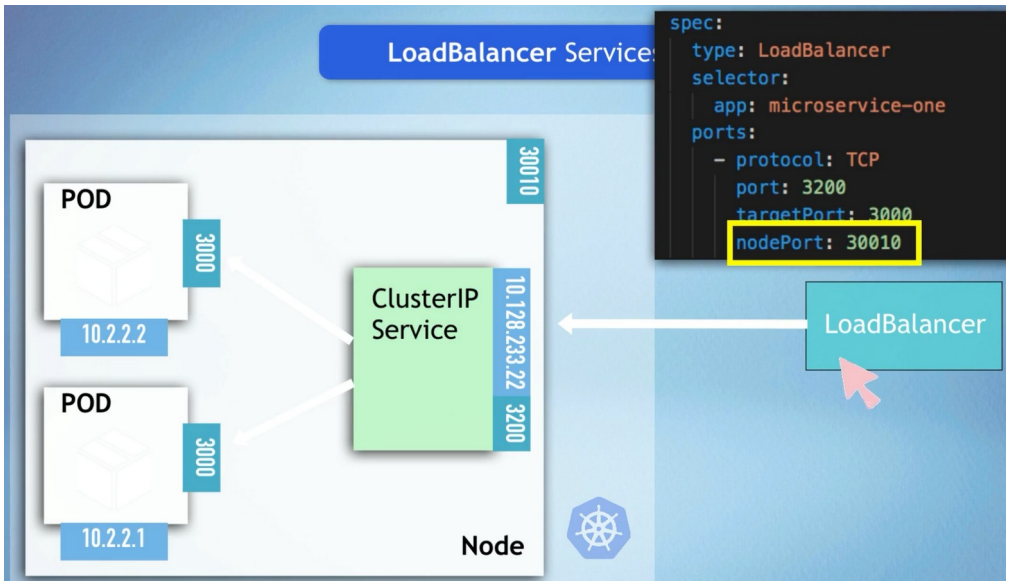


Above is the pod replicas :

Node port services are not very secure .

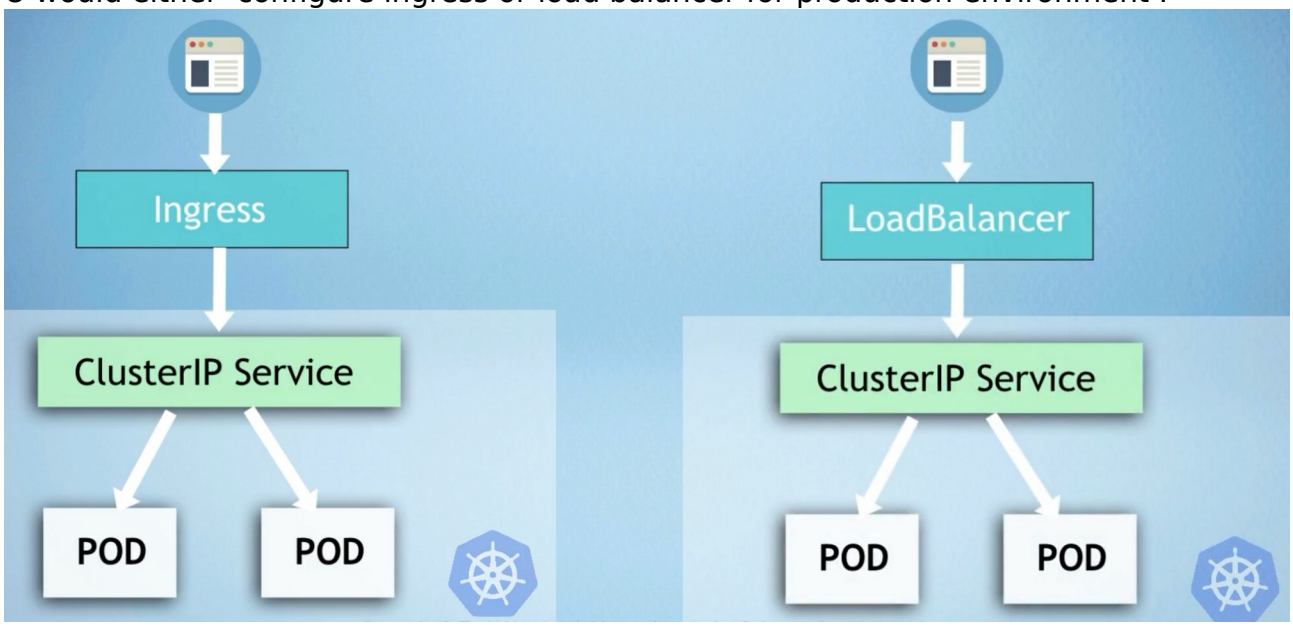
As a better alternative of node port service we have **load balancer service :**
Becomes accessible externally through cloud providers load balancer
When ever we create a load balancer service = cluster ip and nodePort service are created automatically .


NodePort Services
172.90.1.1
30008
172.90.1.2
30008
172.90.1.3
30008
10.128.202.9
Service
3200
POD
3000
10.2.1.3
POD
3000
10.2.2.4
POD
3000
10.2.3.1
Node 1
Node 2
Node 3


LoadBalancer Service
spec:
type: LoadBalancer
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
nodePort: 30010
30010
POD
3000
10.2.2.2
POD
3000
10.2.2.1
ClusterIP
Service
10.128.233.22
3200
LoadBalancer
Node

So in short the load balancer service is a extension of nodes out service type
Node port service is a extension of clusterIP service type .

In real k8s cluster u would probably not use node port for external connection .
U would either configure ingress or load balancer for production environment .



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
kubernetes	ClusterIP	10.128.0.1	<none>	443/TCP
mongodb-service	ClusterIP	10.128.204.105	<none>	27017/TCP
mongodb-service-headless	ClusterIP	None	<none>	27017/TCP
ms-service-loadbalancer	LoadBalancer	10.128.233.22	172.104.255.5	3200:30010/TCP
ms-service-nodeport	NodePort	10.128.202.9	<none>	3200:30008/TCP