K8S

Fundamentals

Url:

Kubernetes Tutorial for Beginners [FULL COURSE in 4 Hours]

Things to KNOW

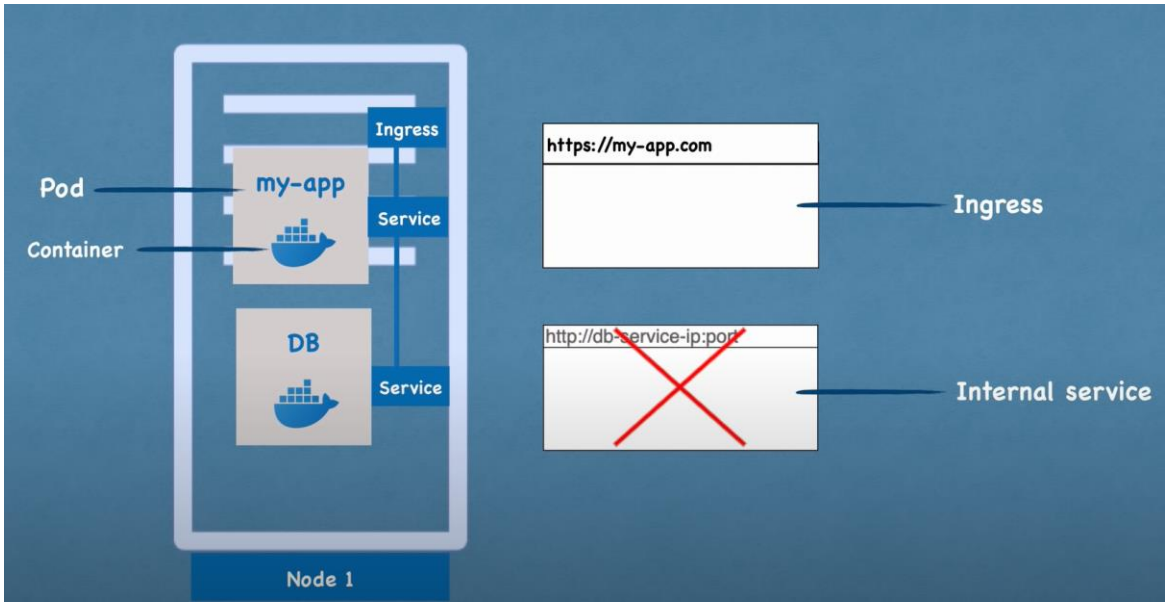
Hirerachy: Cluster -> Deployment -> ReplicaSet -> pod -> container

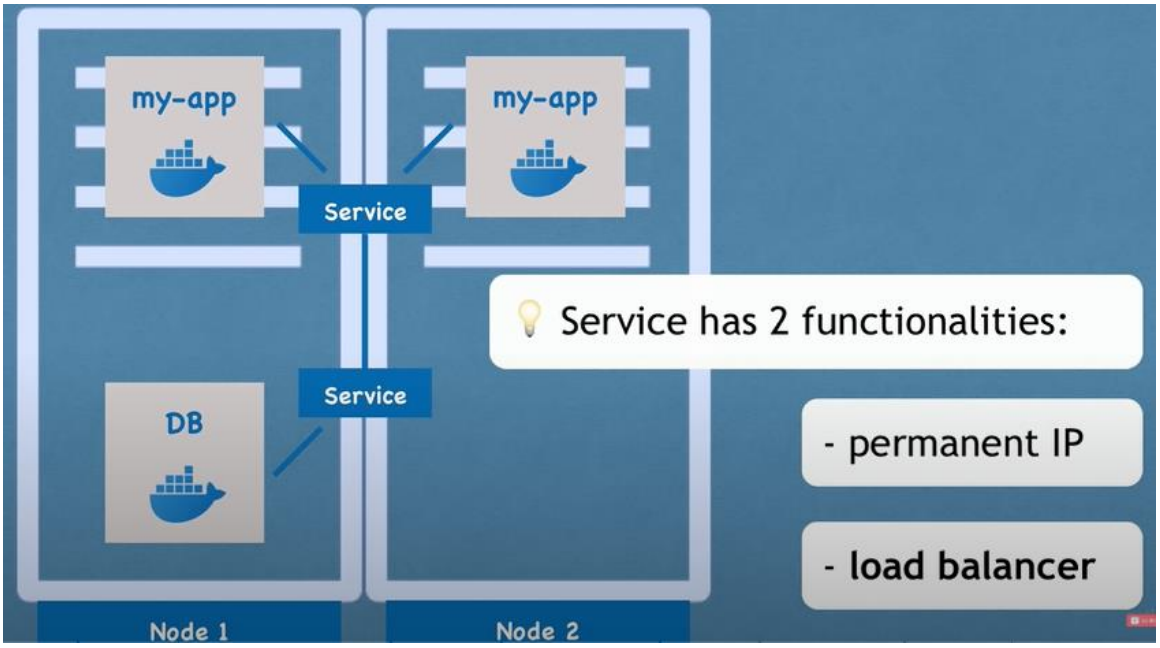
Control manager : get cluster state and push to etcd (cluster store)

Etcd: cluster store in a key: vlaue sets)

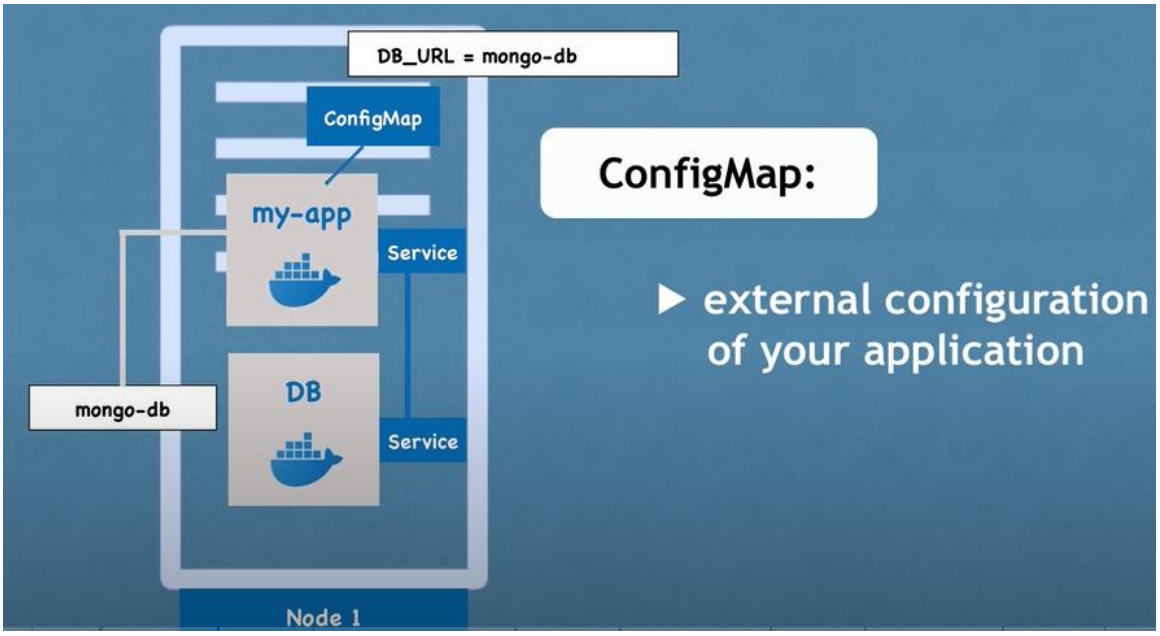
Deployment + Pod (**metadata**: **labels**: app:) connect to service (**spec**: **selector**: app:)

SERVICE and INGRESS





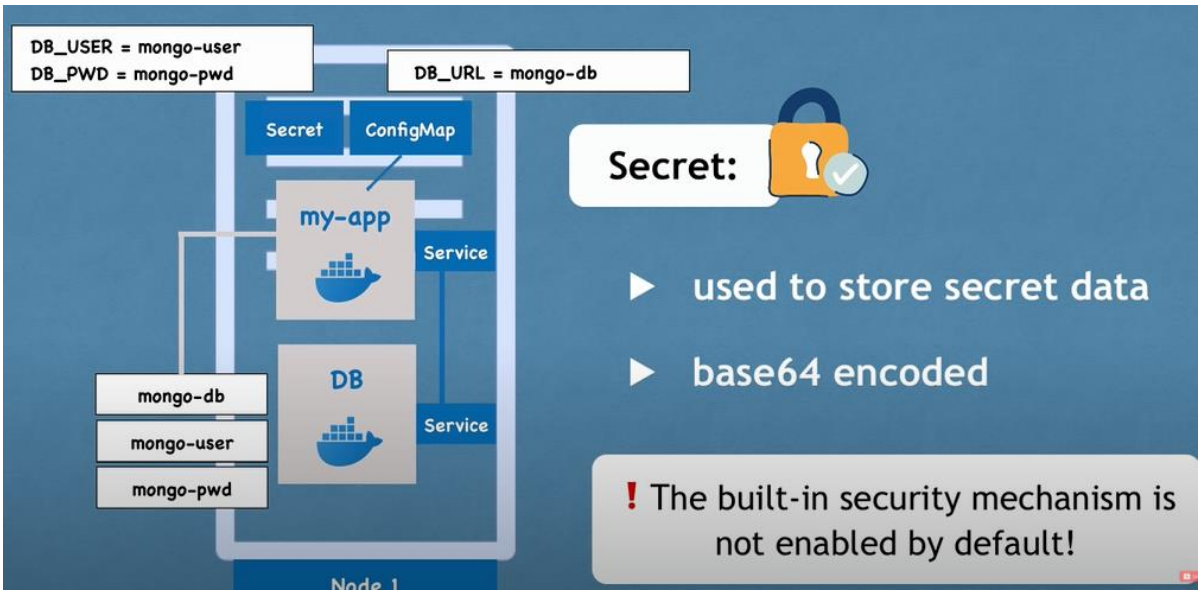
ConfigMap



DONT PUT CREDENTIAL in ConfigMap

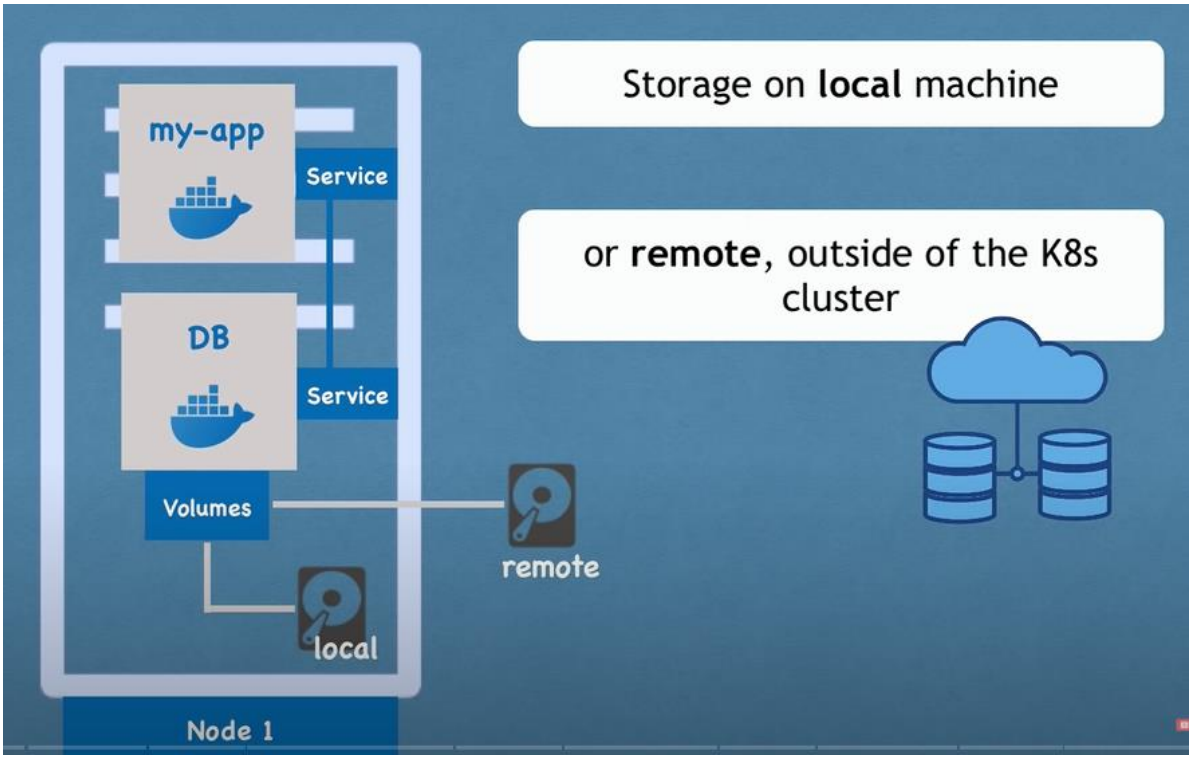
In the application we can use it as env variables or properties files

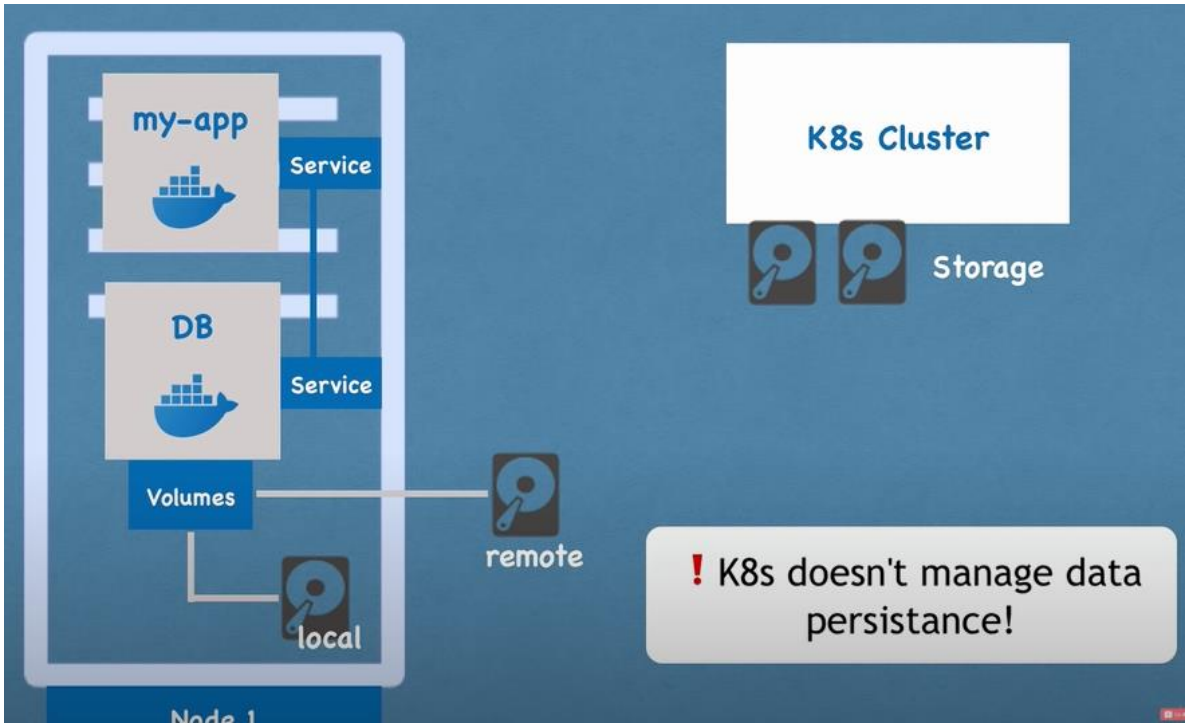
SECRETS



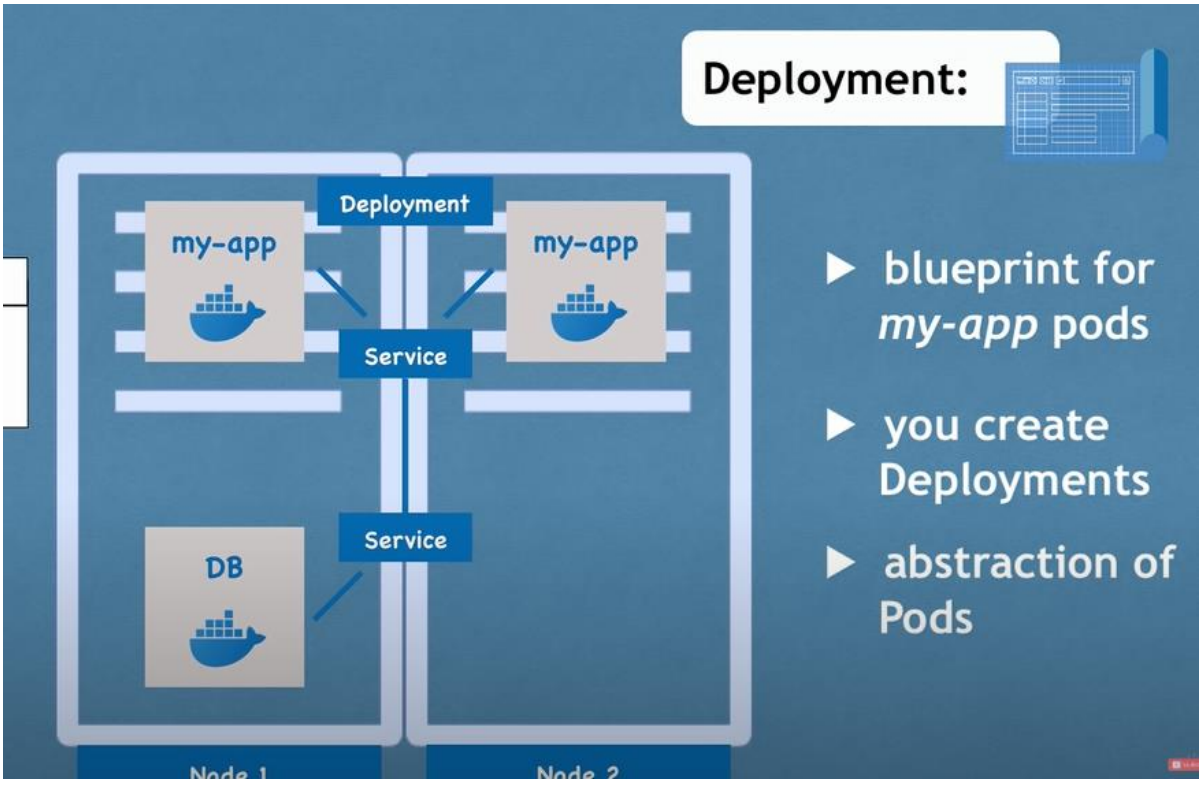
In the application we can use it as env variables or properties files

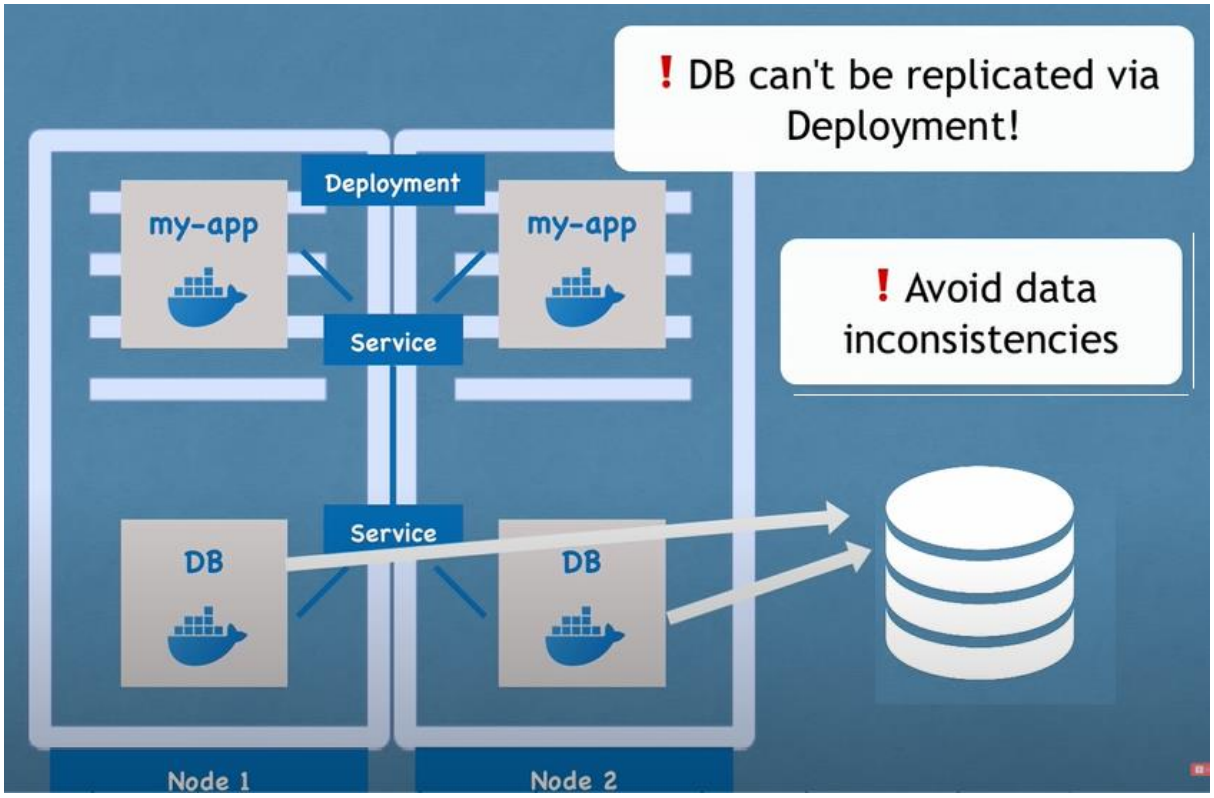
VOLUMES

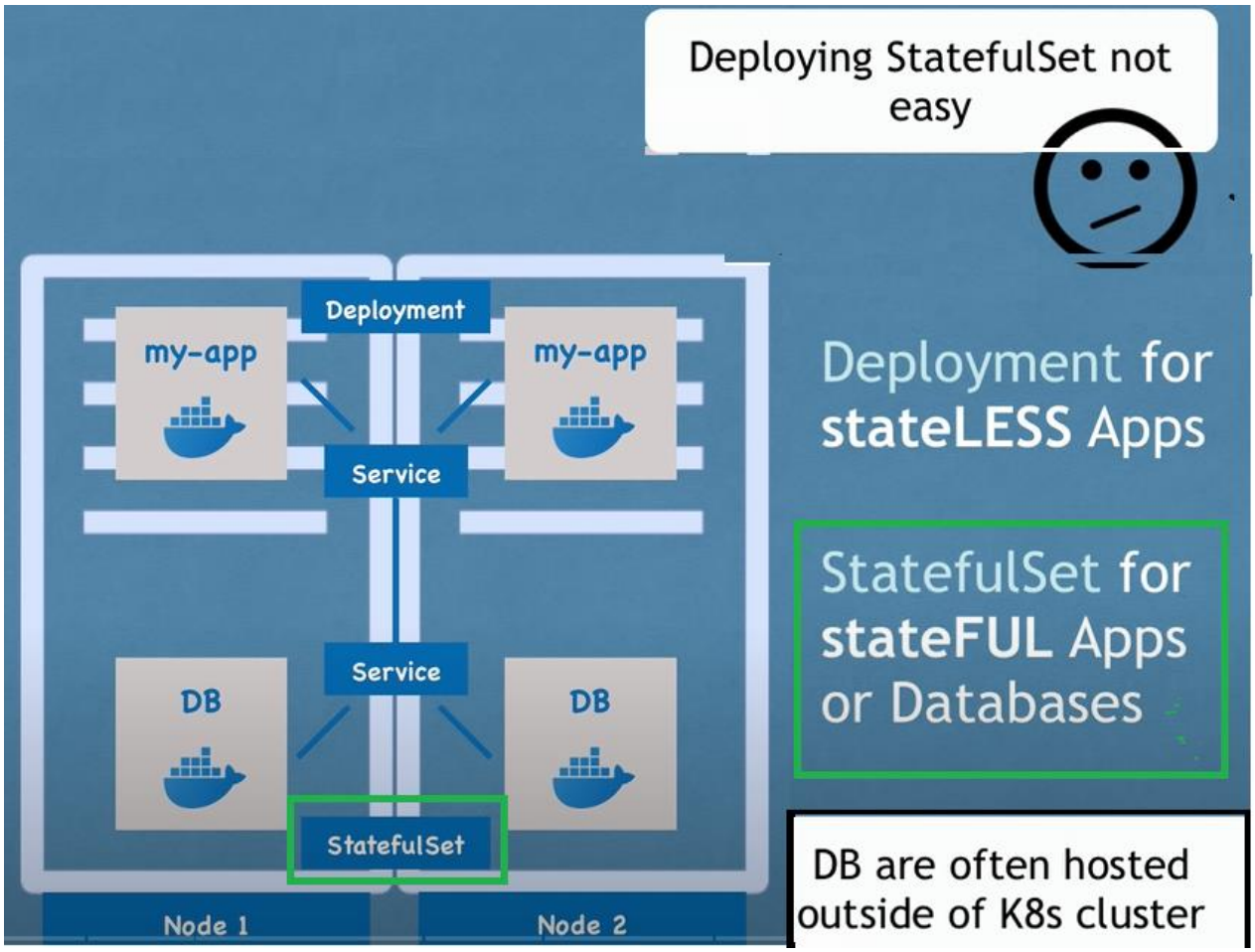




Deployment and Statefull set

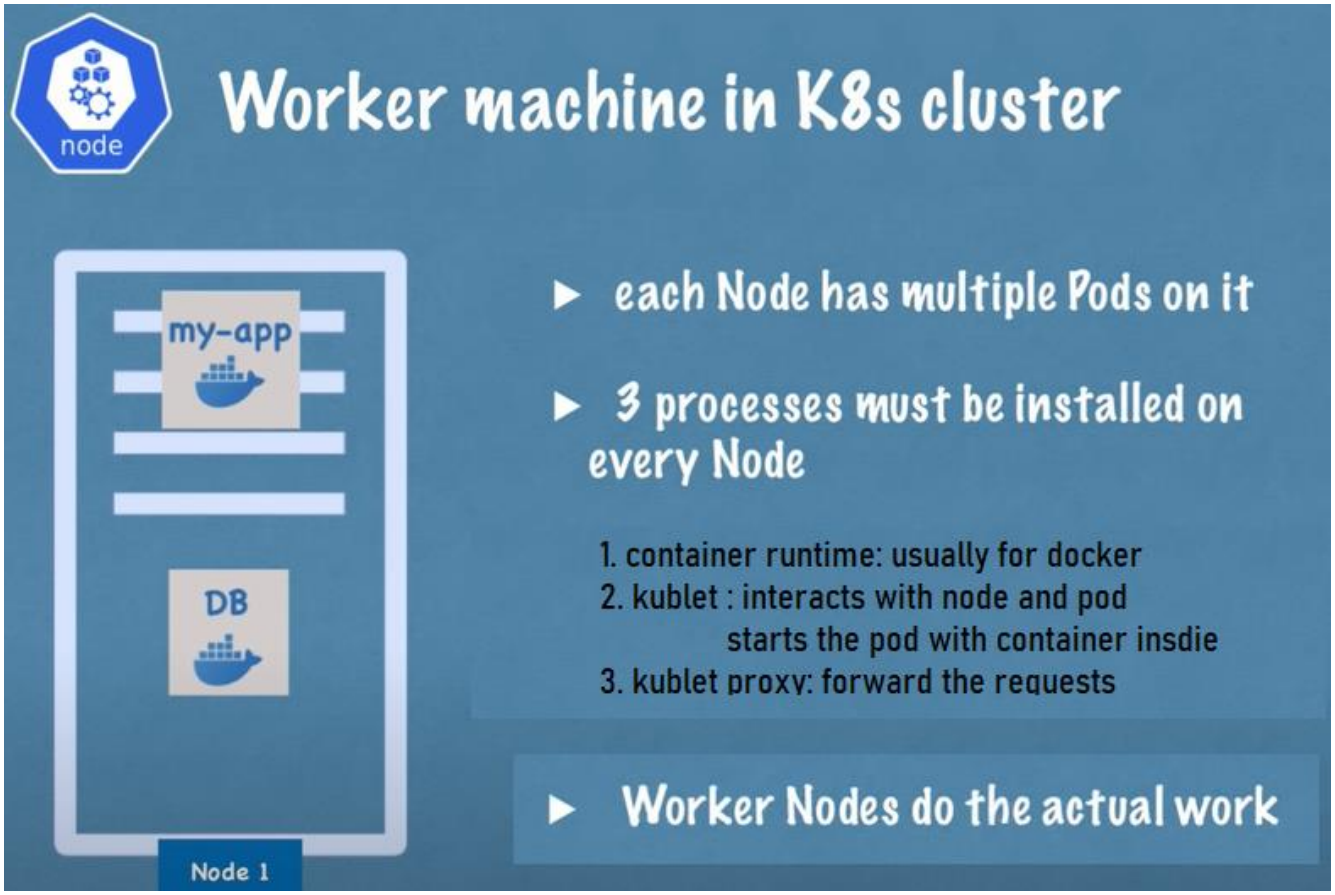



! DB can't be replicated via Deployment!
Deployment
my-app
my-app
Service
! Avoid data inconsistencies
Service
DB
DB
Node 1
Node 2

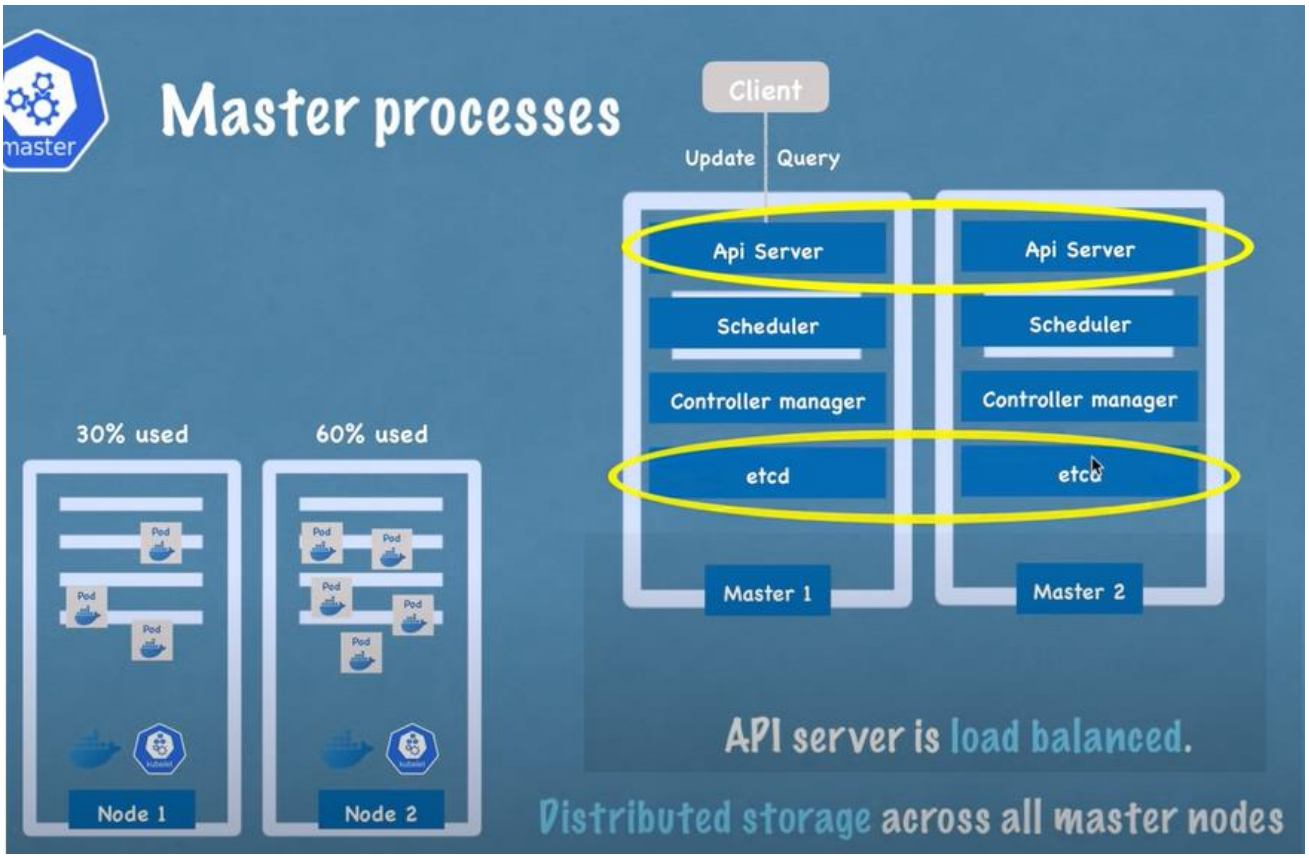

Deploying StatefulSet not easy
Deployment
my-app
my-app
Service
Service
DB
DB
StatefulSet
Node 1
Node 2
Deployment for stateLESS Apps
StatefulSet for stateFUL Apps or Databases
DB are often hosted outside of K8s cluster

Architecture

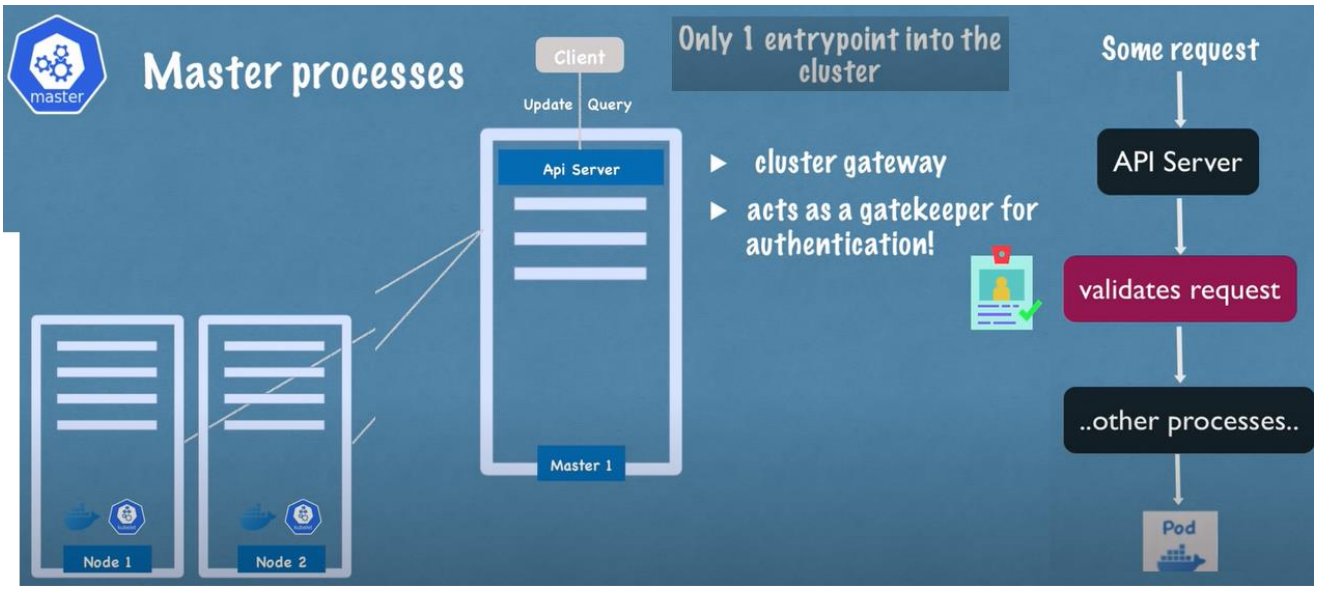
Worker node



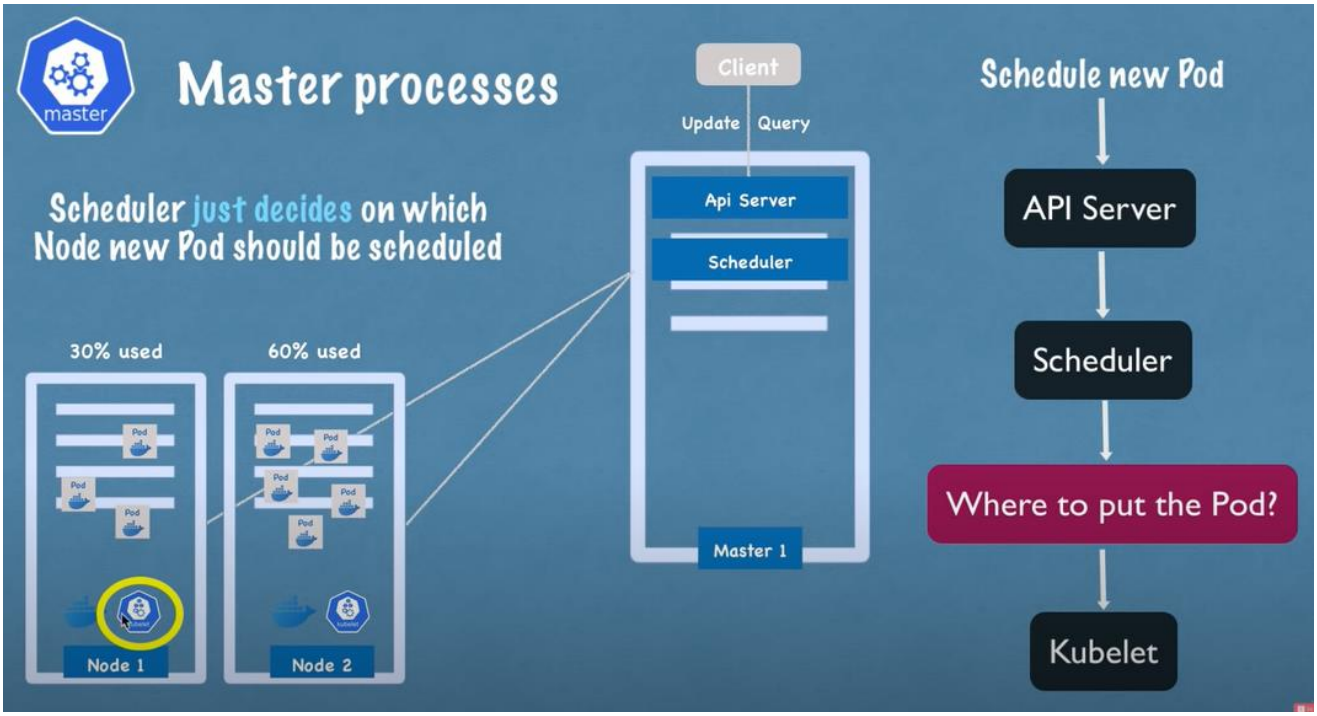
MASTER NODE



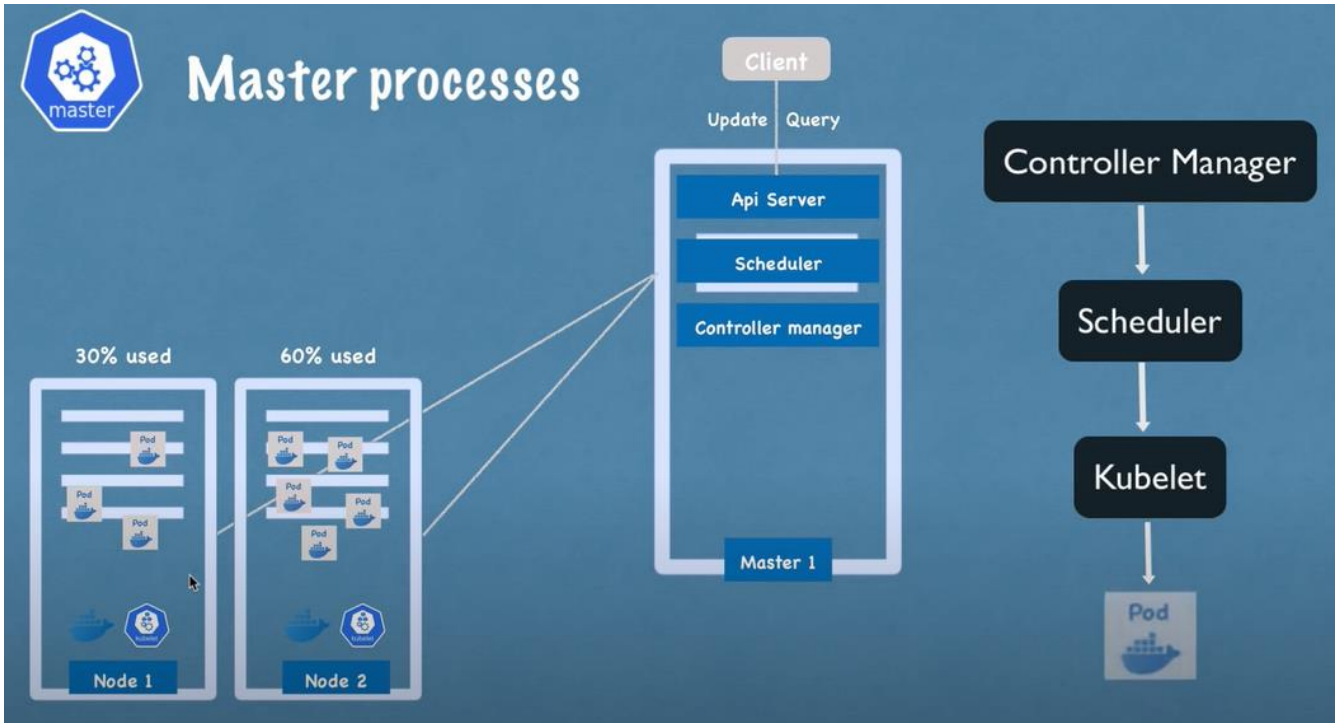
API Server



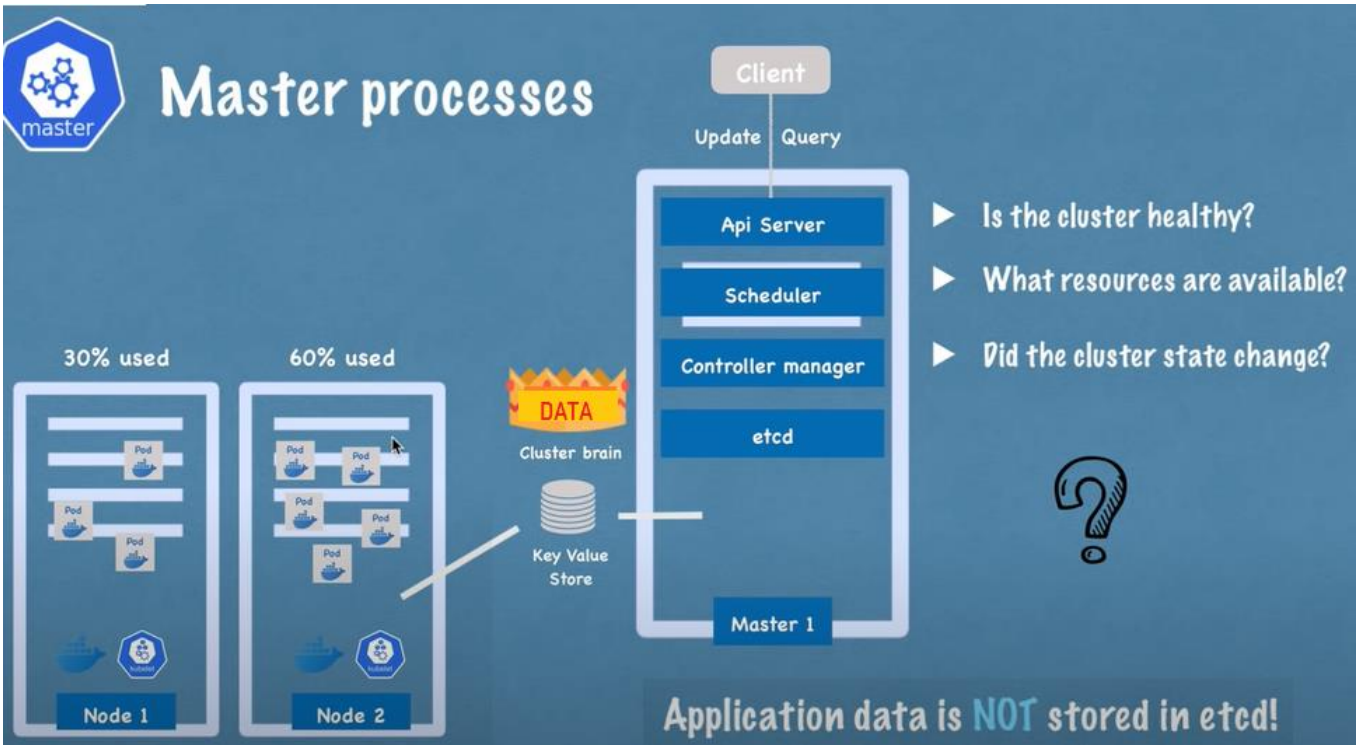
Scheduler



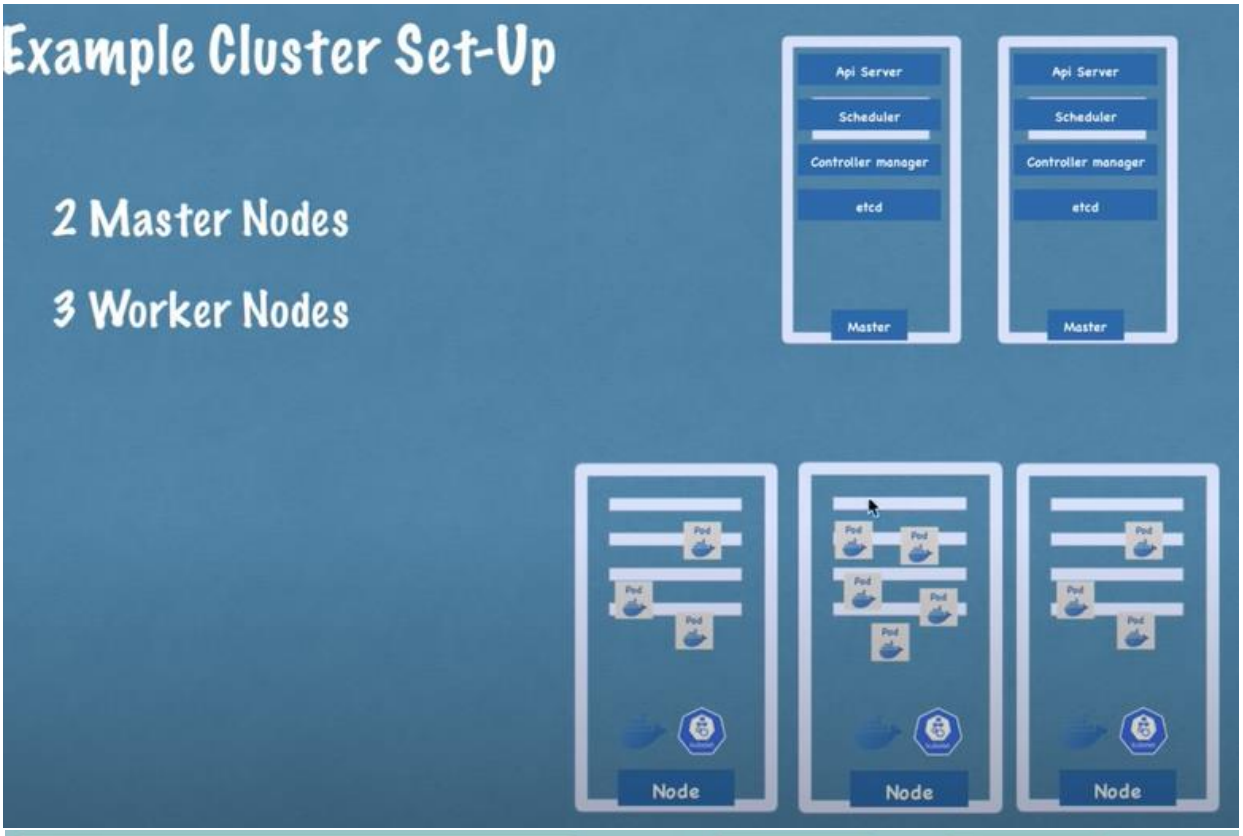
Controller Manager : detects cluster state changes

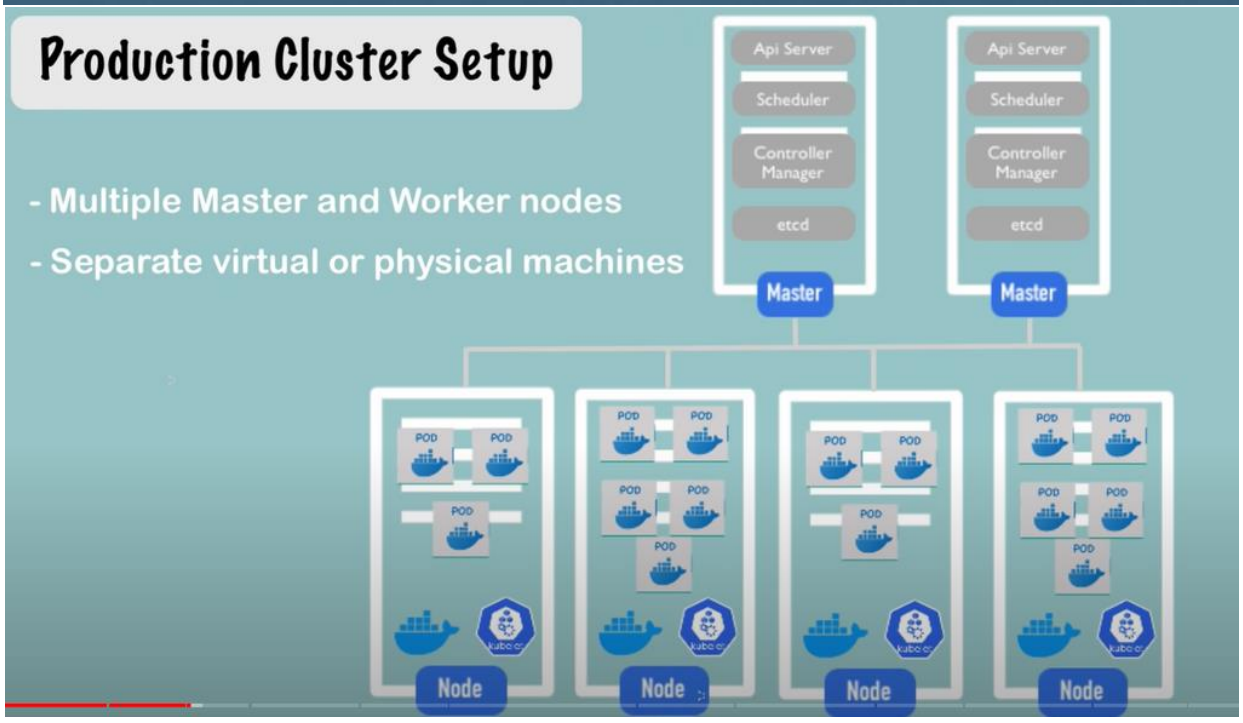


etcd : cluster store


Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
etcd
Master 1
Is the cluster healthy?
What resources are available?
Did the cluster state change?
30% used
60% used
DATA
Cluster brain
Key Value Store
Pod
Node 1
Node 2
Application data is NOT stored in etcd!

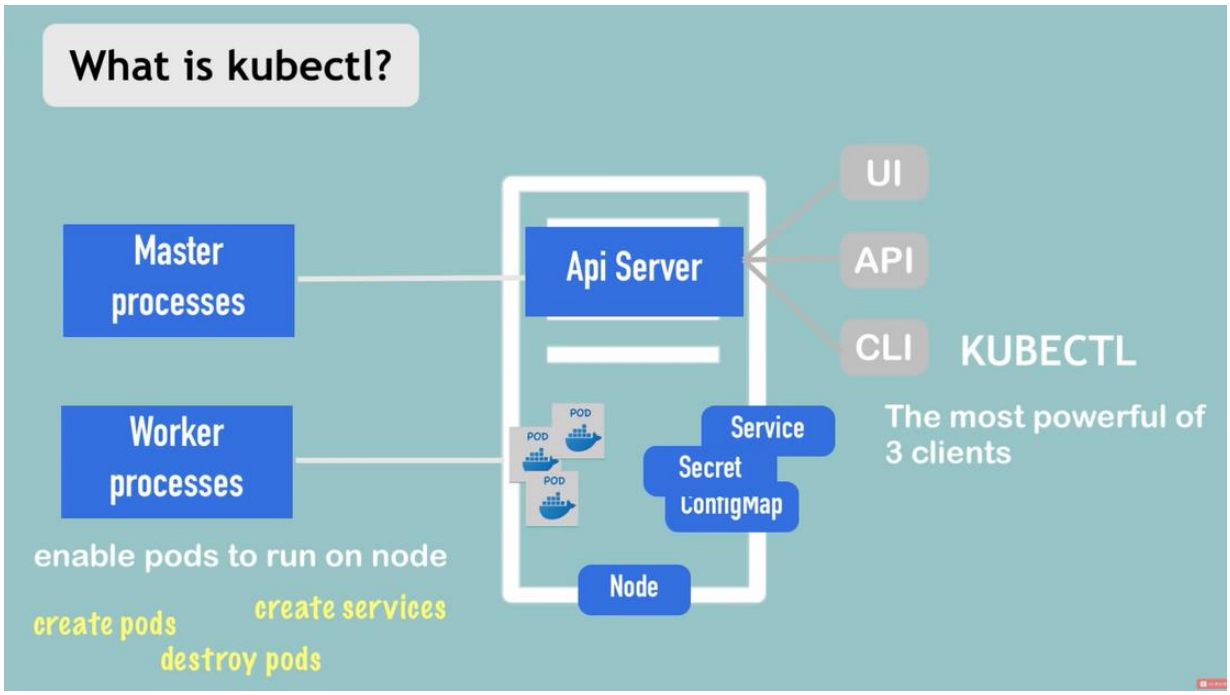
CLUSTER


Example Cluster Set-Up
2 Master Nodes
3 Worker Nodes
Api Server
Scheduler
Controller manager
etcd
Master
Api Server
Scheduler
Controller manager
etcd
Master
Pod
Node
Node
Node


Production Cluster Setup
- Multiple Master and Worker nodes
- Separate virtual or physical machines
Api Server
Scheduler
Controller Manager
etcd
Master
Api Server
Scheduler
Controller Manager
etcd
Master
POD
Node
Node
Node
Node

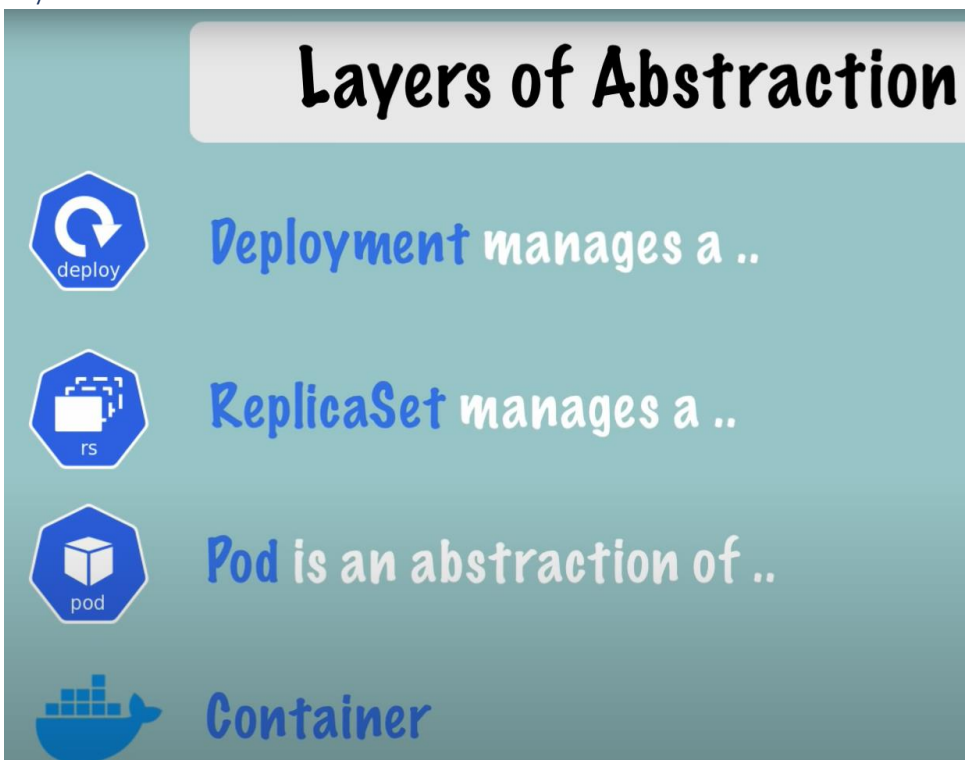
Minikube and kubectl local setup

API SERVER - Kubectl

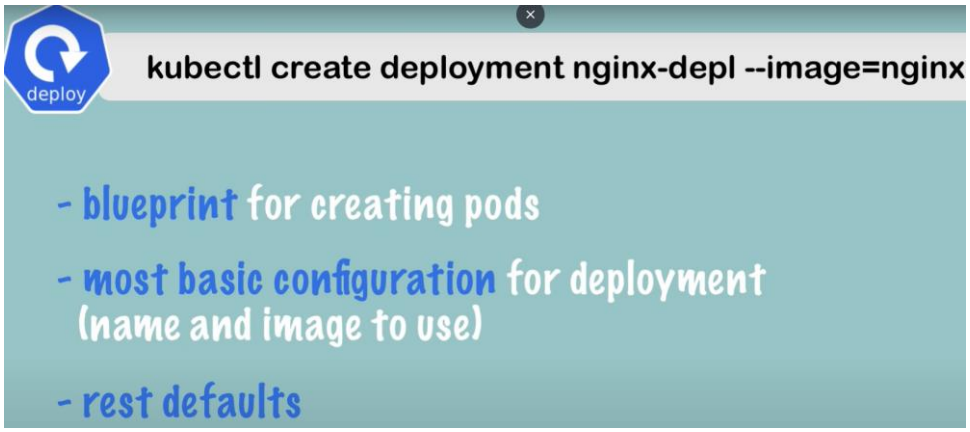


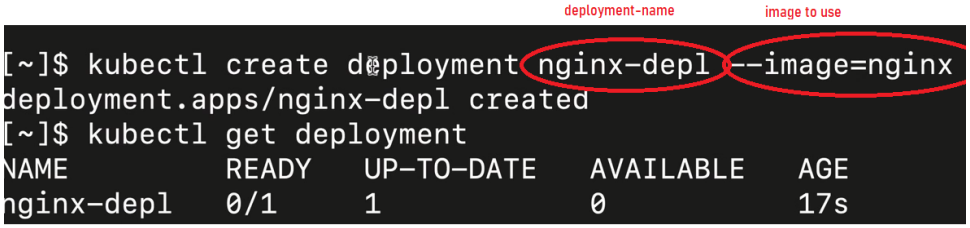
Kubectl

Layers



deployment






```
[~]$ kubectl create deployment nginx-depl --image=nginx
deployment.apps/nginx-depl created
[~]$ kubectl get deployment
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx-depl   0/1     1            0           17s
[~]$ kubectl get pod
NAME                          READY   STATUS              RESTARTS   AGE
nginx-depl-7d9447675c-j9j8k   0/1     ContainerCreating   0          31s
```

Can be edited to update image version / etc

kubectl edit deployment <deployment name>

Kubectl delete deployment <deployment name >

Example:

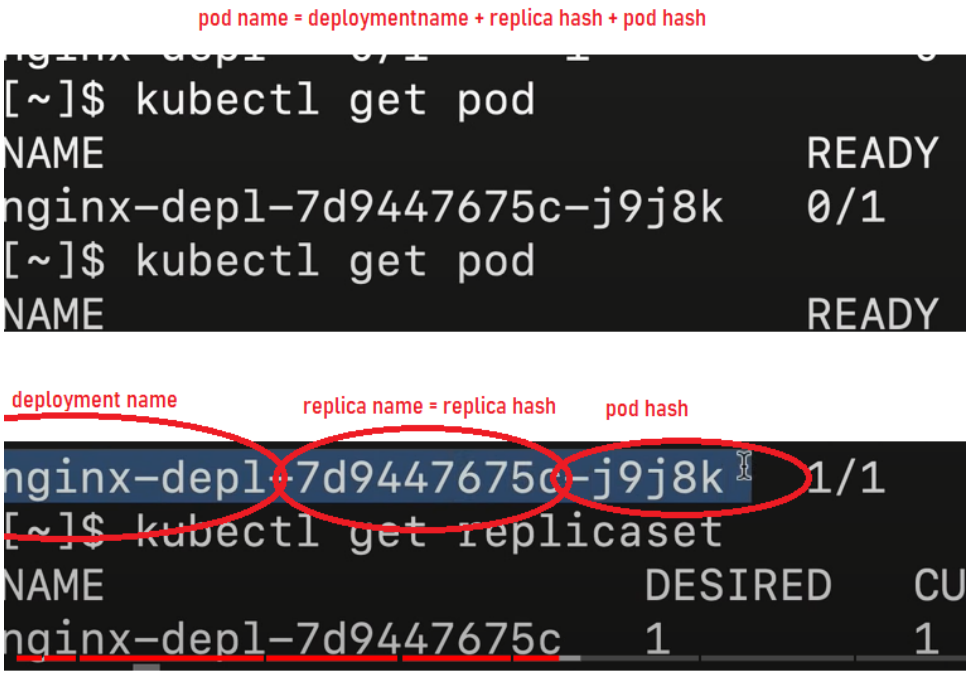
Edit the deployment 'mydeployment' in YAML and save the modified config in its annotation

kubectl edit deployment/mydeployment -o yaml --save-config

replicaset

Another abstract layer between deployment and pod:

Deployment -> replicaset -> pod :



Debugging logs – same as in docker

Kubectl logs <pod name>

Kubectl describe <pod-name>

To solve tty Error: Unable to use a TTY - input is not a terminal or the right kind of file

Run: $ alias kubectl='winpty kubectl'

Kubectl exec –it <pod name> -- bin/bash

Kubectl to use configuration file: apply & update

Kubectl apply –f <file name>

Example for nginx-deployment.yaml:

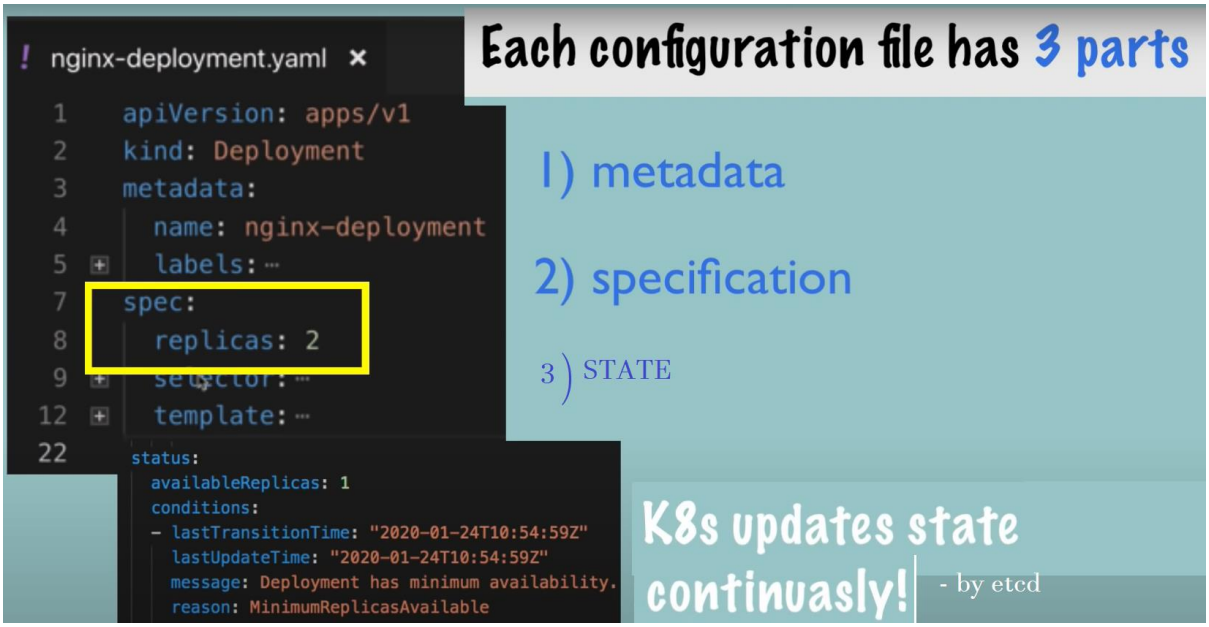
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 80
```

Kubectl apply –f nginx-deployment.yaml:

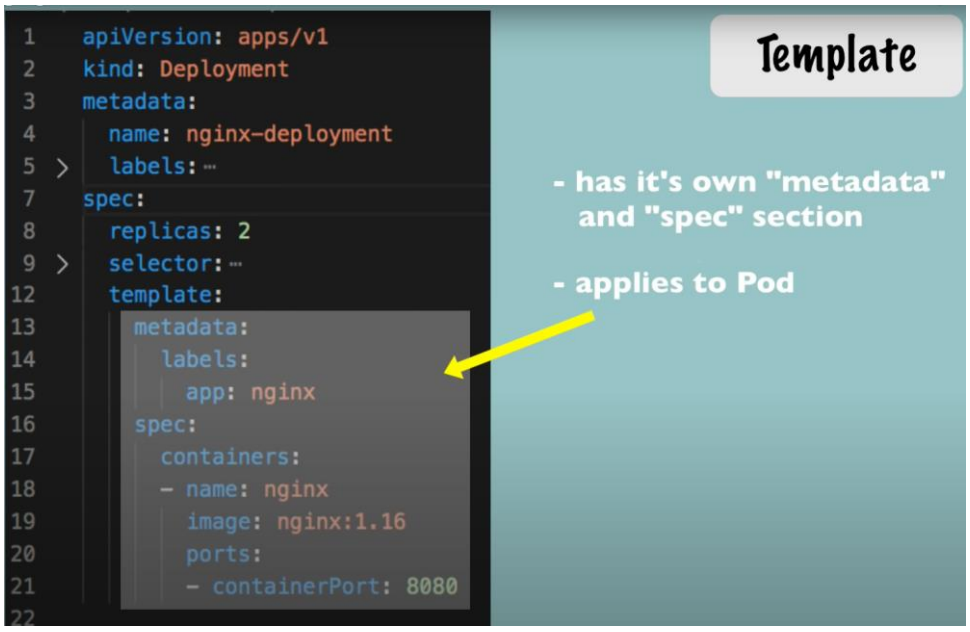
```
[~]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
[~]$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-594cc45b78-pq5dx   1/1     Running   0          7s
[~]$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1/1     1            1           52s
```

Configuration-file

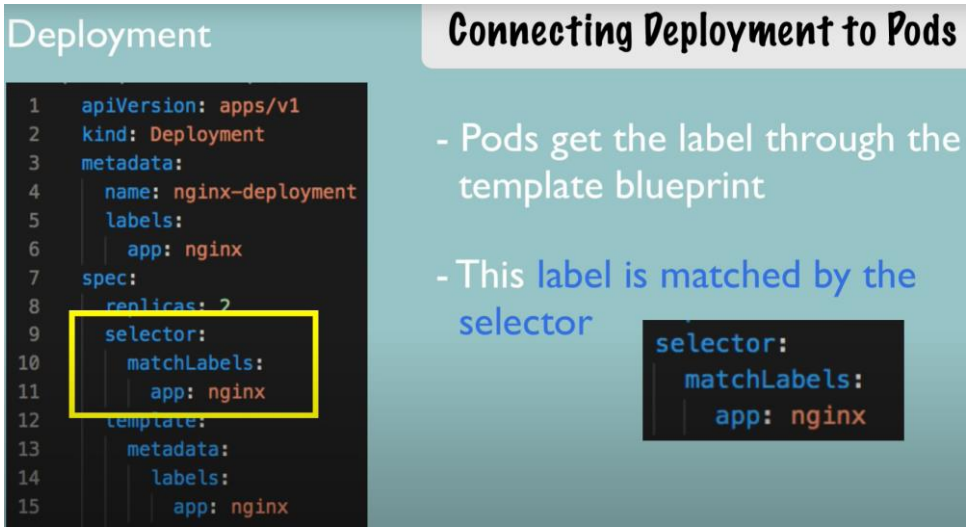
contains 3 parts

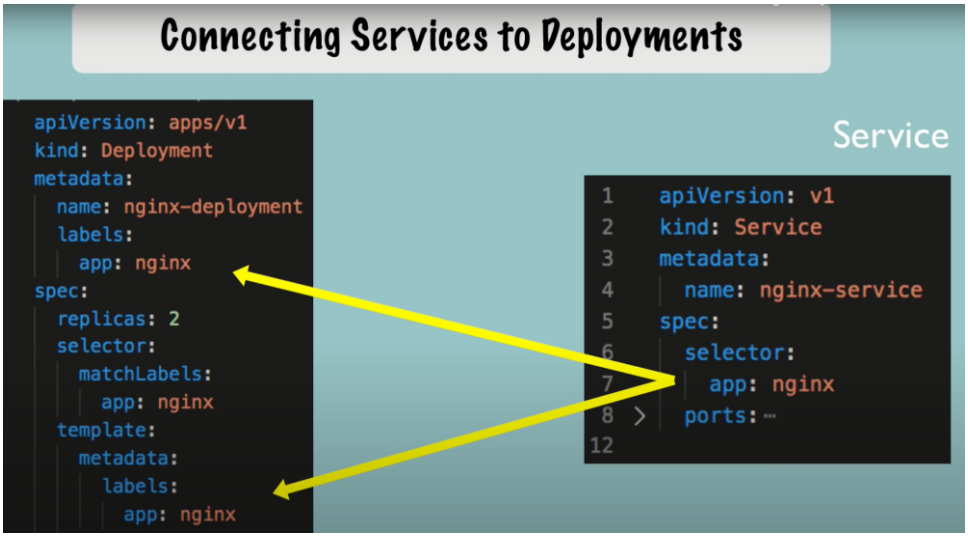


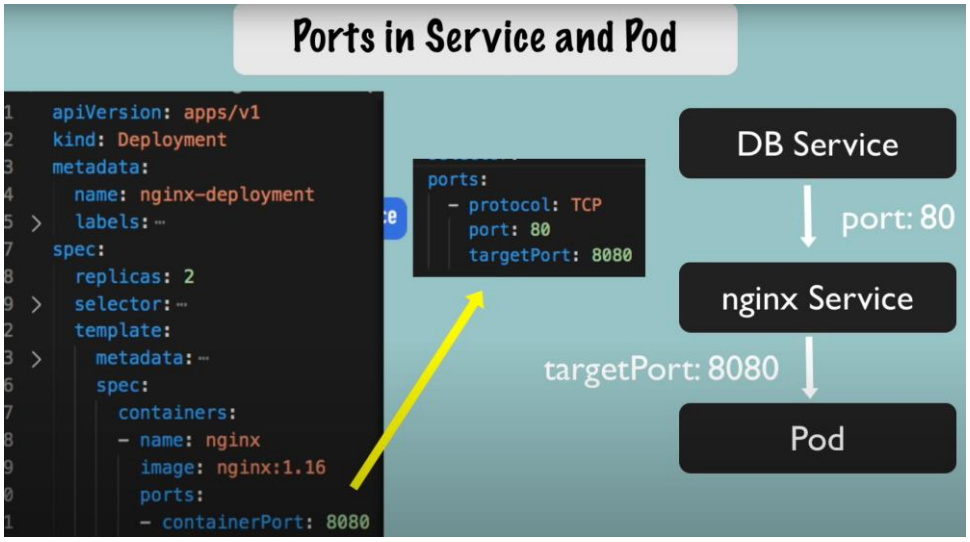
Template – apply to specific pod:

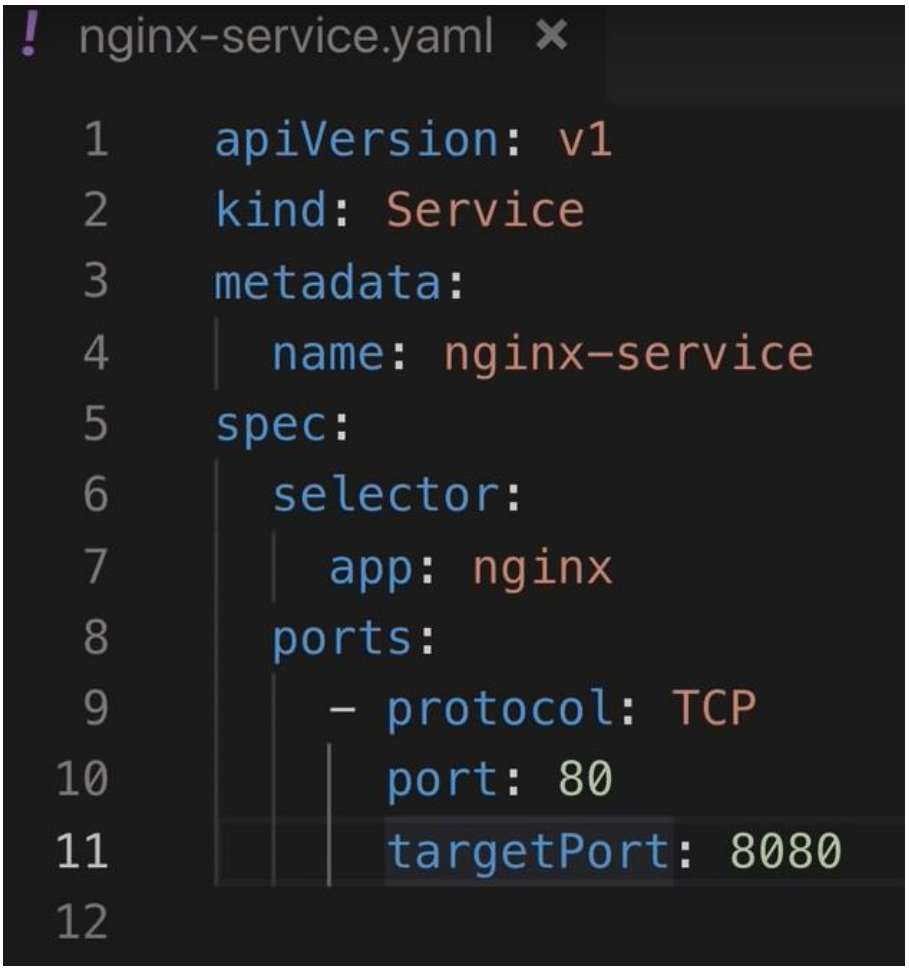


Connect service to pods to deployment









Service listen on port 80 and forward to pod port 8080:

Service listen on port 80:

```
[Documents]$ kubectl apply -f nginx-service.yaml
service/nginx-service created
[Documents]$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-7d64f4b574-fklxj   1/1     Running   0          10s
nginx-deployment-7d64f4b574-v7mwj   1/1     Running   0          10s
[Documents]$ kubectl get service
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP   2d
nginx-service   ClusterIP   10.96.25.229   <none>        80/TCP    19s
[Documents]$
```

forward to pod port 8080:

Kubectl describe service nginx-service:

```
kubernetes      ClusterIP   10.96.0.1
nginx-service   ClusterIP   10.96.25.22
[Documents]$ kubectl describe service n
Name:              nginx-service
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.i
                     {"apiVersion":"v1"
me":"nginx-service","namespace":"defaul
Selector:          app=nginx
Type:              ClusterIP
IP:                10.96.25.229
Port:              <unset>  80/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.6:8080,172.17.0.7:8080
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

How do we know that these are the correct ip addresses ? Using get pod –o wide:

```
Port:              <unset>  80/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.6:8080,172.17.0.7:8080
Session Affinity:  None
Events:            <none>
[Documents]$ kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP
E       NOMINATED NODE   READINESS GATES
nginx-deployment-7d64f4b574-fklxj   1/1     Running   0          2m17s   172.17.0.7
ikube   <none>           <none>
nginx-deployment-7d64f4b574-v7mwj   1/1     Running   0          2m17s   172.17.0.6
ikube   <none>           <none>
```

Get deployment status (from etcd): -o yaml

Get deployment nginx-deployment –o yaml > nginx-deployment=result.yaml

```
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2020-01-24T10:54:59Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: "2020-01-24T10:54:56Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: ReplicaSet "nginx-deployment-7d64f4b574" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
  updatedReplicas: 2
[Documents]$ kubectl get deployment nginx-deployment -o yaml > nginx-deployment-result.ya
ml
```

nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 8080
```

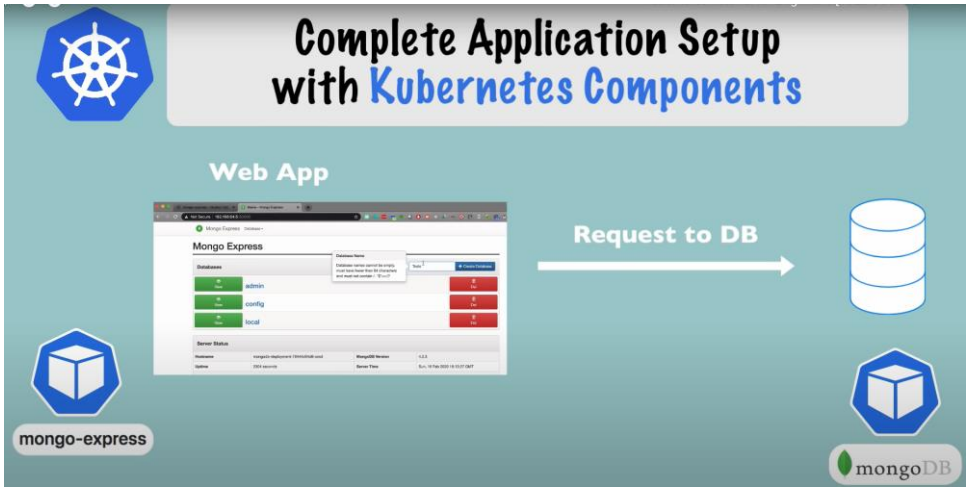
nginx-deployment-result.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata: …
spec: …
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2020-01-24T10:54:59Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: Deployment has minimum availabili
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: "2020-01-24T10:54:56Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: ReplicaSet "nginx-deployment-7d64
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
```

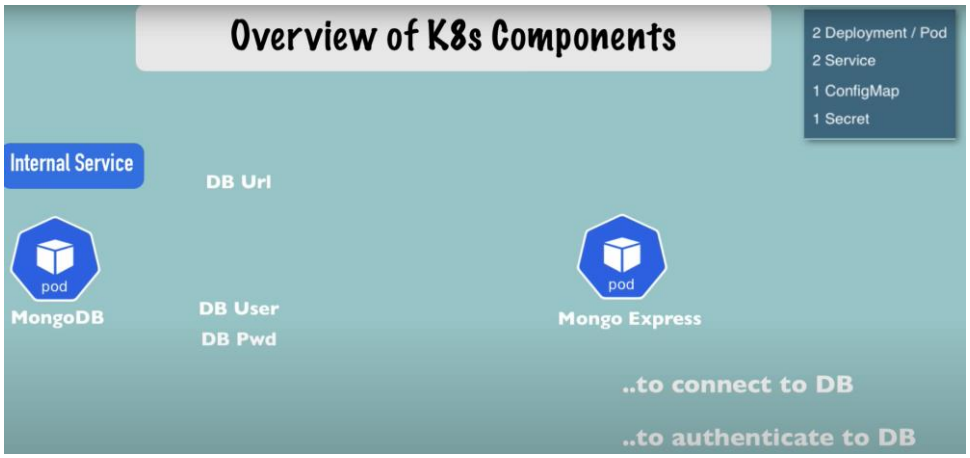
Delete and edit configuratuion files:

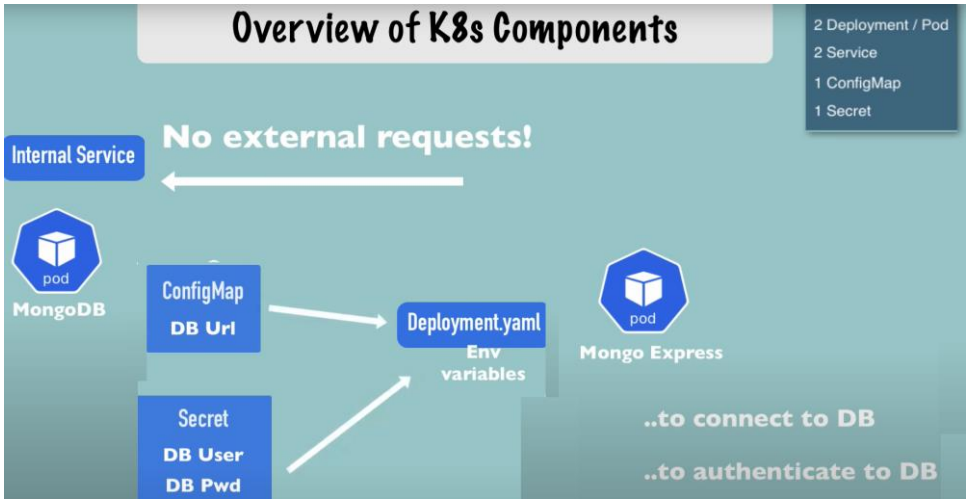
```
[Documents]$ kubectl get deployment nginx-deployment -o yaml > nginx-deplo
ml
[Documents]$ kubectl delete -f nginx-deployment.yaml
deployment.apps "nginx-deployment" deleted
```

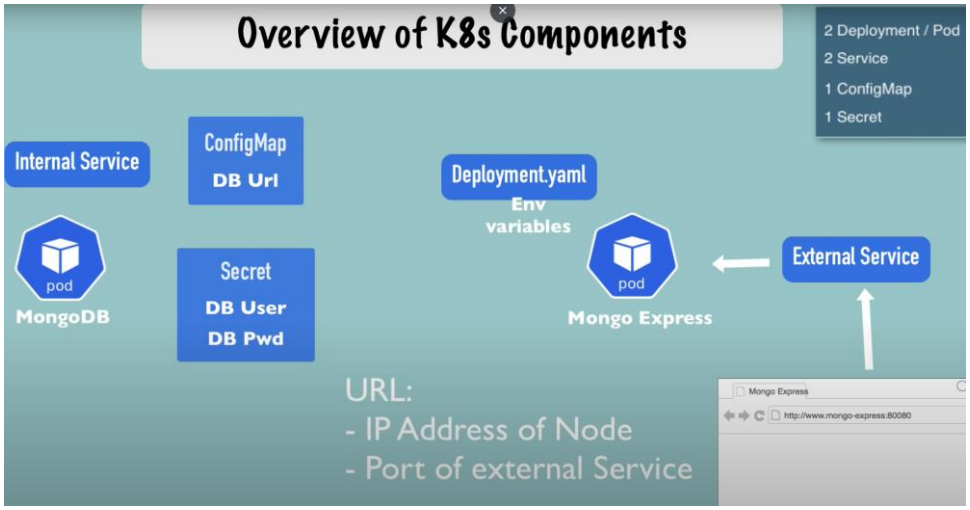
Demo on mongodb

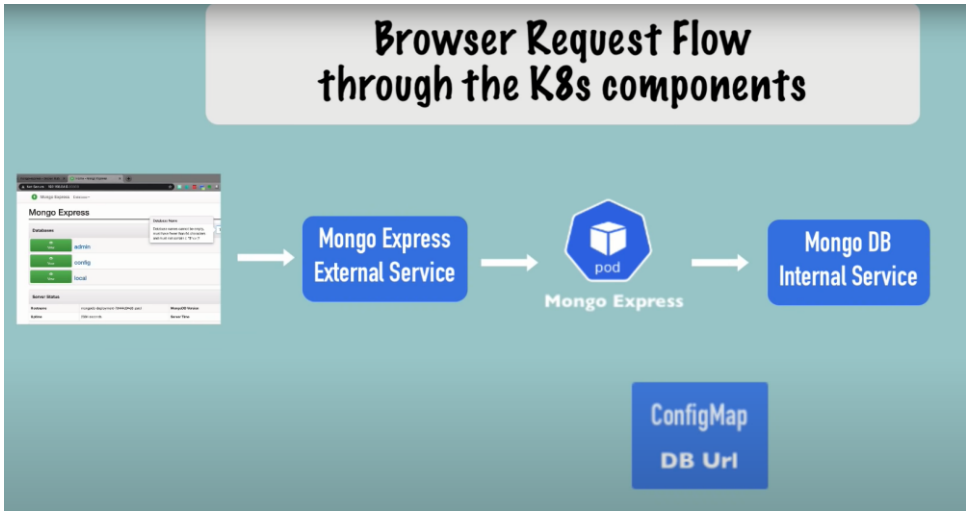


Architecture

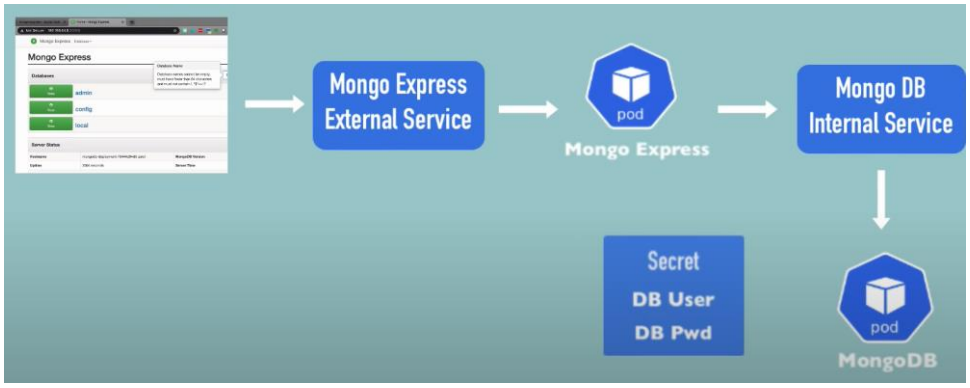






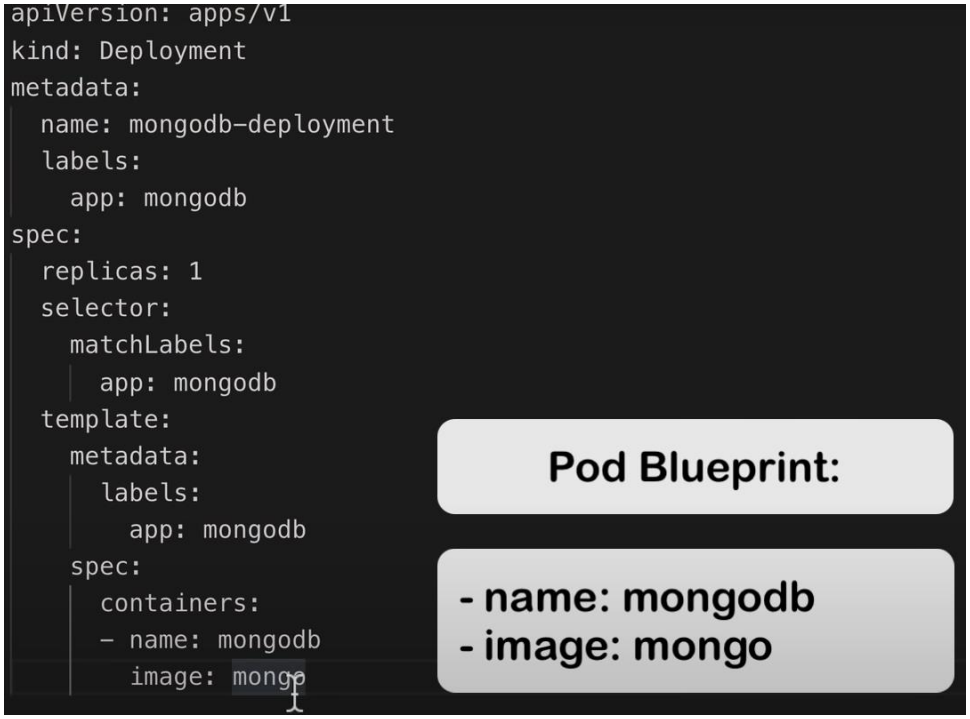


Authentication:

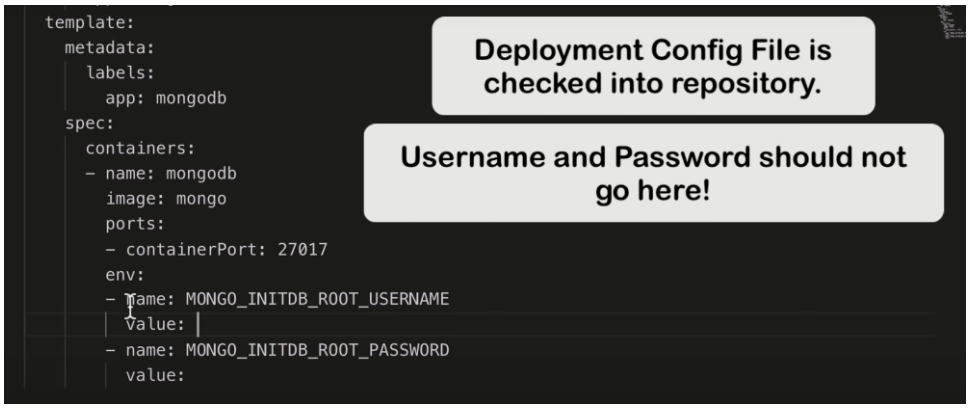


mongo.yaml deployment file;

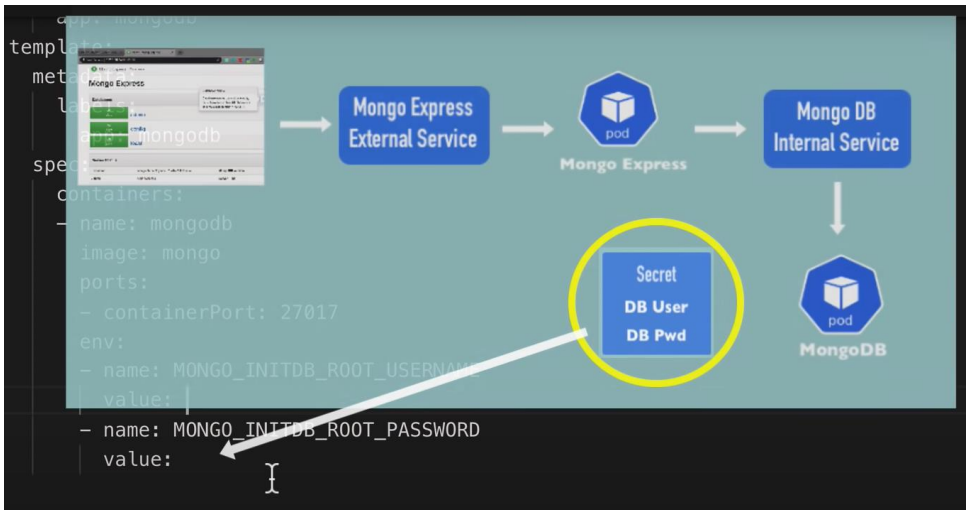
Good reference: https://devopscube.com/deploy-mongodb-kubernetes/



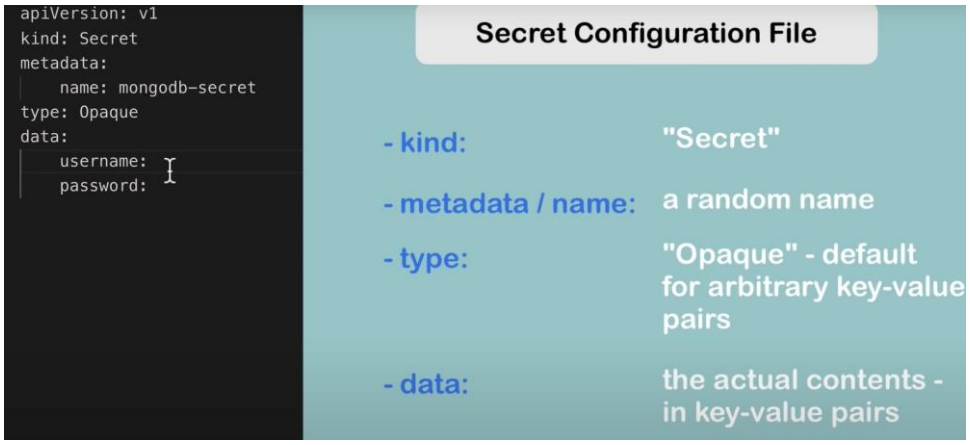
Getting ports and env variables from docker hub mongo image and af=ding to template pod:



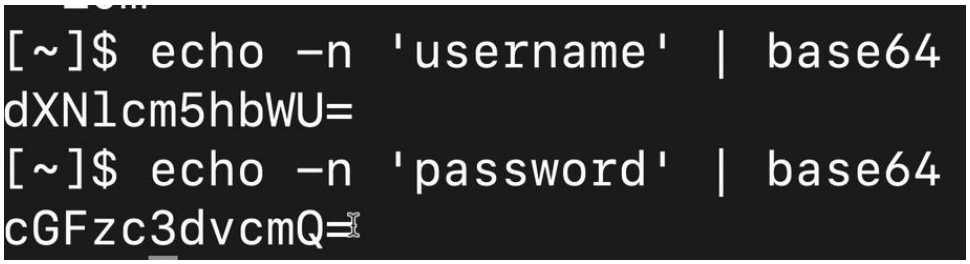
Creating secrets:

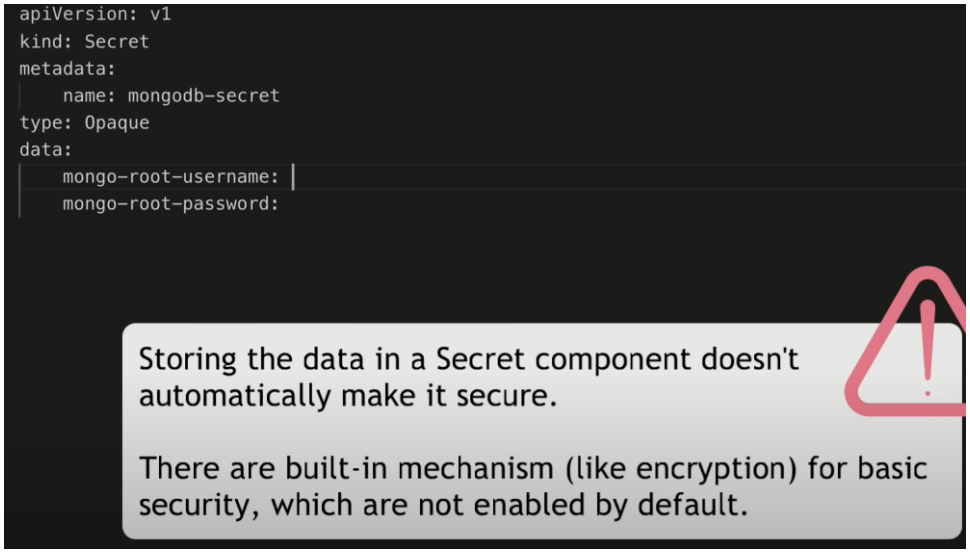


mongo-secret.yaml deployment file:



Adding value as base64 :



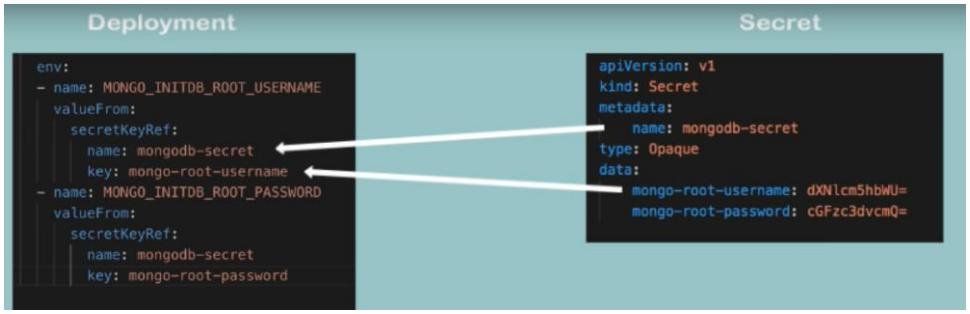


mongo.yaml | mongo-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
    name: mongodb-secret
type: Opaque
data:
    mongo-root-username: dXNlcm5hbWU=
    mongo-root-password: cGFzc3dvcmQ=
```

```
[~]$ cd k8s-configuration/
[k8s-configuration]$ ls
mongo-secret.yaml       mongo.yaml
[k8s-configuration]$ kubectl apply -f mongo-secret.yaml
secret/mongodb-secret created
[k8s-configuration]$
[k8s-configuration]$ kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-4clzs   kubernetes.io/service-account-token   3      22m
mongodb-secret        Opaque                                2      68s
[k8s-configuration]$
```

Secret can be referenced now in Deployment



mongo.yaml ✕ mongo-secret.yaml

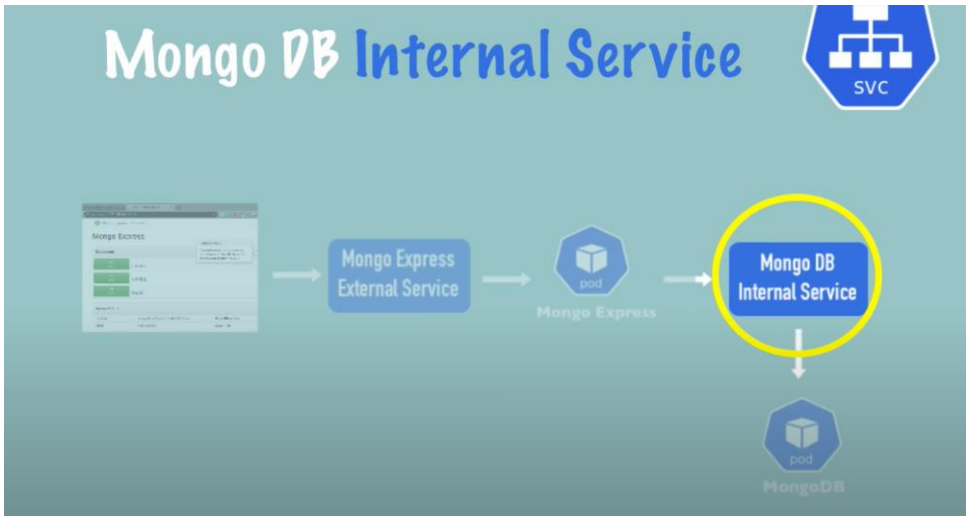
```yaml
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo
        ports:
        - containerPort: 27017
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-username
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-password
```

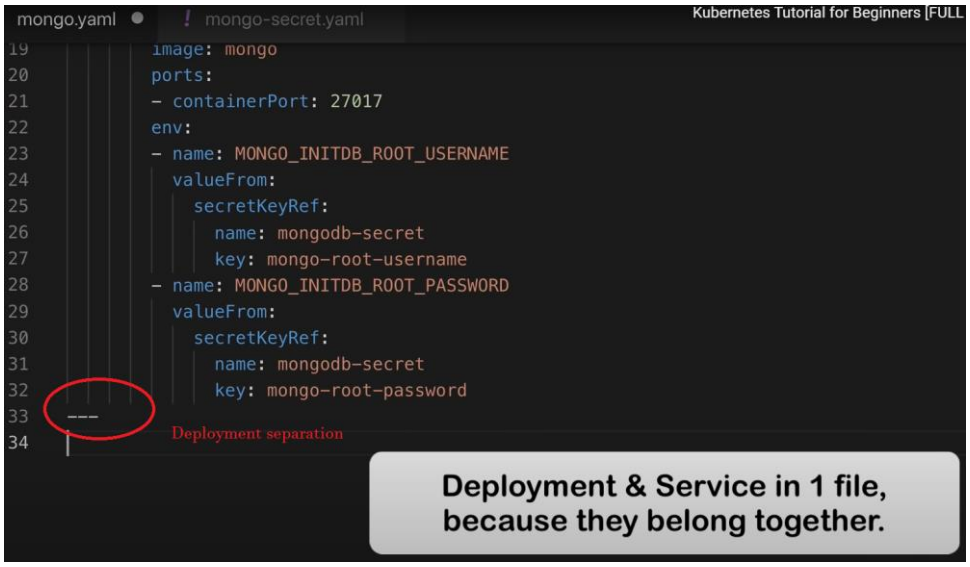
Mongo db Deployment +

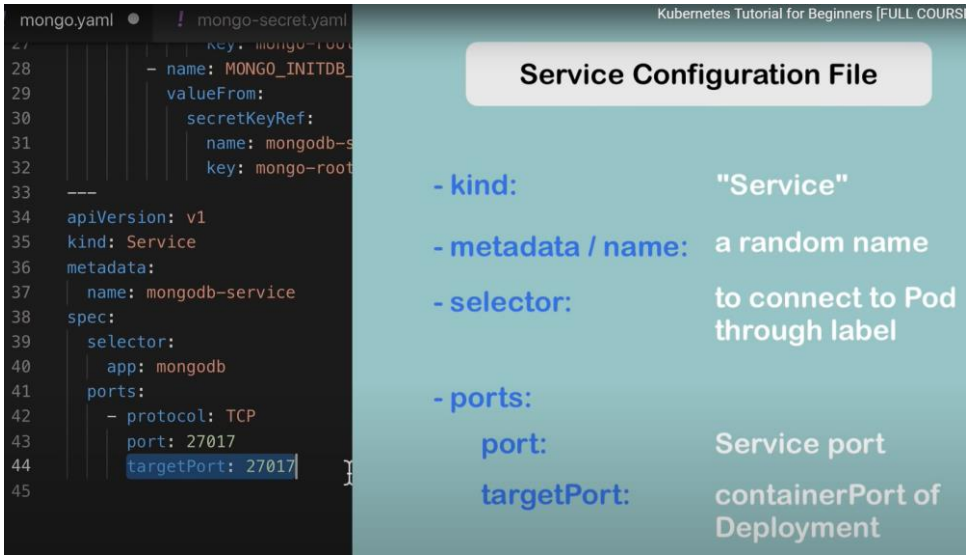
Get deployment, replicaset and pod:

```
[k8s-configuration]$ kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-4clzs   kubernetes.io/service-account-token   3      22m
mongodb-secret        Opaque                                2      68s
[k8s-configuration]$ kubectl apply -f mongo.yaml
deployment.apps/mongodb-deployment created
[k8s-configuration]$ kubectl get all
NAME                                     READY   STATUS              RESTARTS
  AGE
pod/mongodb-deployment-78444d94d6-zsrcl   0/1     ContainerCreating   0
  7s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   25m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb-deployment   0/1     1            0           7s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-deployment-78444d94d6   1         1         0       7s
```

Mongo db internal service inside same deployment file:







```
[k8s-configuration]$ kubectl apply -f mongo.yaml
deployment.apps/mongodb-deployment unchanged
service/mongodb-service created
[k8s-configuration]$
```

Service is listening on 27017 (mongo port):

```
[k8s-configuration]$ kubectl get service
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP     36m
mongodb-service   ClusterIP   10.96.86.105    <none>        27017/TCP   51s
[k8s-configuration]$
```

Make sure service is connecting to the pods (using describe on the service and get pod –o wide):

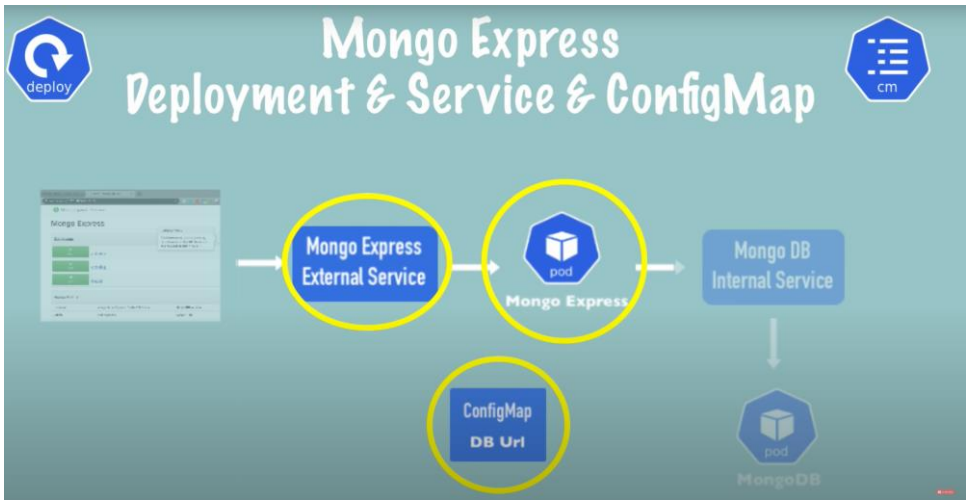
```
[k8s-configuration]$ kubectl get service
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP     36m
mongodb-service   ClusterIP   10.96.86.105    <none>        27017/TCP   51s
[k8s-configuration]$ kubectl describe service mongodb-service
Name:              mongodb-service
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotatio
ns":{},"name":"mongodb-service","namespace":"default"},"spec":{"ports":[{"port"
:...
Selector:          app=mongodb
Type:              ClusterIP
IP:                10.96.86.105
Port:              <unset>  27017/TCP
TargetPort:        27017/TCP
Endpoints:         172.17.0.6:27017
Session Affinity:  None
Events:            <none>
```

```
Selector:          app=mongodb
Type:              ClusterIP
IP:                10.96.86.105
Port:              <unset>  27017/TCP
TargetPort:        27017/TCP
Endpoints:         172.17.0.6:27017
Session Affinity:  None
Events:            <none>
[k8s-configuration]$ kubectl get pod -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP
      NODE       NOMINATED NODE   READINESS GATES
mongodb-deployment-78444d94d6-zsrcl   1/1     Running   0          12m   172.17
.0.6   minikube   <none>           <none>
```

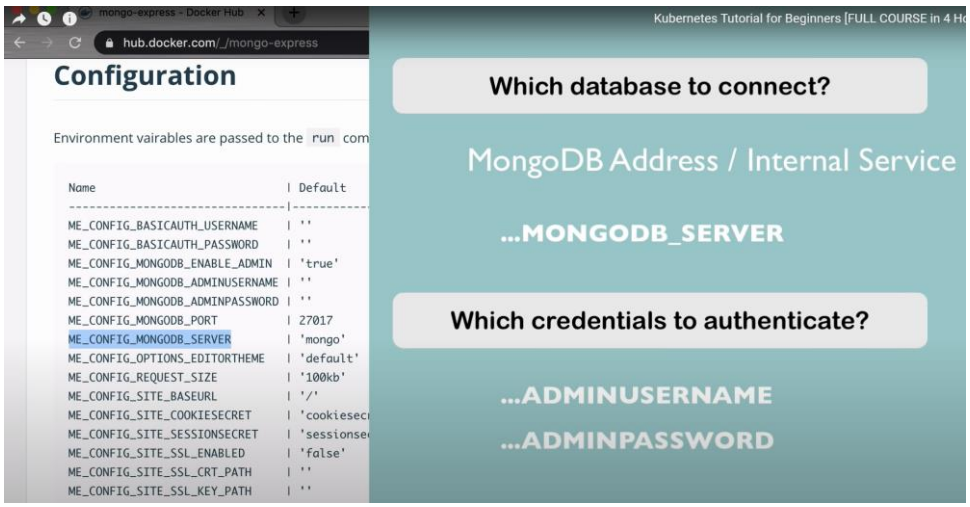
Summarize the mongo service / pod deployment / replicaset:

```
[k8s-configuration]$ kubectl get all | grep mongodb
pod/mongodb-deployment-78444d94d6-zsrcl   1/1     Running   0          13m
service/mongodb-service   ClusterIP   10.96.86.105   <none>        27017/TCP
2m52s
deployment.apps/mongodb-deployment   1/1     1            1           13m
replicaset.apps/mongodb-deployment-78444d94d6   1         1         1       13m
```

mongo express deployment , service and configMap:



Environment vars for mongo-express (from docker-hub):

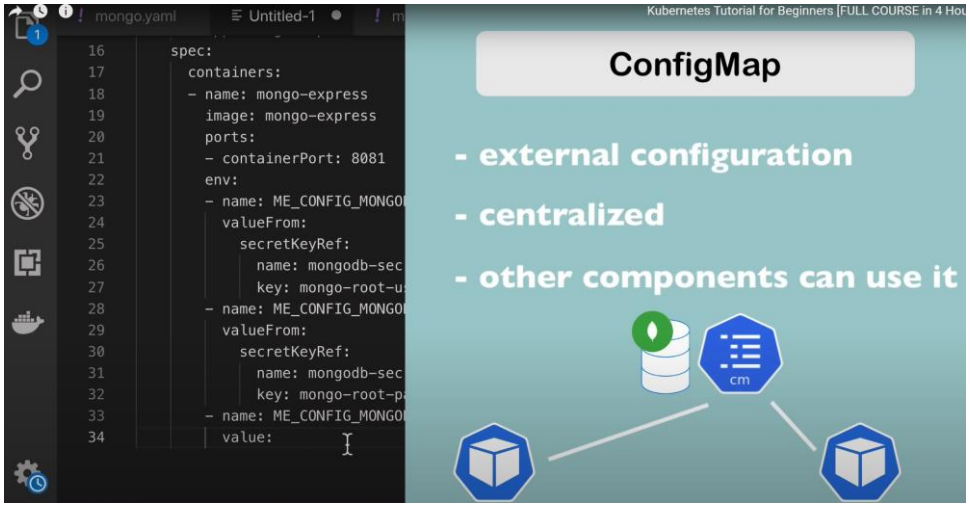


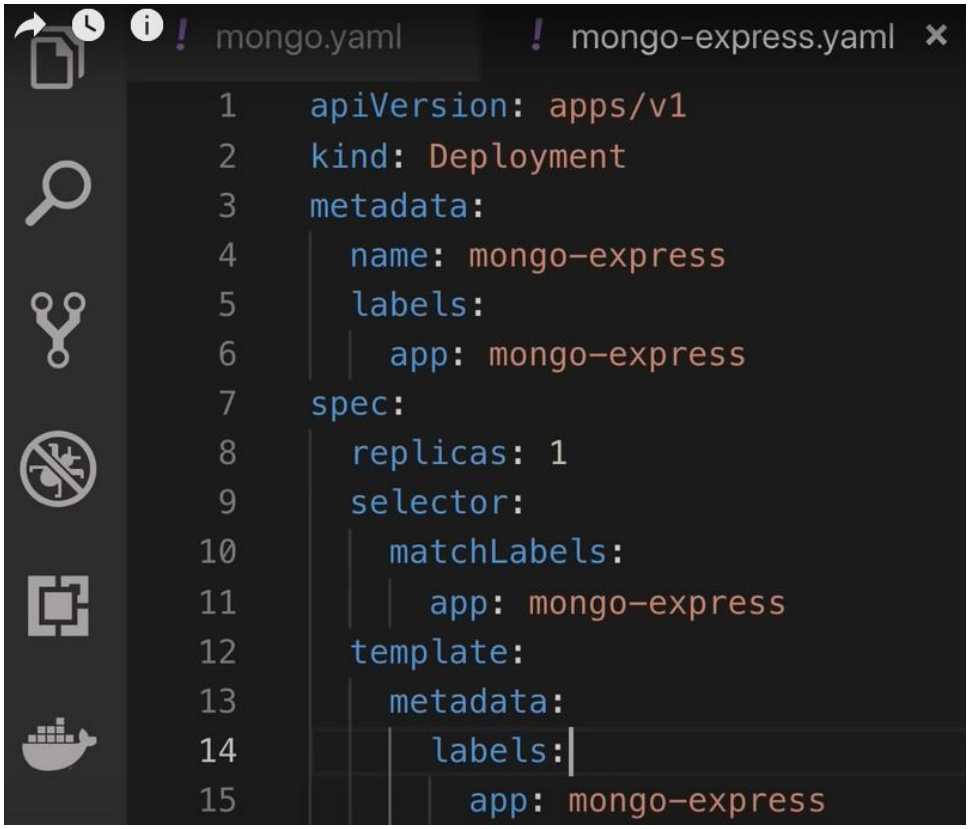
Port: 8081

Mongo-express.yanl config file:

Secrets should be same secrets as we use in mongo.yaml file.

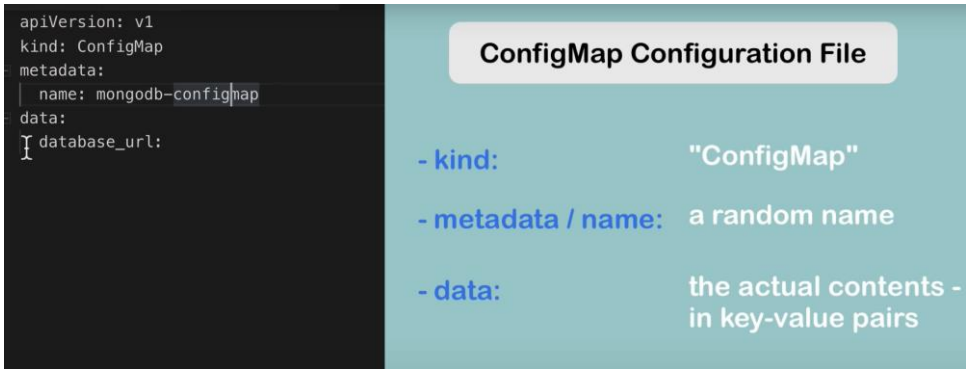
ConfigMap is being shared between several applications:


mongo.yaml
Untitled-1
Kubernetes Tutorial for Beginners [FULL COURSE in 4 Hou
spec:
containers:
- name: mongo-express
image: mongo-express
ports:
- containerPort: 8081
env:
- name: ME_CONFIG_MONGO
valueFrom:
secretKeyRef:
name: mongodb-sec
key: mongo-root-u
- name: ME_CONFIG_MONGO
valueFrom:
secretKeyRef:
name: mongodb-sec
key: mongo-root-p
- name: ME_CONFIG_MONGO
value:
ConfigMap
- external configuration
- centralized
- other components can use it
cm


mongo.yaml
mongo-express.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
name: mongo-express
labels:
app: mongo-express
spec:
replicas: 1
selector:
matchLabels:
app: mongo-express
template:
metadata:
labels:
app: mongo-express

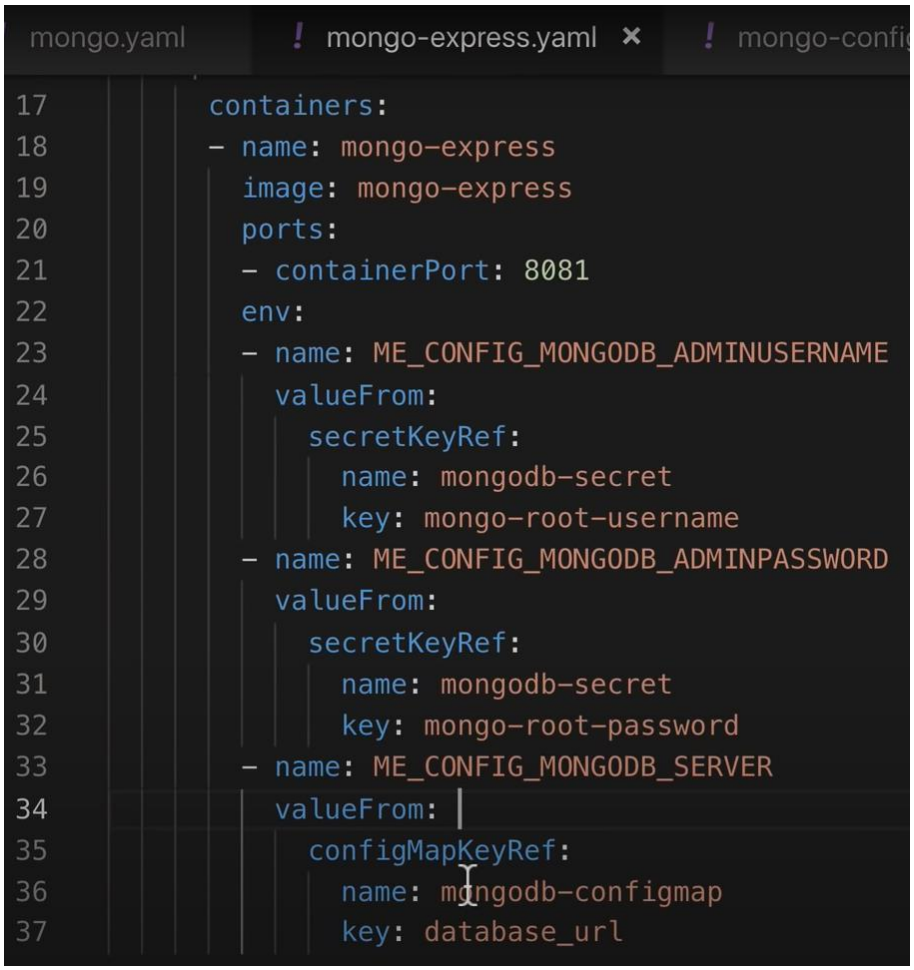
```yaml
      app: mongo-express
  spec:
    containers:
    - name: mongo-express
      image: mongo-express
      ports:
      - containerPort: 8081
      env:
      - name: ME_CONFIG_MONGODB_ADMINUSERNAME
        valueFrom:
          secretKeyRef:
            name: mongodb-secret
            key: mongo-root-username
      - name: ME_CONFIG_MONGODB_ADMINPASSWORD
        valueFrom:
          secretKeyRef:
            name: mongodb-secret
            key: mongo-root-password
      - name: ME_CONFIG_MONGODB_SERVER
        value:
```

Mongo-configmap.yml – does not have a type as we have just one config map:

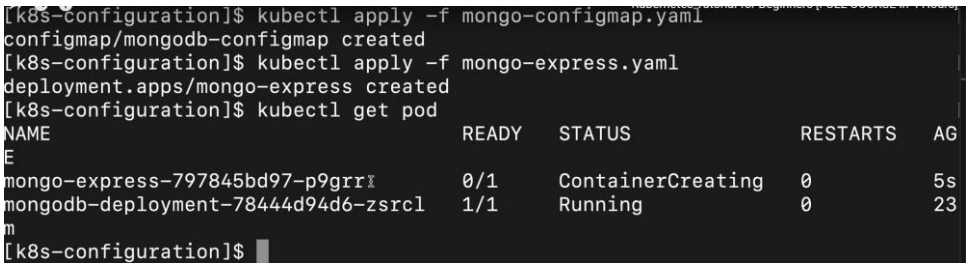


Database-url is the name of the service !!

mongo-express.yml with configmap:



Create config-map first, and mongo-express

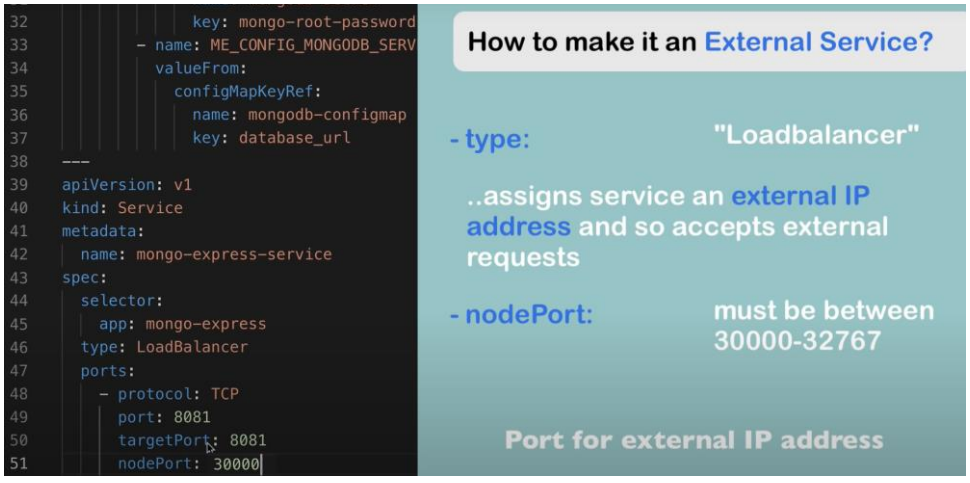


Make sure mongo-express is up and running”:

```
[k8s-configuration]$ kubectl get pod
NAME                                  READY   STATUS    RESTARTS   AGE
mongo-express-797845bd97-p9grr        1/1     Running   0          7m30s
mongodb-deployment-78444d94d6-zsrcl   1/1     Running   0          31m
[k8s-configuration]$ kubectl logs mongo-express-797845bd97-p9grr
Waiting for mongodb-service:27017...
Welcome to mongo-express
------------------------


Mongo Express server listening at http://0.0.0.0:8081
Server is open to allow connections from anyone (0.0.0.0)
basicAuth credentials are "admin:pass", it is recommended you change this in yo
ur config.js!
Database connected
Admin Database connected
[k8s-configuration]$
```

Mongo express external service

Mongo-express additional deployment:

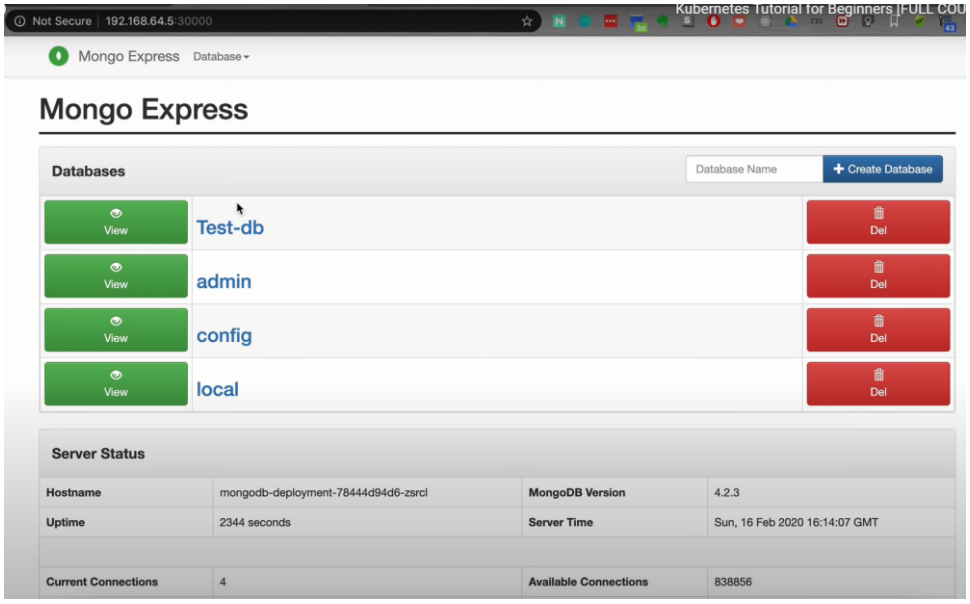


```
[k8s-configuration]$ kubectl apply -f mongo-express.yaml
deployment.apps/mongo-express unchanged
service/mongo-express-service created
[k8s-configuration]$ kubectl get service
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
kubernetes              ClusterIP      10.96.0.1       <none>        443/TCP
      62m
mongo-express-service   LoadBalancer   10.96.178.16    <pending>     8081:30000/
TCP   6s
mongodb-service         ClusterIP      10.96.86.105    <none>        27017/TCP
      26m
[k8s-configuration]$
```

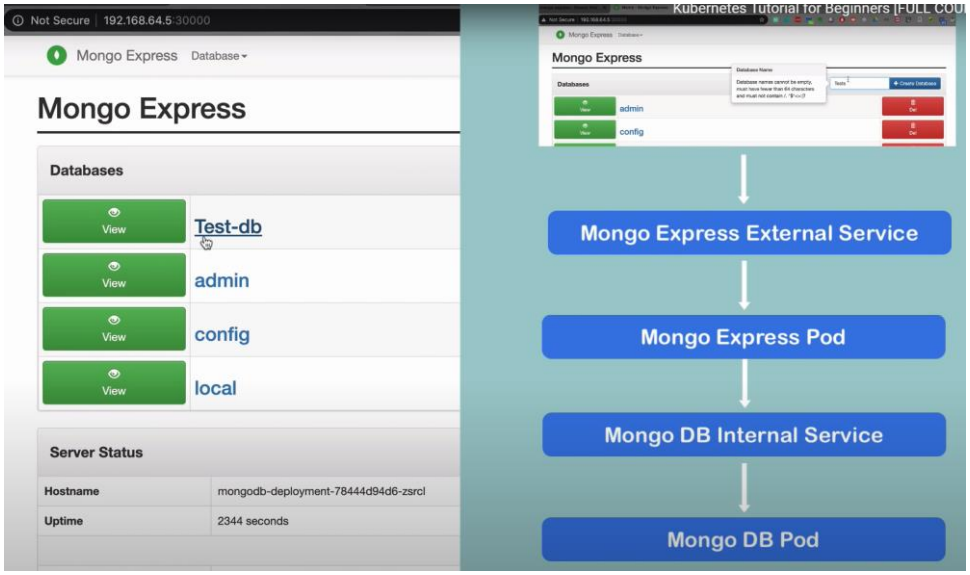
In actual k8s cluster , we will get external ip address istead of <pending>.

In minikube, we will need to set it by ourselves:

```
[k8s-configuration]$ kubectl get service
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
kubernetes              ClusterIP      10.96.0.1       <none>        443/TCP
      67m
mongo-express-service   LoadBalancer   10.96.178.16    <pending>     8081:30000/
TCP   5m7s
mongodb-service         ClusterIP      10.96.86.105    <none>        27017/TCP
      32m
[k8s-configuration]$ minikube service mongo-express-service
|-----------|-----------------------|-------------|---------------------------|
| NAMESPACE |         NAME          | TARGET PORT |            URL            |
|-----------|-----------------------|-------------|---------------------------|
| default   | mongo-express-service |             | http://192.168.64.5:30000 |
|-----------|-----------------------|-------------|---------------------------|
   Opening service default/mongo-express-service in default browser...
[k8s-configuration]$
```

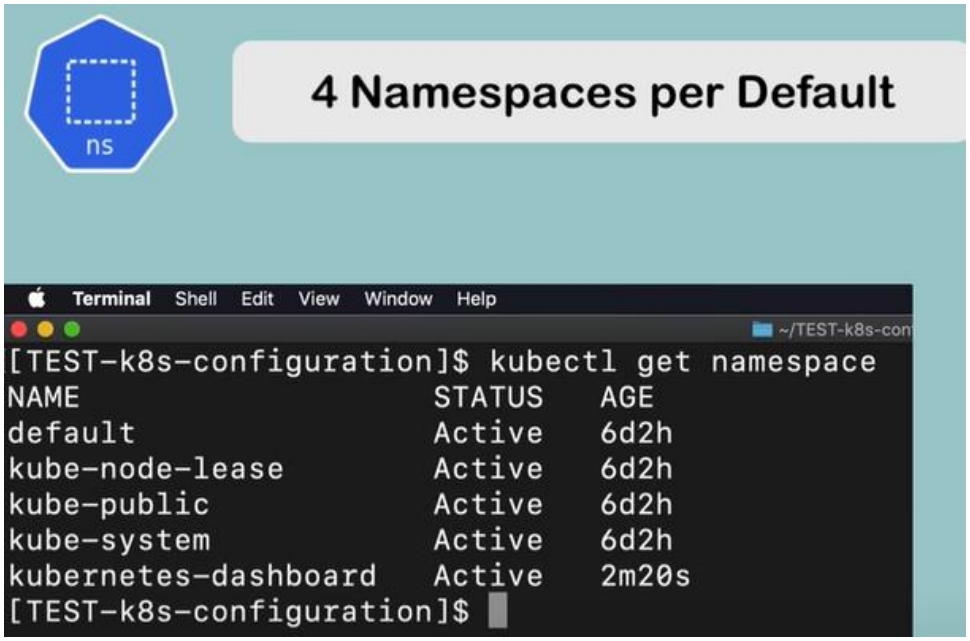


Execution flow



Namespaces

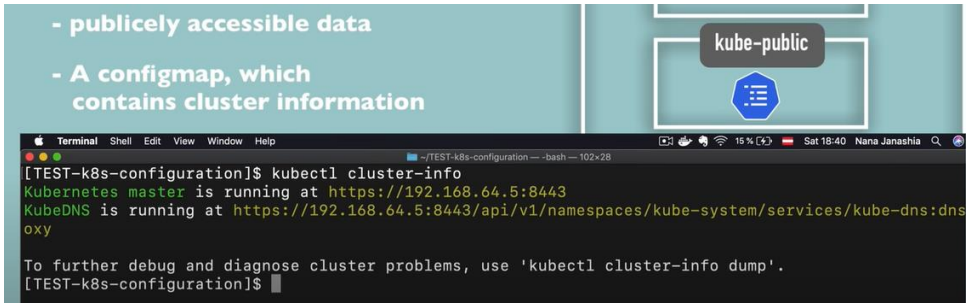
Its like virtual cluster inside cluster



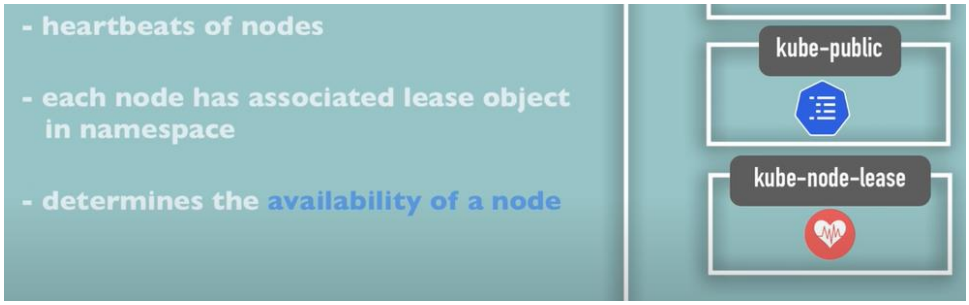
Kubernetes-dashboard – ONLY in mimikube

Kube-system: for master and kubectl / etc – DO NOT use it

Kube-public: configMap and other public data



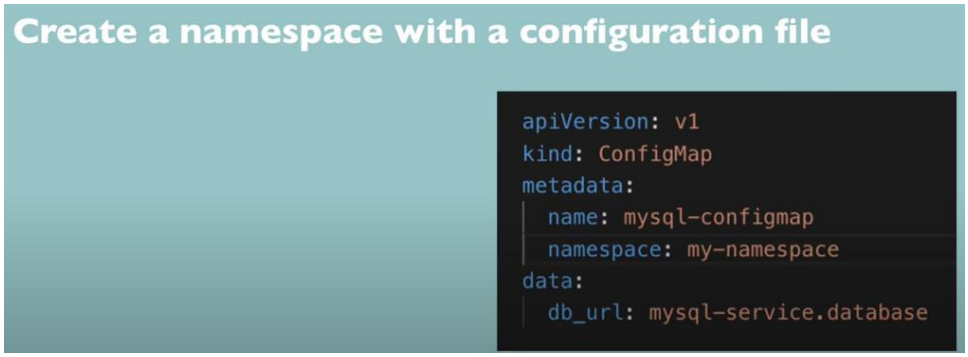
Kube-node-lease: heartbit of all nodes

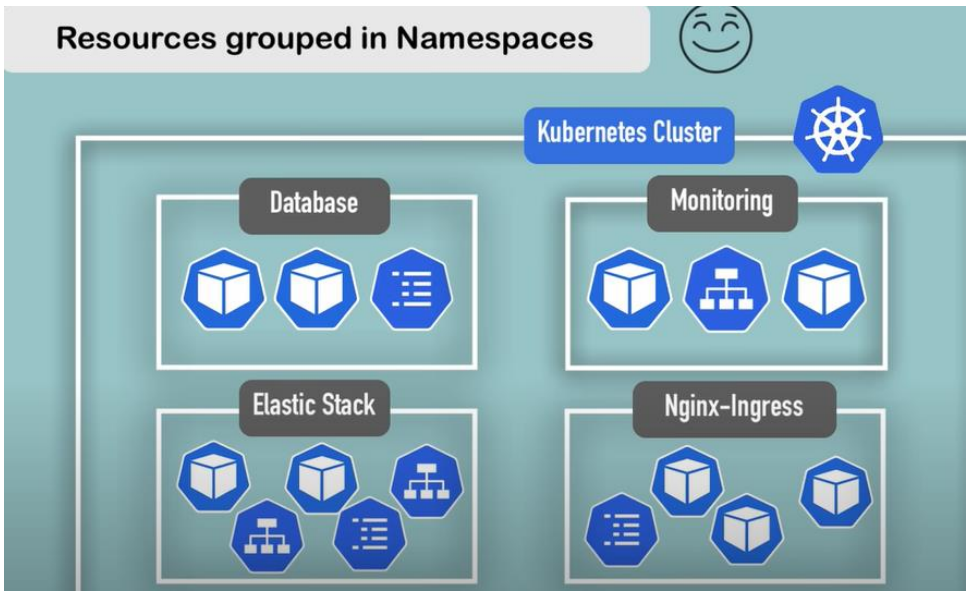


Default: for my resources

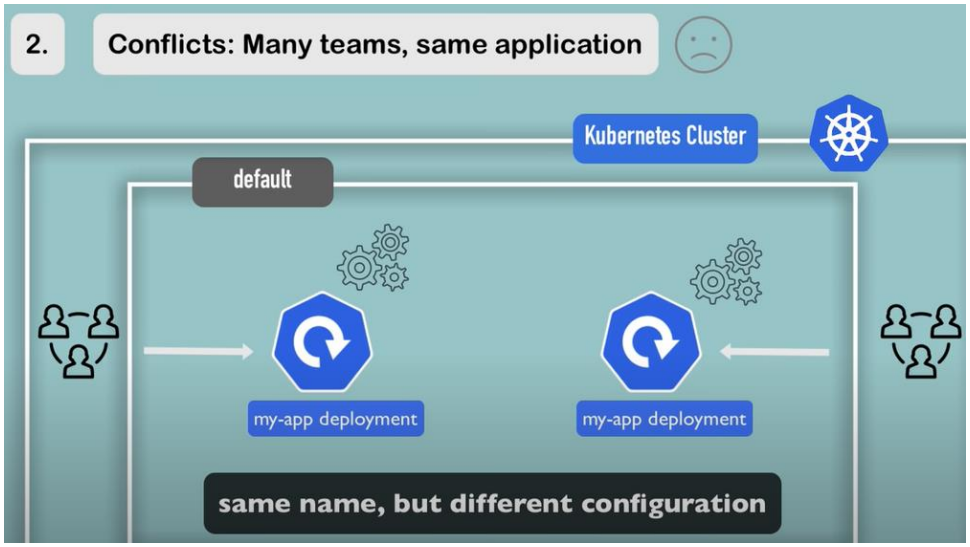
To create namesapce:

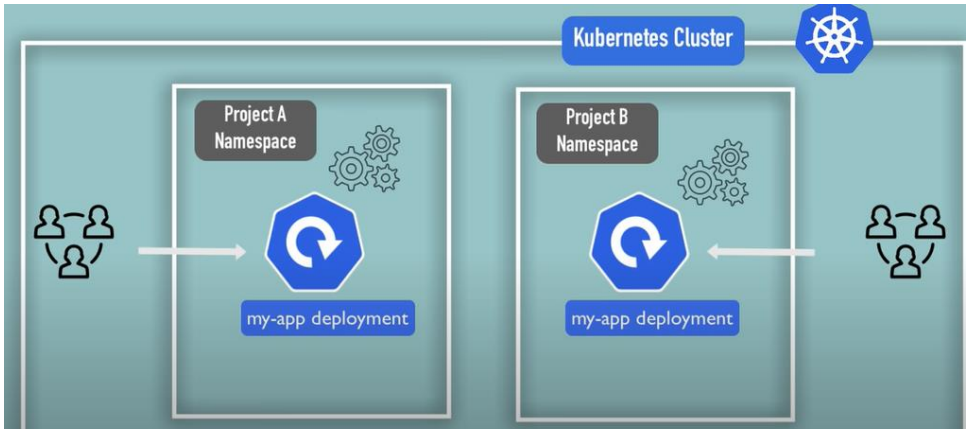
Kubectl create namespace my-namespace

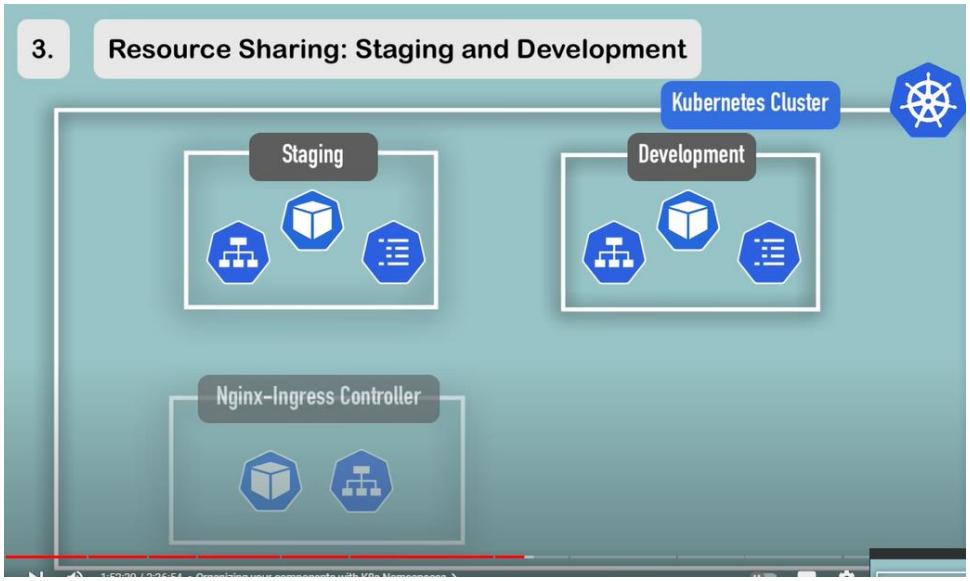


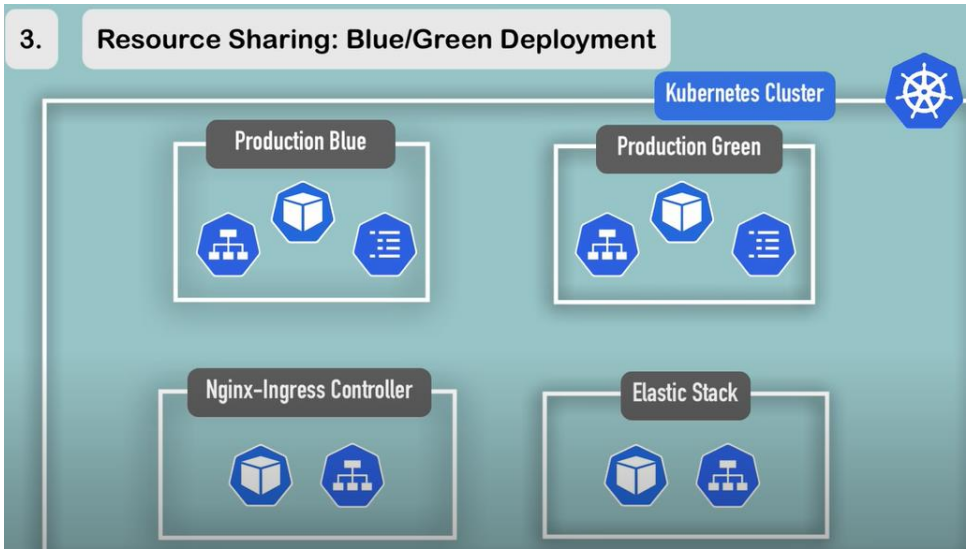


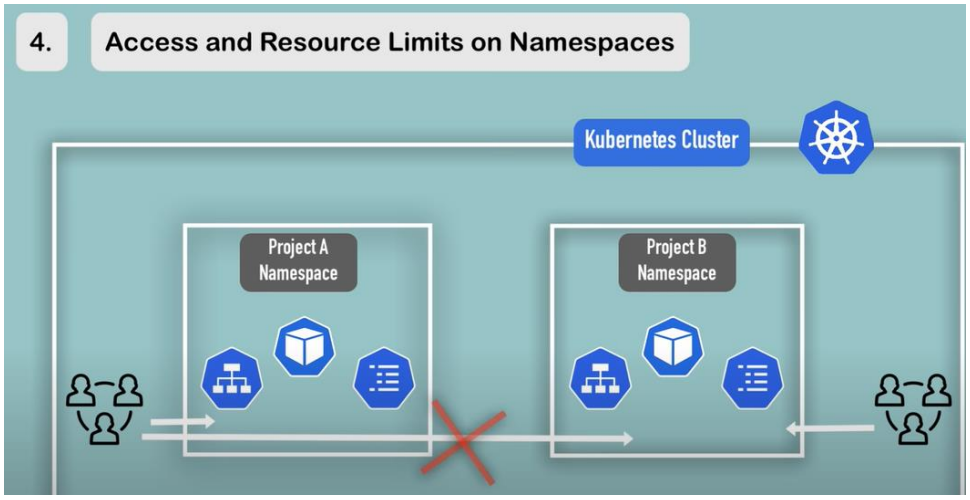
Putting same name application in different namespace:

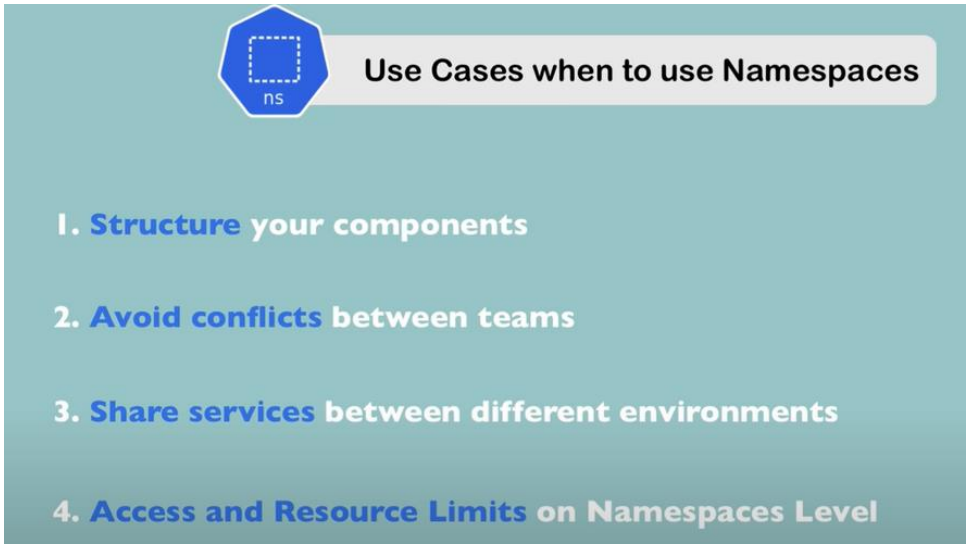


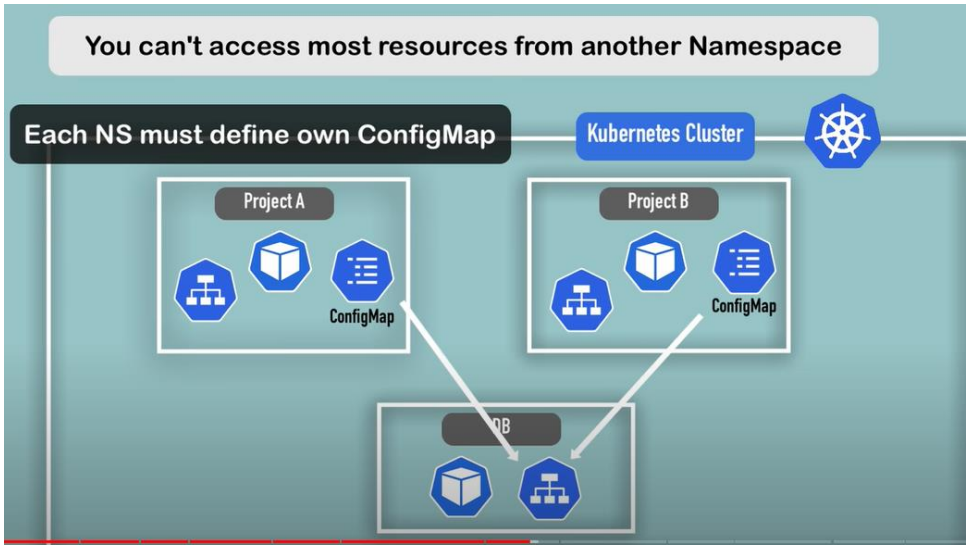




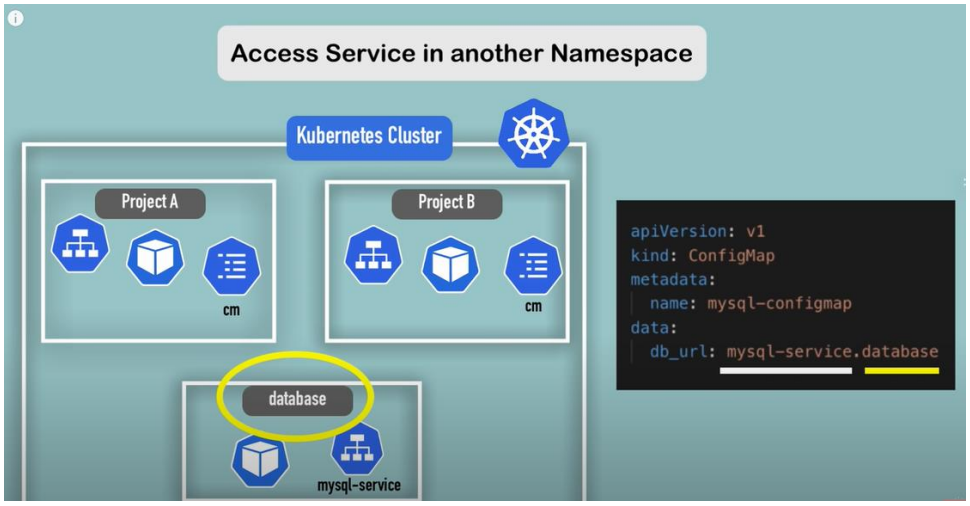

3.
Resource Sharing: Blue/Green Deployment
Kubernetes Cluster
Production Blue
Production Green
Nginx-Ingress Controller
Elastic Stack


4.
Access and Resource Limits on Namespaces
Kubernetes Cluster
Project A Namespace
Project B Namespace


ns
Use Cases when to use Namespaces
1. Structure your components
2. Avoid conflicts between teams
3. Share services between different environments
4. Access and Resource Limits on Namespaces Level



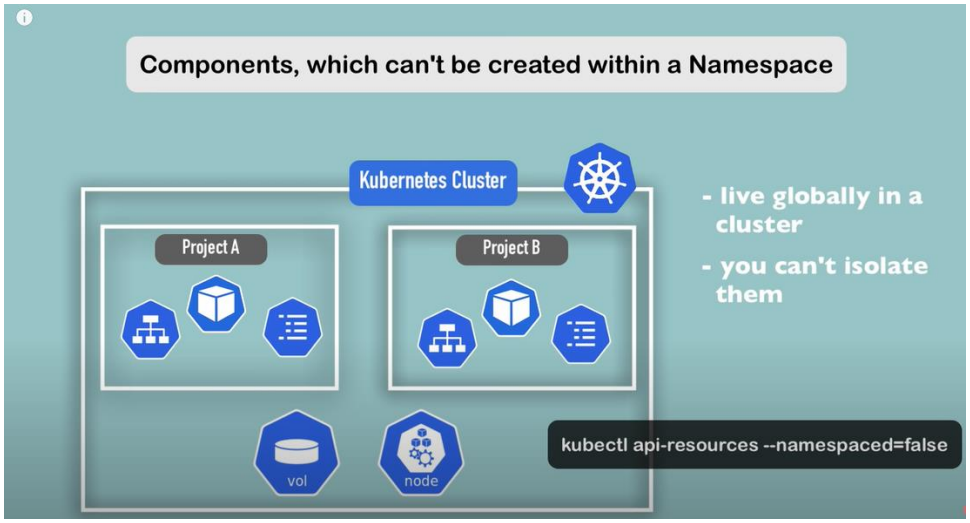
And secret file



Node and vol components cannot live inside namespace

To get the global components:

Kubectl api-resources –namecpased=false



Creare component inside namespace:

Kubectl apply –f mysql-configmap.yaml --namespace=my-namespace

Inside yaml file: (better documented and in automatic way, will be deployed correctly!!)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-configmap
  namespace: my-namespace
data:
  db_url: mysql-service.database
```

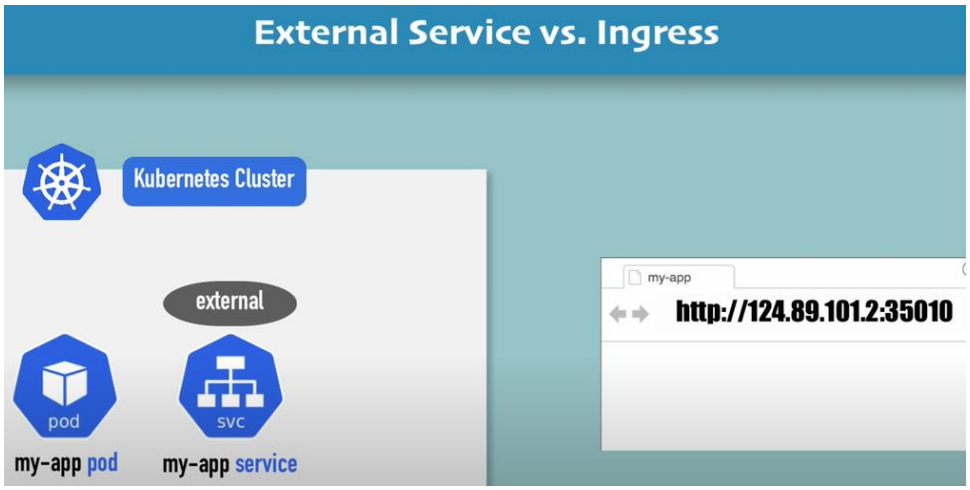
Kubectx – 3rd pary tool to change active namespace from default to different one

Inside it there is executables: kubens to change active namespace

```
[~]$ kubens
default
kube-node-lease
kube-public
kube-system
kubernetes-dashboard
my-namespace
[~]$ kubens my-namespace
Context "minikube" modified.
Active namespace is "my-namesp
[~]$ kubens
default
kube-node-lease
kube-public
kube-system
kubernetes-dashboard
my-namespace
[~]$
```

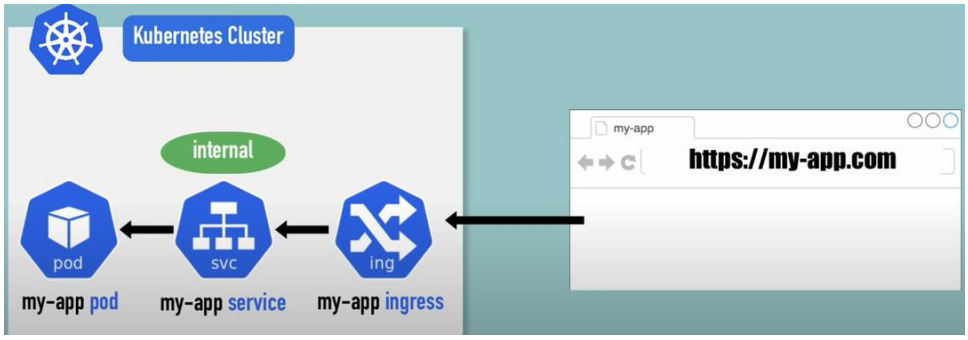
Ingress

External service good for tests / quick eployment



This is the target:

<Secure protocol> ://Domain-name



Example YAML File: External Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-external-service
spec:
  selector:
    app: myapp
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 35010
```

Assign external IP address to service

Example YAML File: Ingress

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: myapp-ingress
spec:
  rules:
  - host: myapp.com
    http:
      paths:
      - backend:
          serviceName: myapp-internal-service
          servicePort: 8080
```

kind: Ingress

Routing rules:

my-app
http://my-app.com

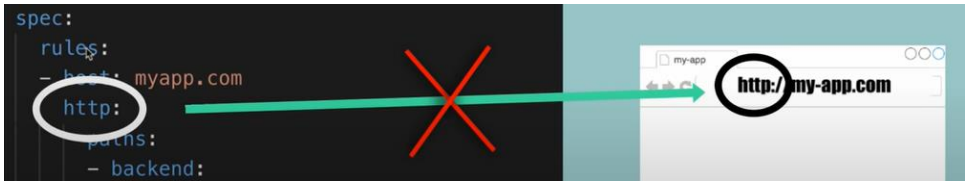
Forward request to the internal service.

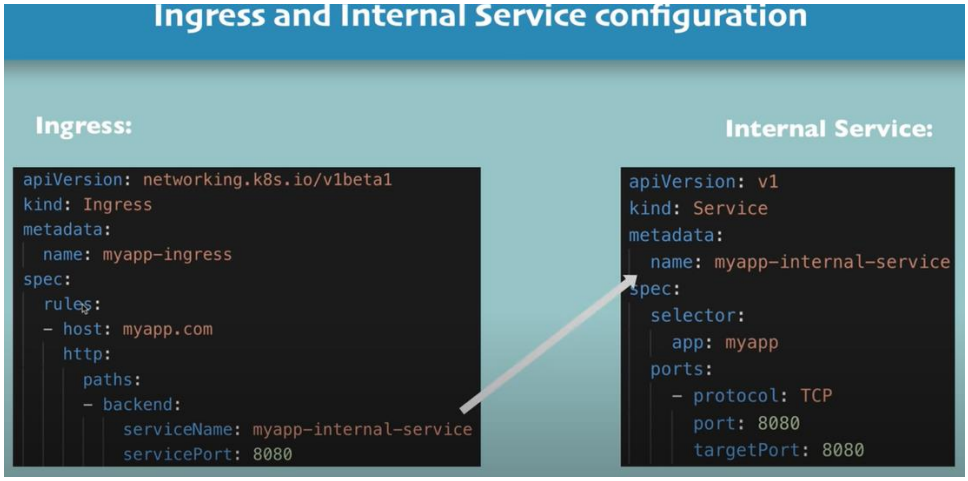
```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: myapp-ingress
spec:
  rules:
  - host: myapp.com
    http:
      paths:
      - backend:
          serviceName: myapp-internal-service
          servicePort: 8080
```

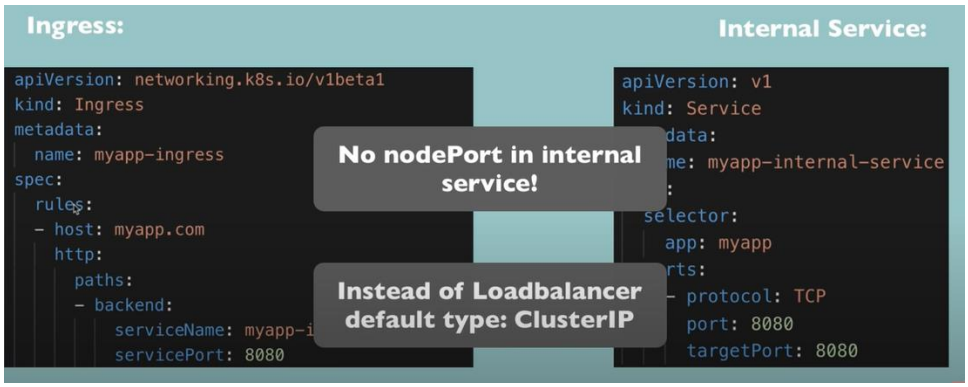
Routing rules:

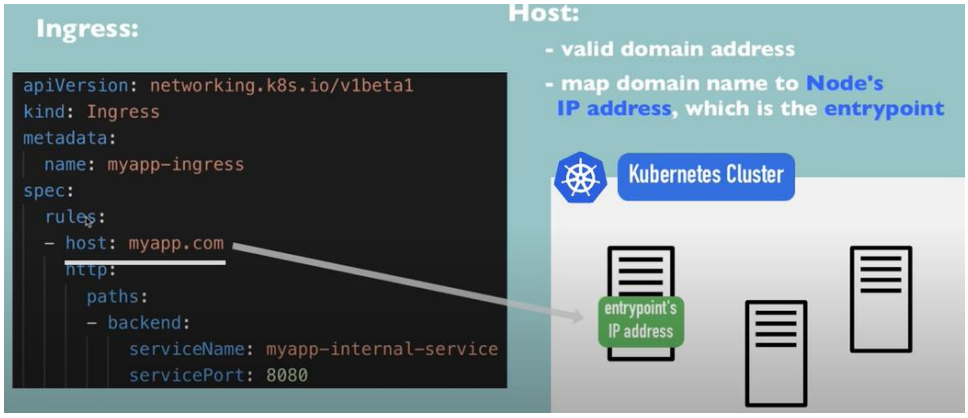
Forward request to the internal service.

my-app
http://my-app.com /..

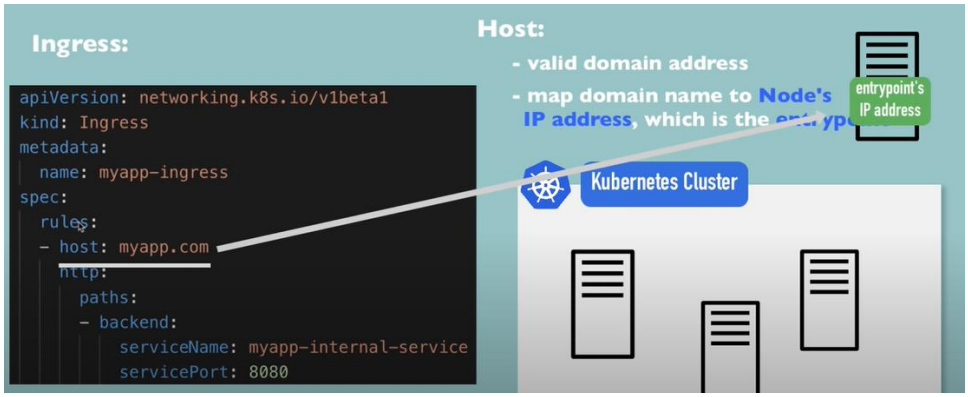




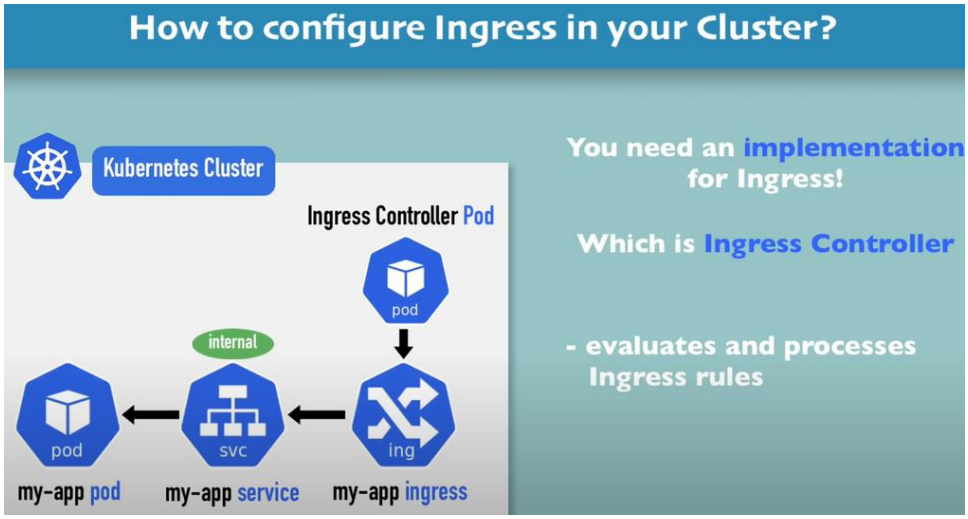


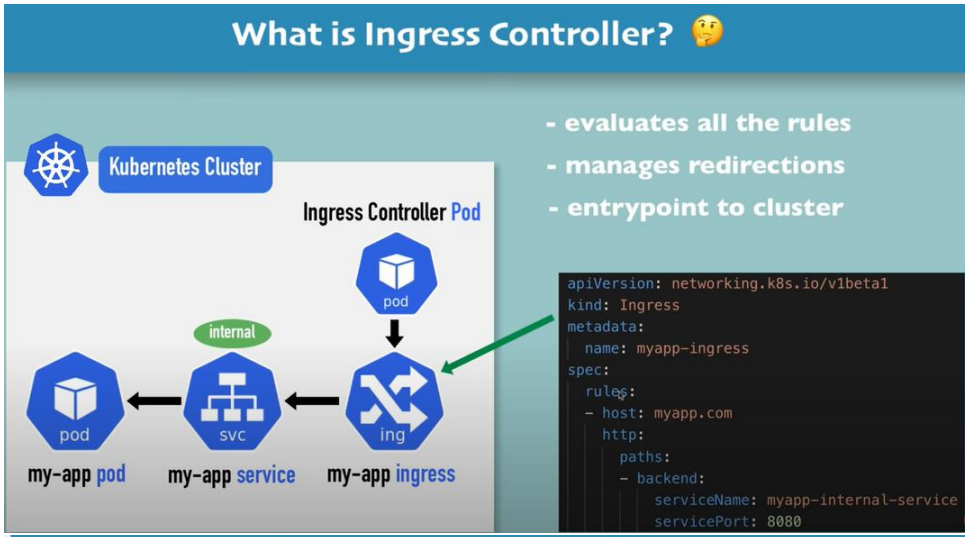


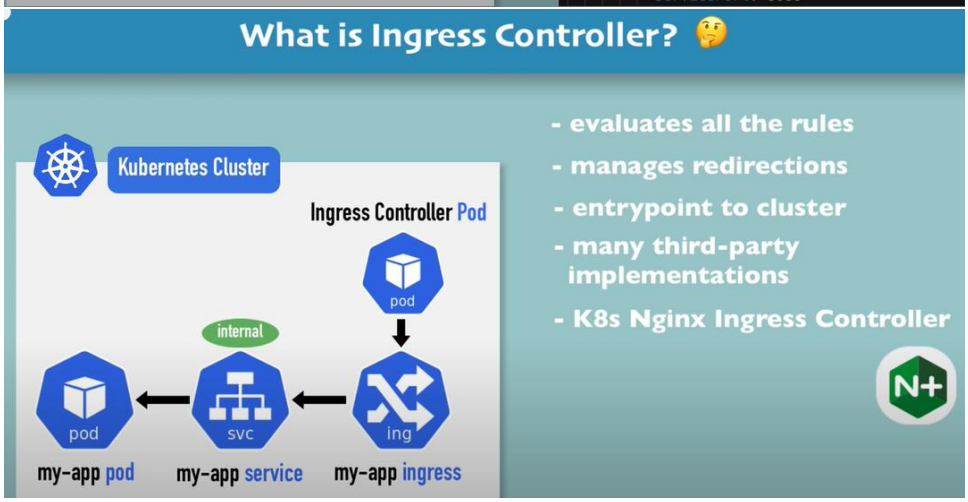
Additional option:



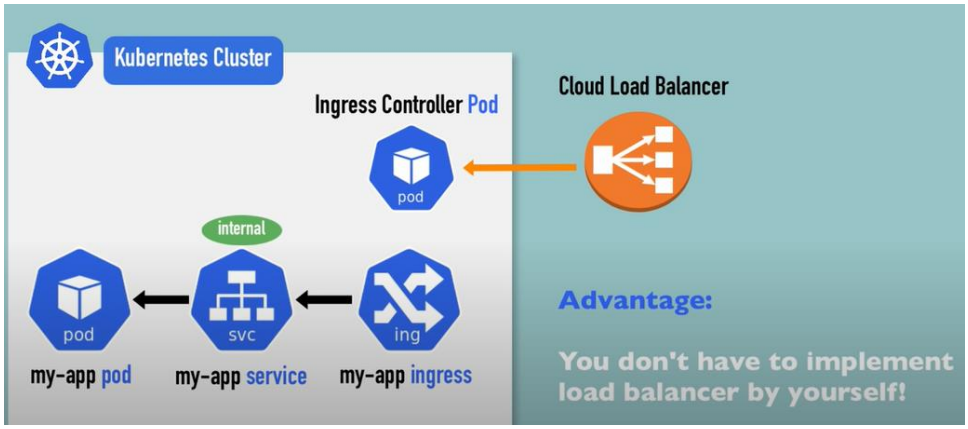
Ingress Controller

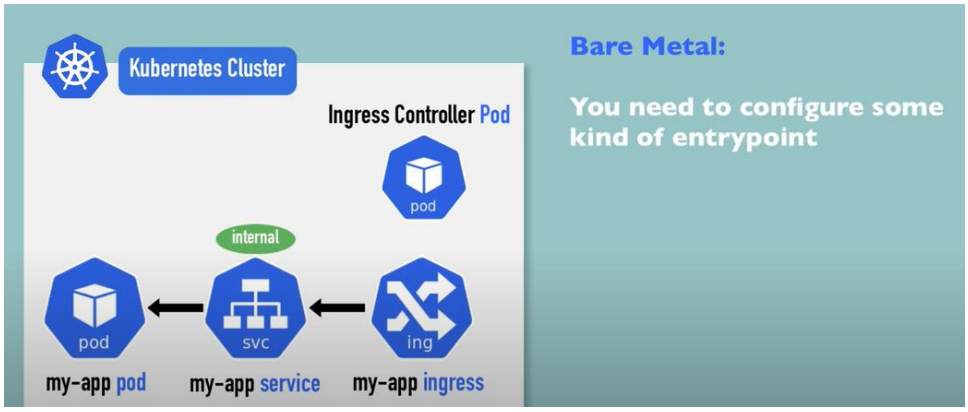


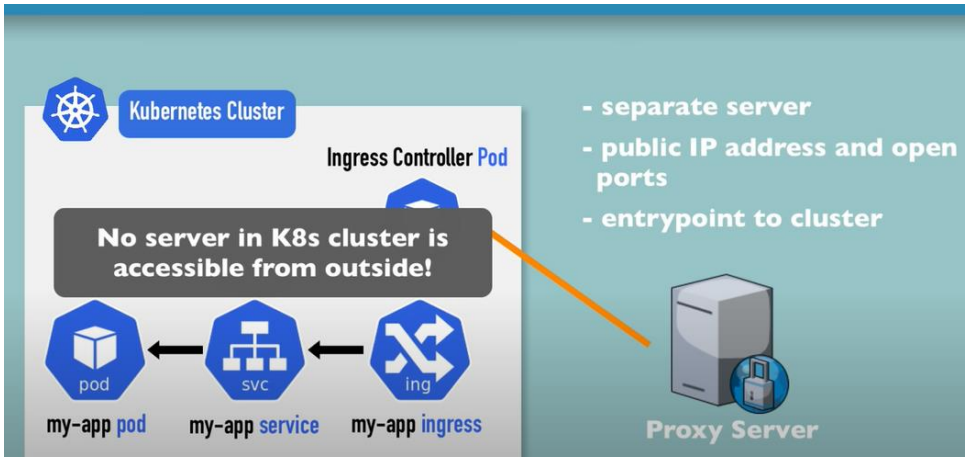

What is Ingress Controller?
- evaluates all the rules
- manages redirections
- entrypoint to cluster
Kubernetes Cluster
Ingress Controller Pod
pod
internal
pod
svc
ing
my-app pod
my-app service
my-app ingress
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: myapp-ingress
spec:
rules:
- host: myapp.com
http:
paths:
- backend:
serviceName: myapp-internal-service
servicePort: 8080


What is Ingress Controller?
- evaluates all the rules
- manages redirections
- entrypoint to cluster
- many third-party implementations
- K8s Nginx Ingress Controller
Kubernetes Cluster
Ingress Controller Pod
pod
internal
pod
svc
ing
my-app pod
my-app service
my-app ingress

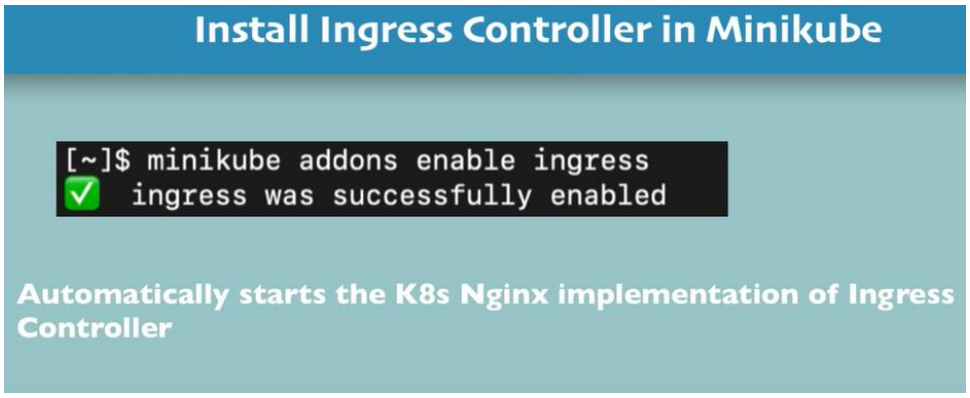
K8s on cloud:

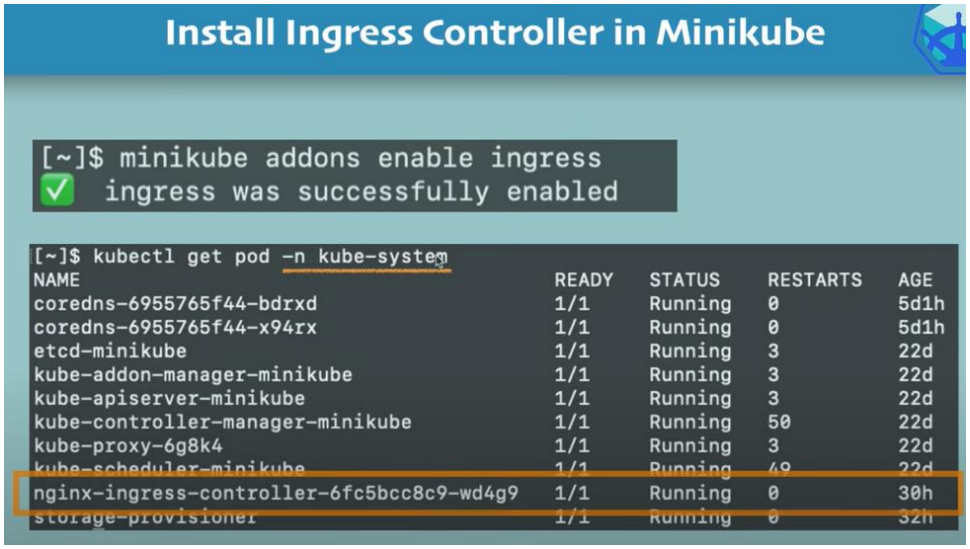

Kubernetes Cluster
Ingress Controller Pod
Cloud Load Balancer
pod
internal
pod
svc
ing
my-app pod
my-app service
my-app ingress
Advantage:
You don't have to implement load balancer by yourself!



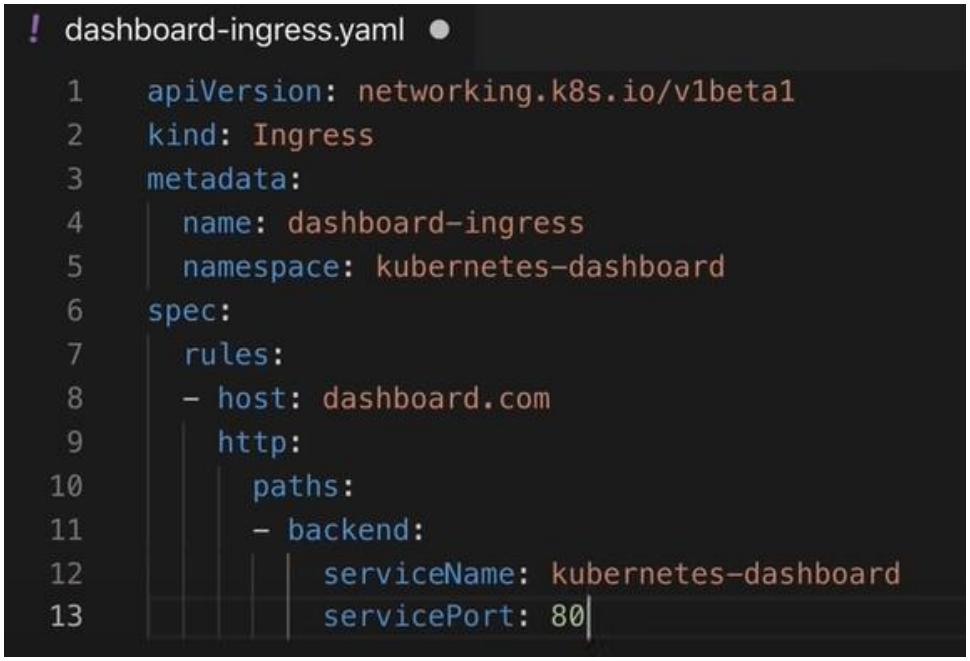


Implementation

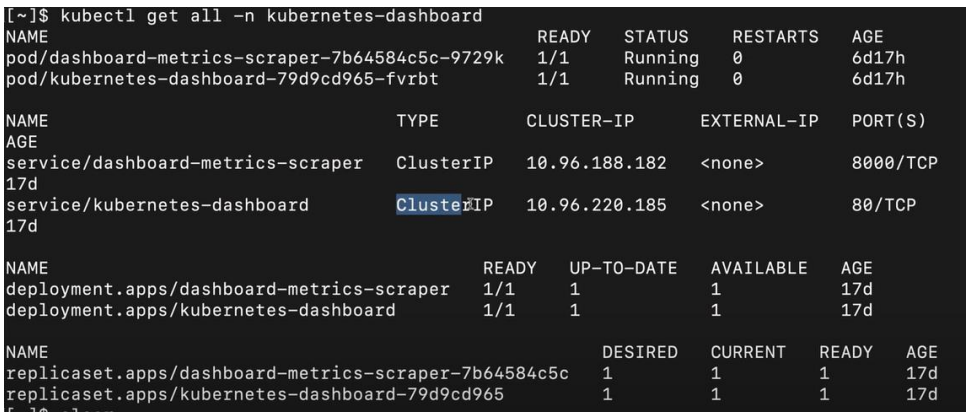




Ingress config file:



It will forward to the internal service (ClusterIP TYPE)



How to resolve ip adress to domain name (dashboard.com) ?

K8s will assign ip address to ingress:

```
[~]$ kubectl apply -f dashboard-ingress.yaml
ingress.networking.k8s.io/dashboard-ingress created
[~]$ kubectl get ingress -n kubernetes-dashboard
NAME                HOSTS           ADDRESS   PORTS   AGE
dashboard-ingress   dashboard.com             80      14s
[~]$ kubectl get ingress -n kubernetes-dashboard --watch
NAME                HOSTS           ADDRESS         PORTS   AGE
dashboard-ingress   dashboard.com   192.168.64.5    80      42s
c^C[~]$
```

In /etc/hosts file on the PC/ laptop add the domain name:

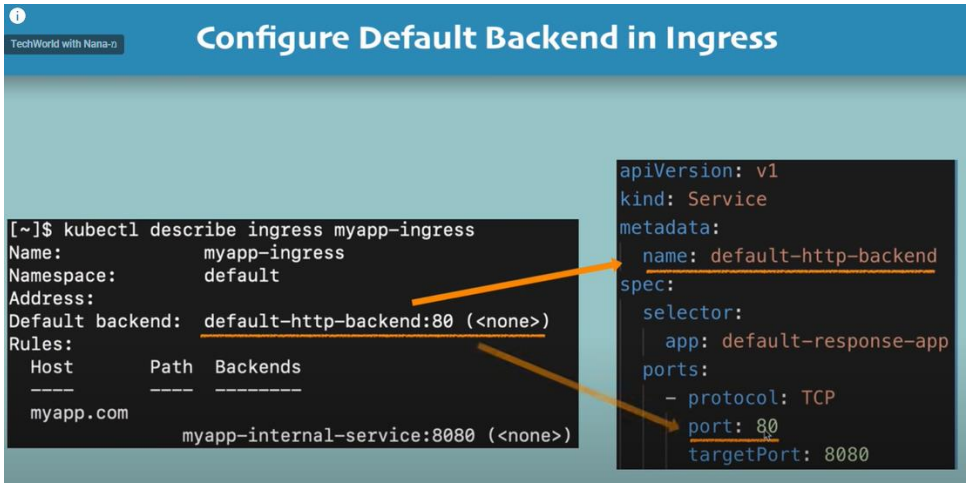
```
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
192.168.64.5    dashboard.com
~
```

Ingress backend

```
[~]$ kubectl describe ingress dashboard-ingress -n kubernetes-dashboard
Name:             dashboard-ingress
Namespace:        kubernetes-dashboard
Address:          192.168.64.5
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host          Path  Backends
  ----          ----  --------
  dashboard.com
                   kubernetes-dashboard:80 (172.17.0.3:9090)
Annotations:
  kubectl.kubernetes.io/last-applied-configuration:  {"apiVersion":"networking.k8s.io/v1beta1","kind":"Ingress","metadata":{"annotations":{},"name":"dashboard-ingress","namespace":"kubernetes-dashboard"},"spec":{"rules":[{"host":"dashboard.com","http":{"paths":[{"backend":{"serviceName":"kubernetes-dashboard","servicePort":80}}]}}]}}

Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  CREATE  12m   nginx-ingress-controller  Ingress kubernetes-dashboard/dashboard-ingress
  Normal  UPDATE  12m   nginx-ingress-controller  Ingress kubernetes-dashboard/dashboard-ingress
```

```
Default backend:  default-http-backend:80 (<none>)
```

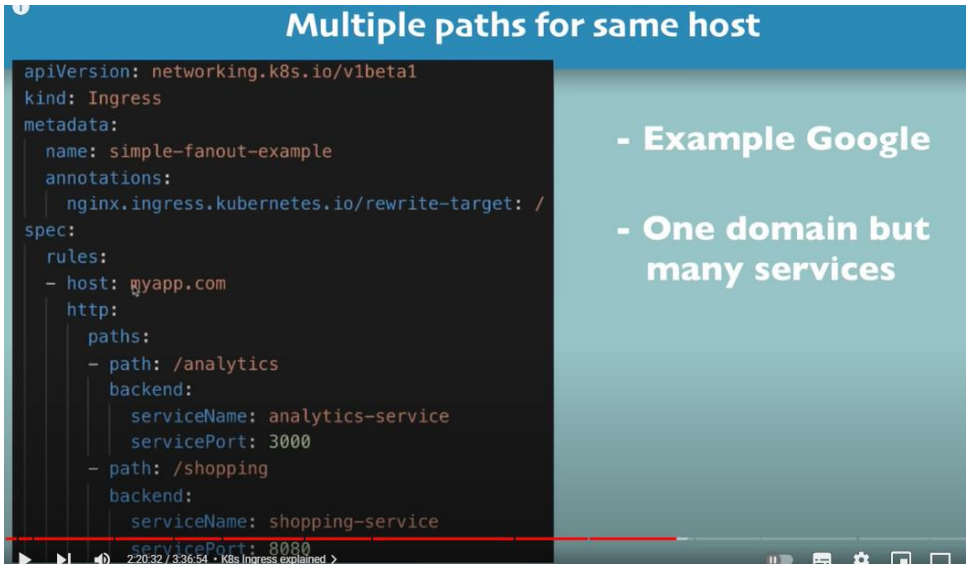
If u call a service that is not mapped to any backend – this will be the default

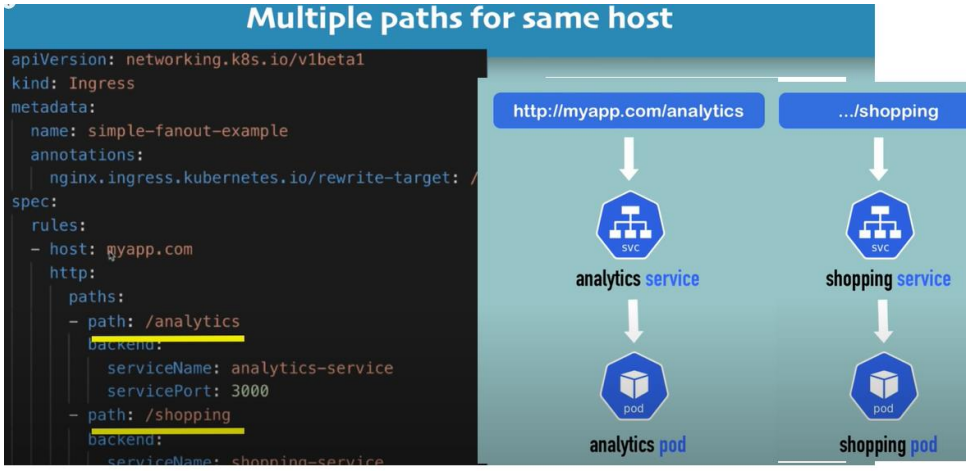


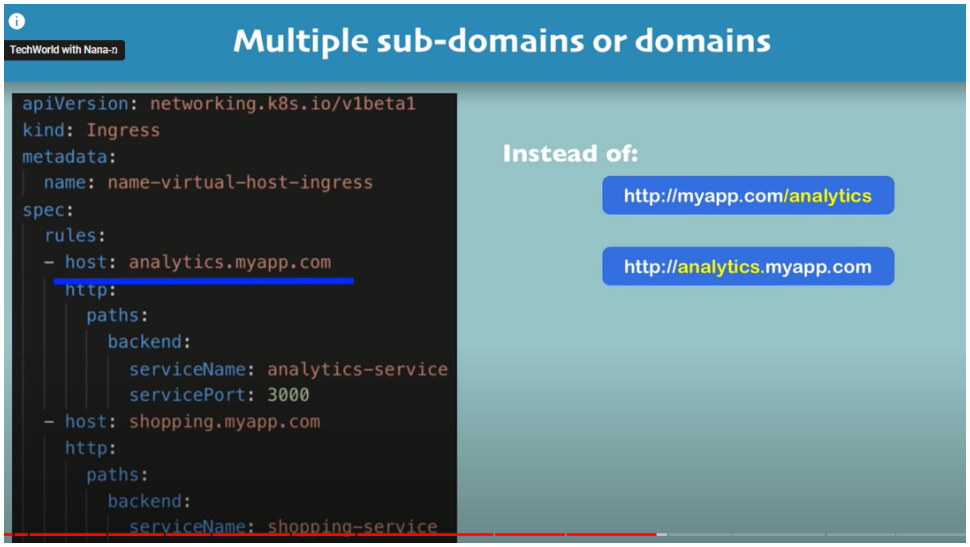
+ to create a pod to return meaningful message

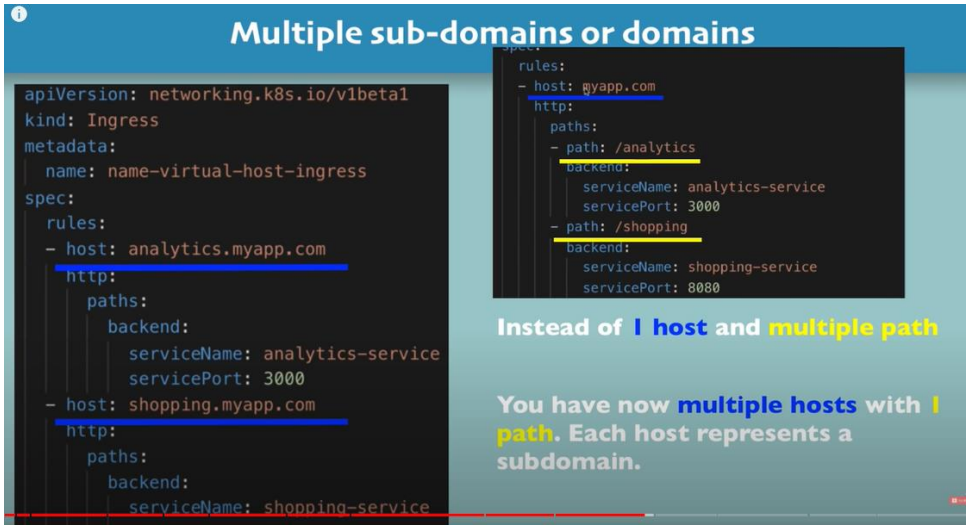
Additional use cases for ingress

Many services on same domain





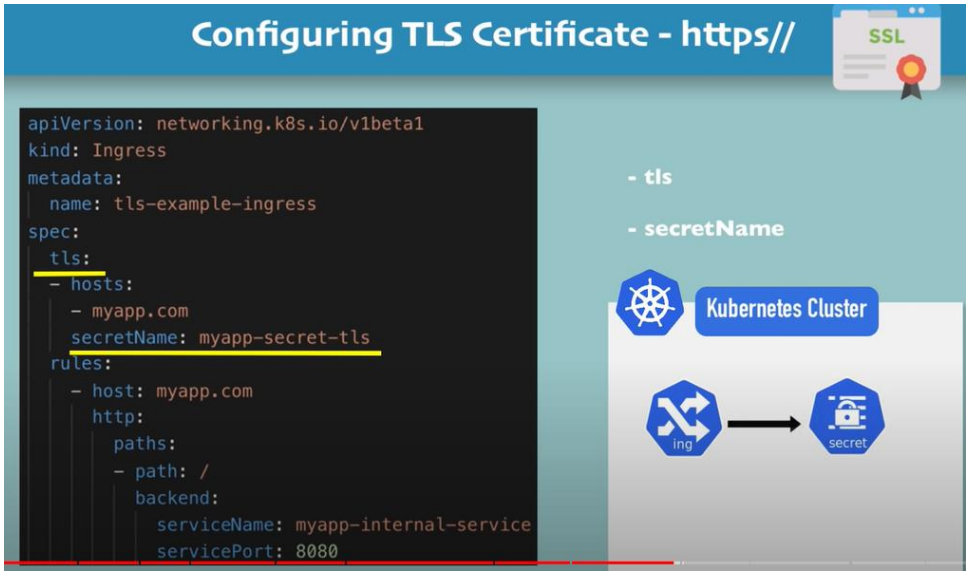


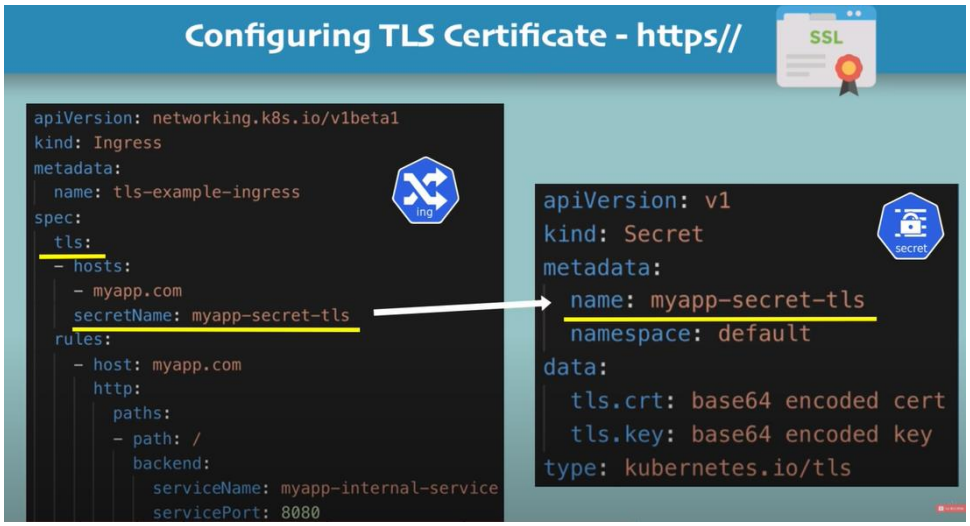


SSL configuration

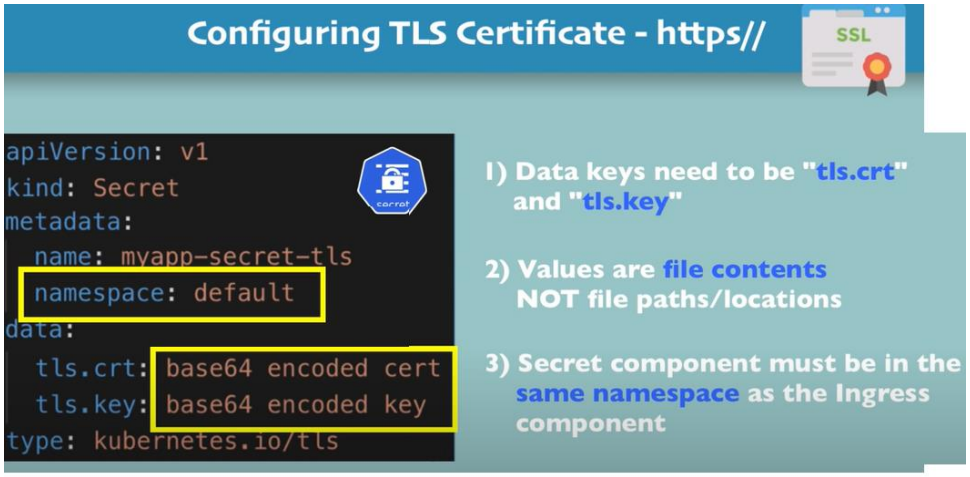
Installation to get crt key/value:

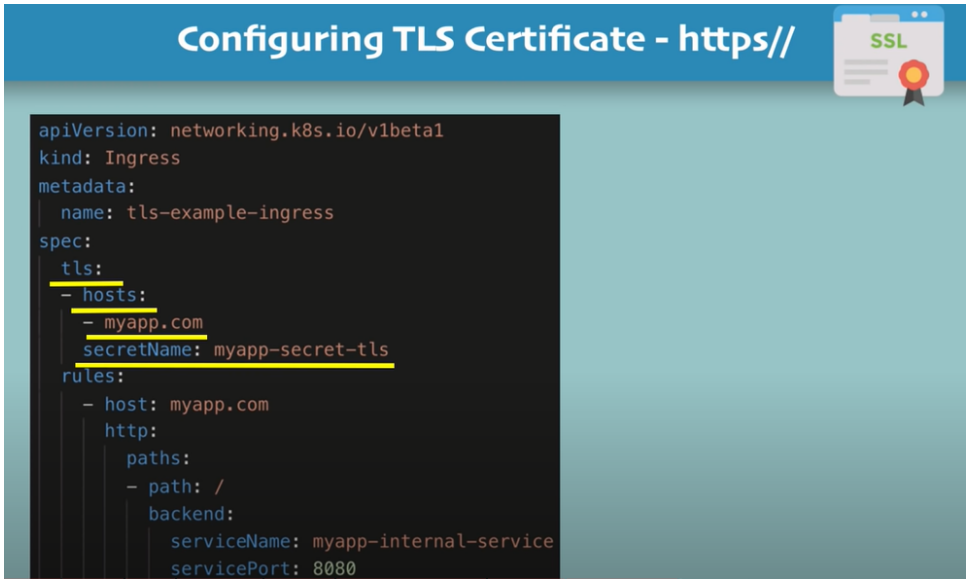
https://www.stechies.com/installing-openssl-windows-10-11/



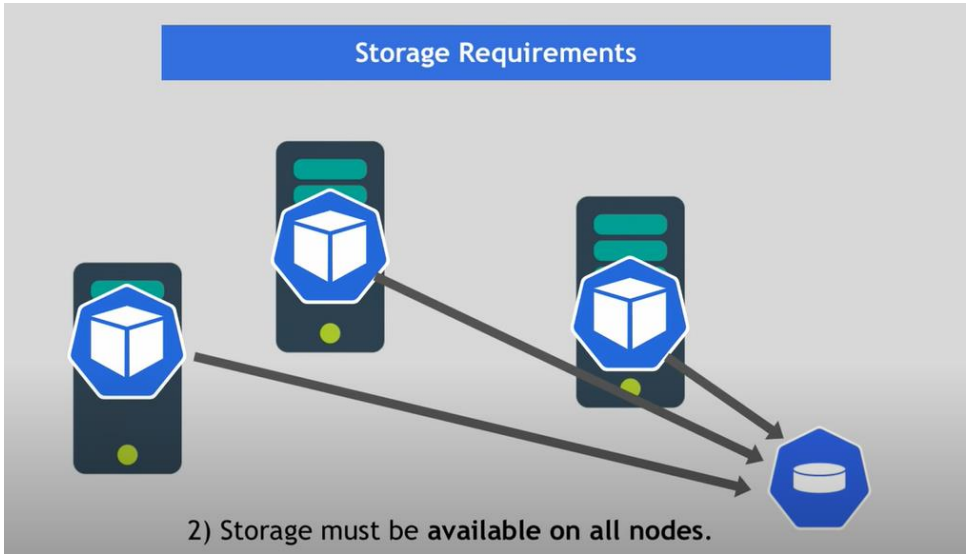


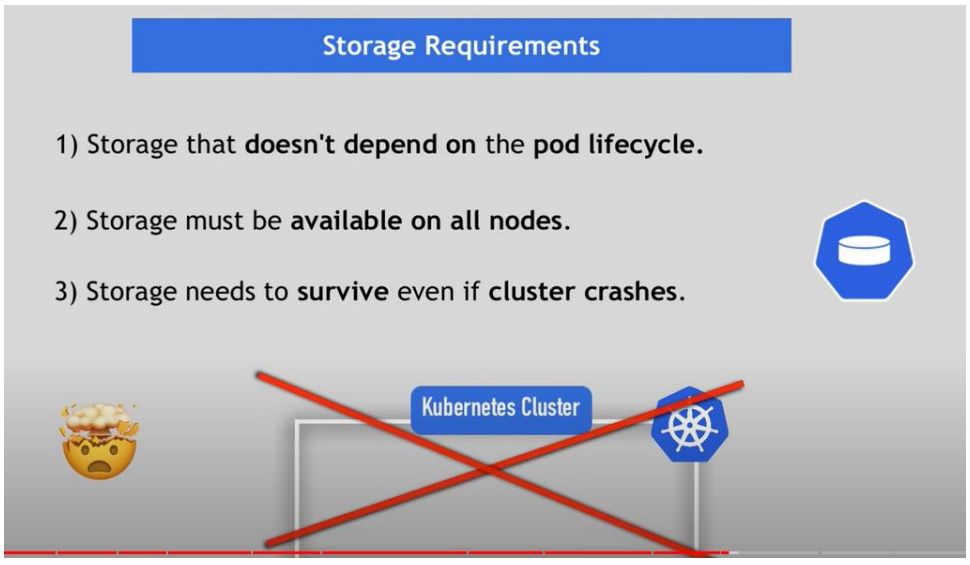
Tls config file:

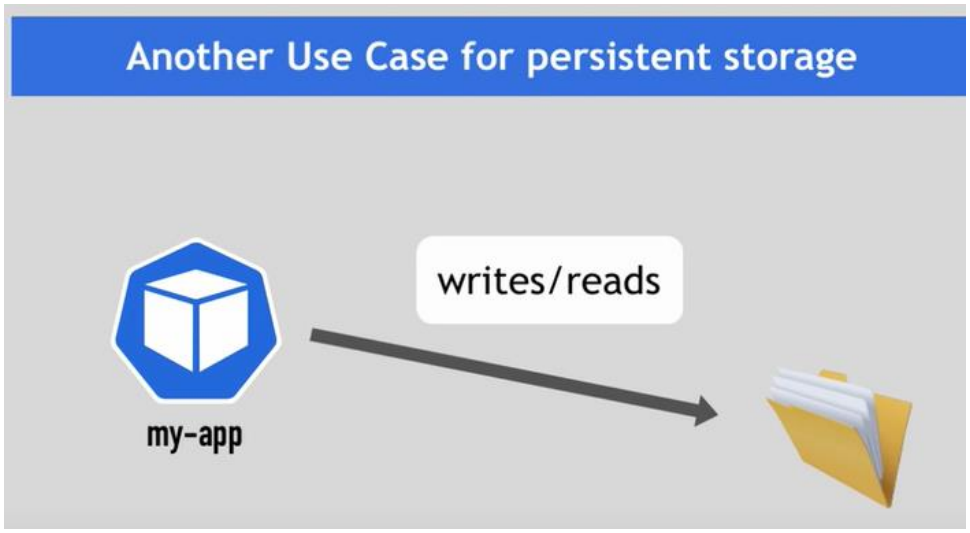


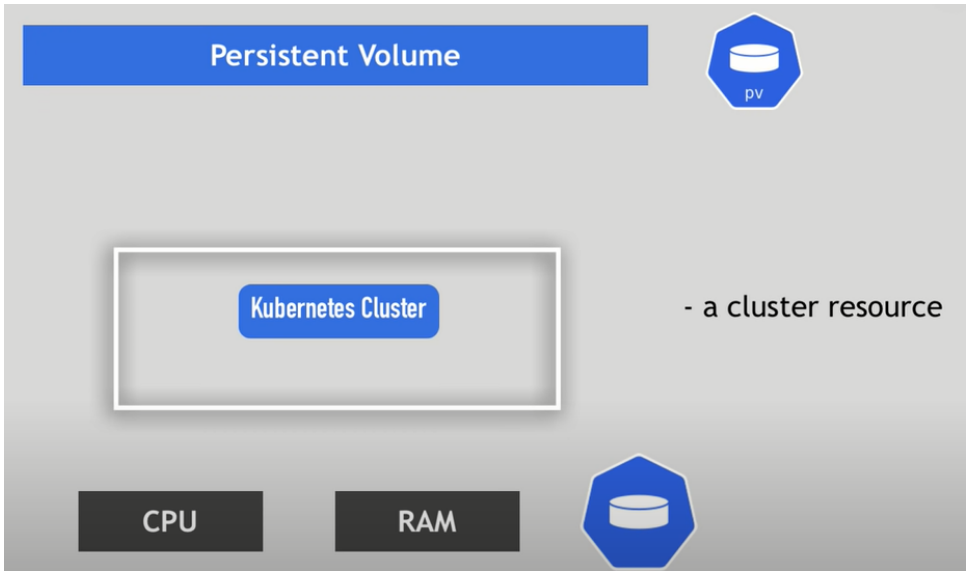


Persistant Data using VOLUMES

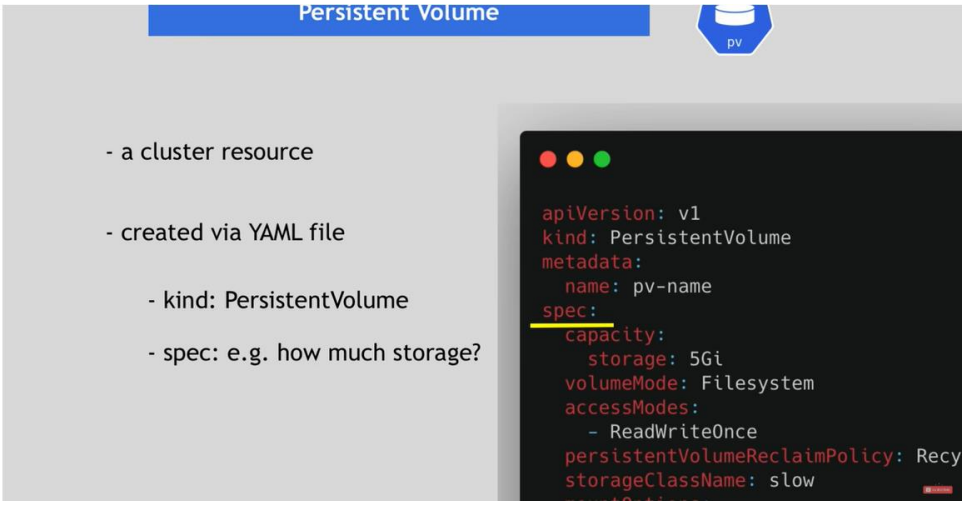


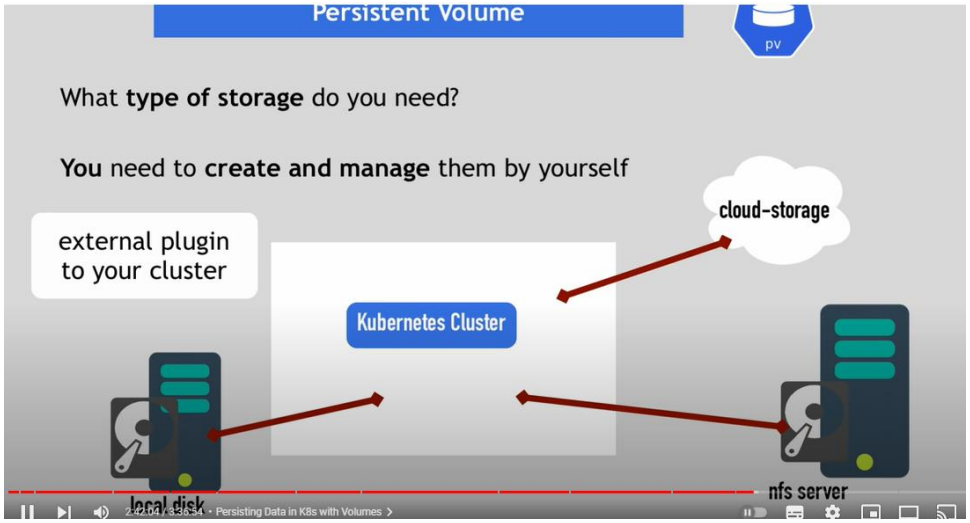


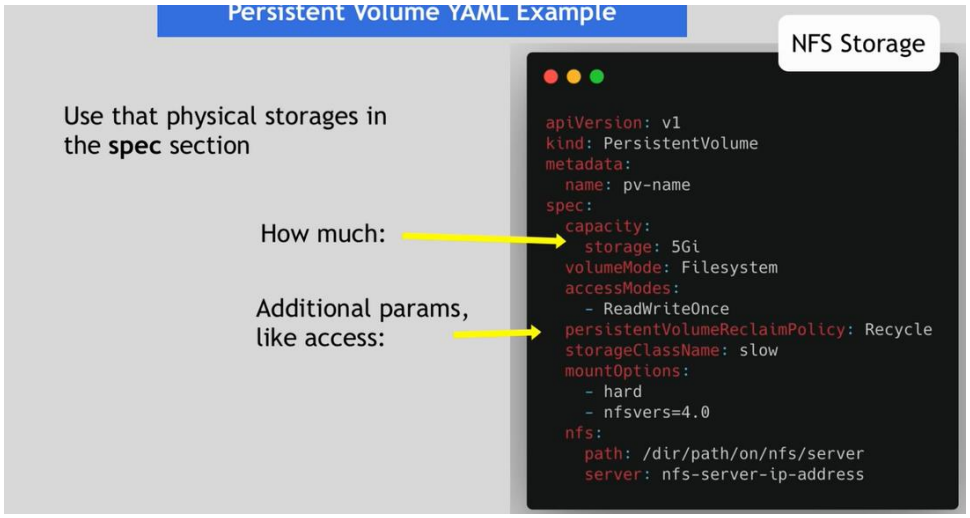

Another Use Case for persistent storage
writes/reads
my-app

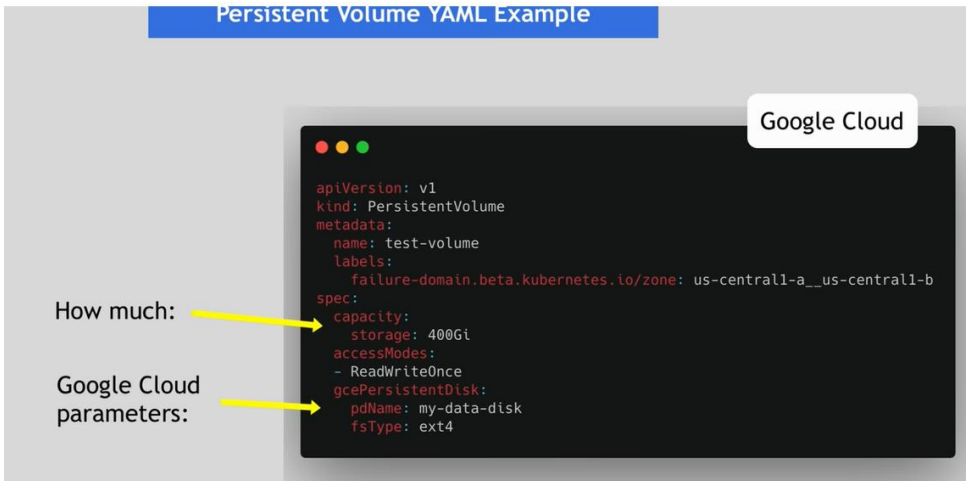

Persistent Volume
pv
Kubernetes Cluster
- a cluster resource
CPU
RAM

Yaml file:


Persistent Volume
pv
- a cluster resource
- created via YAML file
- kind: PersistentVolume
- spec: e.g. how much storage?
apiVersion: v1
kind: PersistentVolume
metadata:
name: pv-name
spec:
capacity:
storage: 5Gi
volumeMode: Filesystem
accessModes:
- ReadWriteOnce
persistentVolumeReclaimPolicy: Recy
storageClassName: slow


Persistent Volume
pv
What type of storage do you need?
You need to create and manage them by yourself
external plugin
to your cluster
cloud-storage
Kubernetes Cluster
local disk
nfs server
Persisting Data in K8s with Volumes


Persistent Volume YAML Example
NFS Storage
Use that physical storages in
the spec section
How much:
Additional params,
like access:
apiVersion: v1
kind: PersistentVolume
metadata:
name: pv-name
spec:
capacity:
storage: 5Gi
volumeMode: Filesystem
accessModes:
- ReadWriteOnce
persistentVolumeReclaimPolicy: Recycle
storageClassName: slow
mountOptions:
- hard
- nfsvers=4.0
nfs:
path: /dir/path/on/nfs/server
server: nfs-server-ip-address


Persistent Volume YAML Example
Google Cloud
How much:
Google Cloud
parameters:
apiVersion: v1
kind: PersistentVolume
metadata:
name: test-volume
labels:
failure-domain.beta.kubernetes.io/zone: us-central1-a__us-central1-b
spec:
capacity:
storage: 400Gi
accessModes:
- ReadWriteOnce
gcePersistentDisk:
pdName: my-data-disk
fsType: ext4

Persistent Volume YAML Example

Depending on storage type,
spec attributes differ

Node Affinity:

Local storage

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

Local vs. Remote Volume Types

Each volume type has it's own use case!

Local volume types violate 2. and 3. requirement for data persistence:

❌ Being tied to 1 specific node

❌ surviving cluster crashes

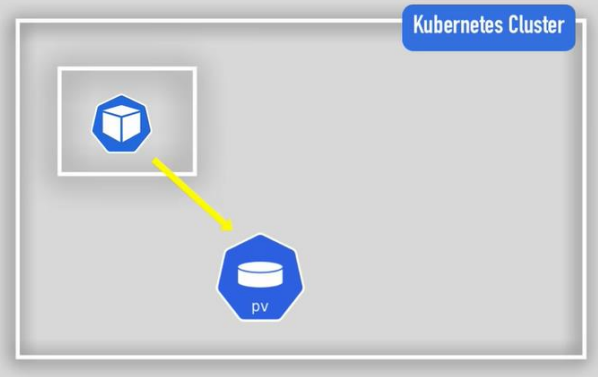
For DB persistence use remote storage!

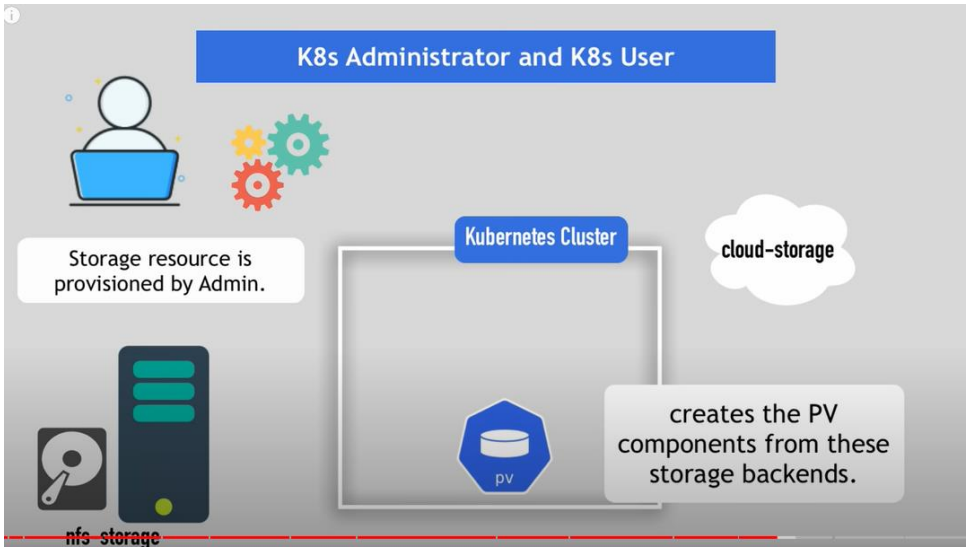
K8s Administrator and K8s User

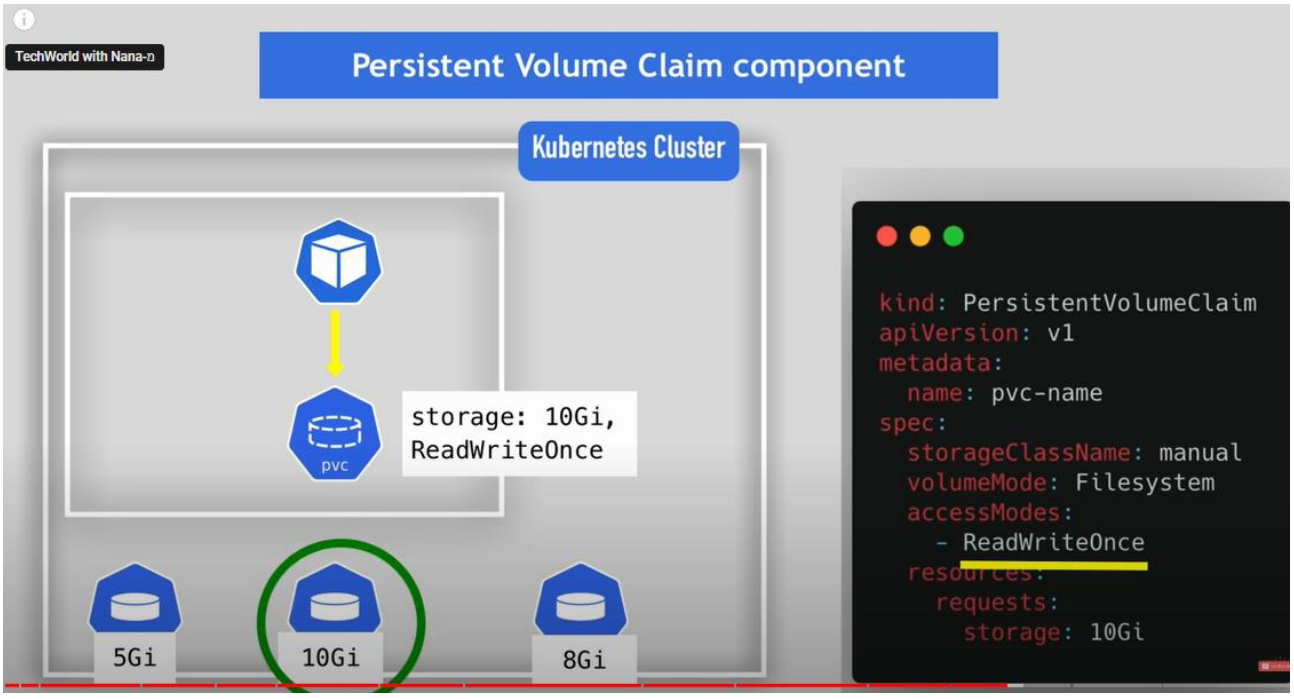
Who creates the Persistent Volumes and when? 🤔

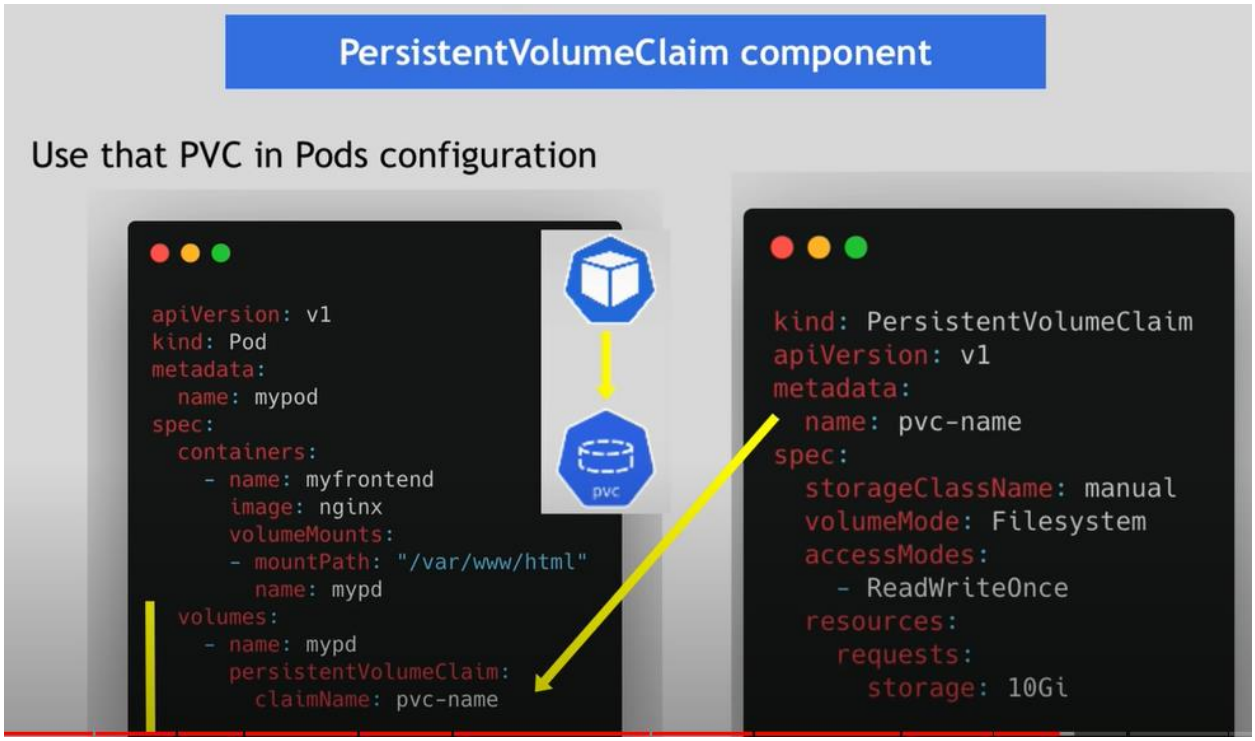
..the Pod that **depends on** it is created

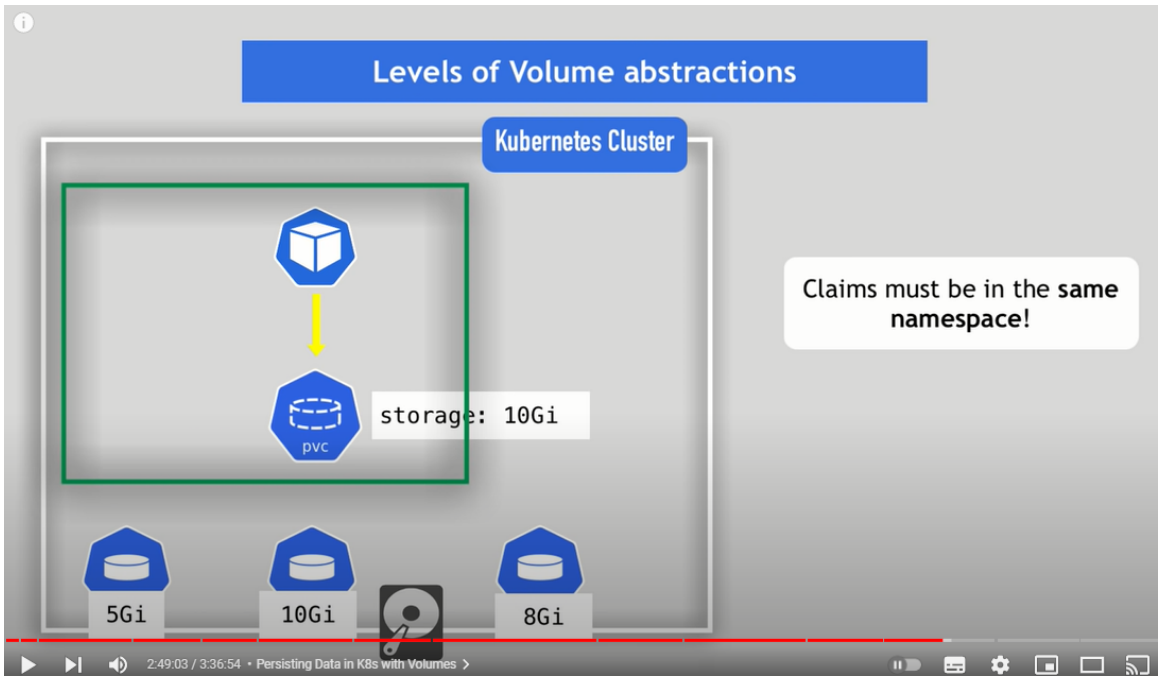
PV are resources that need to be there **BEFORE**..

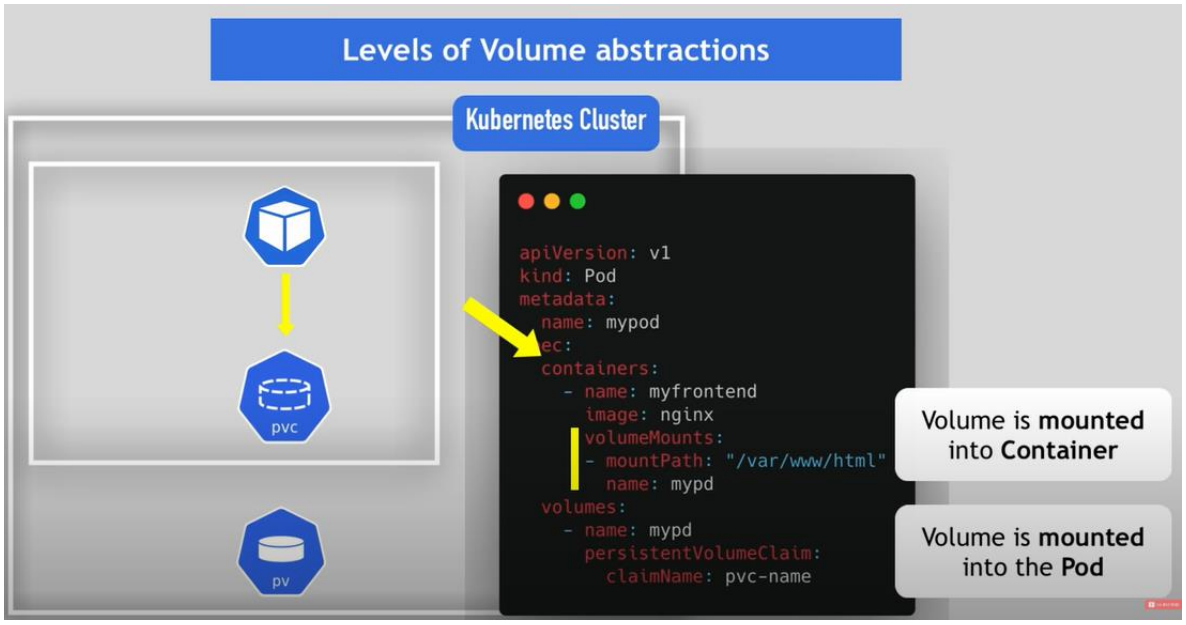



K8s Administrator and K8s User
Storage resource is provisioned by Admin.
Kubernetes Cluster
cloud-storage
pv
creates the PV components from these storage backends.
nfs storage

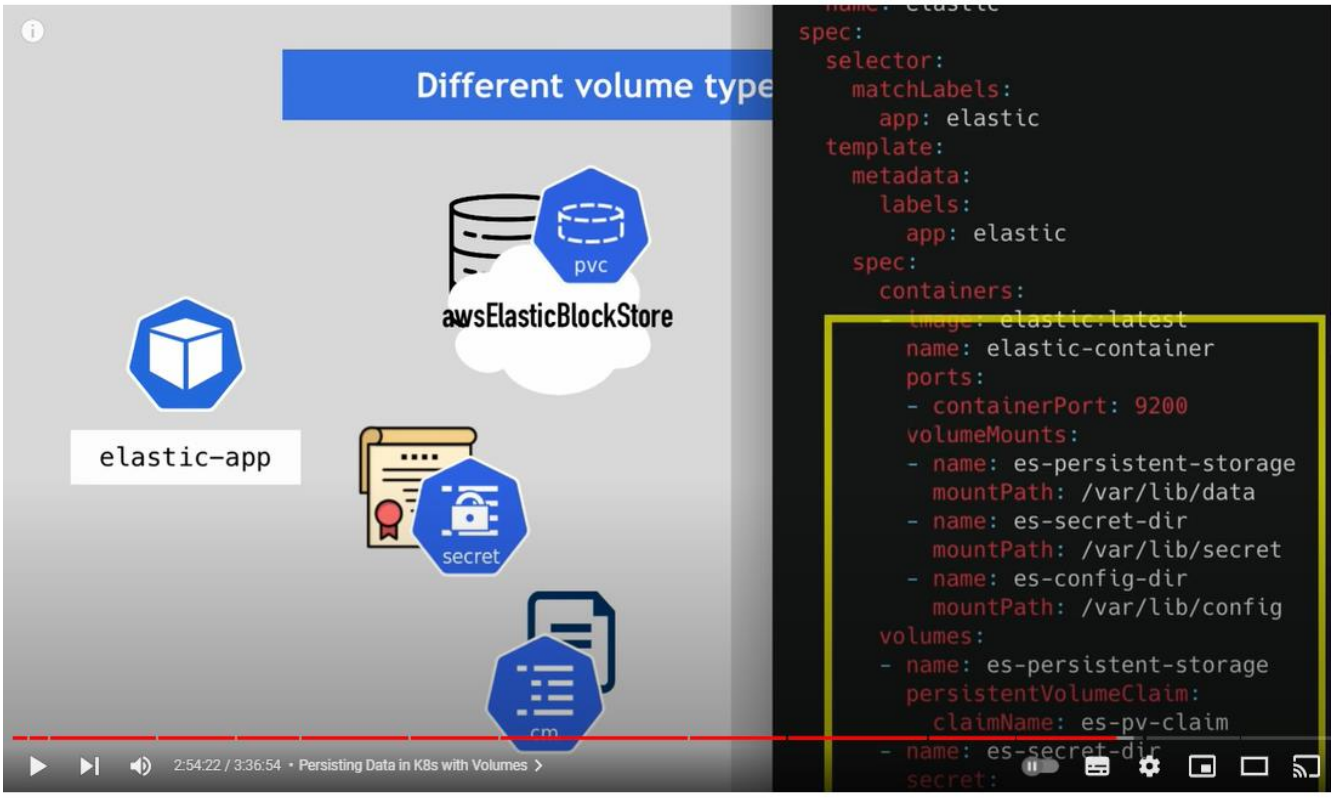

TechWorld with Nana
Persistent Volume Claim component
Kubernetes Cluster
pvc
storage: 10Gi,
ReadWriteOnce
5Gi
10Gi
8Gi
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-name
spec:
storageClassName: manual
volumeMode: Filesystem
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi

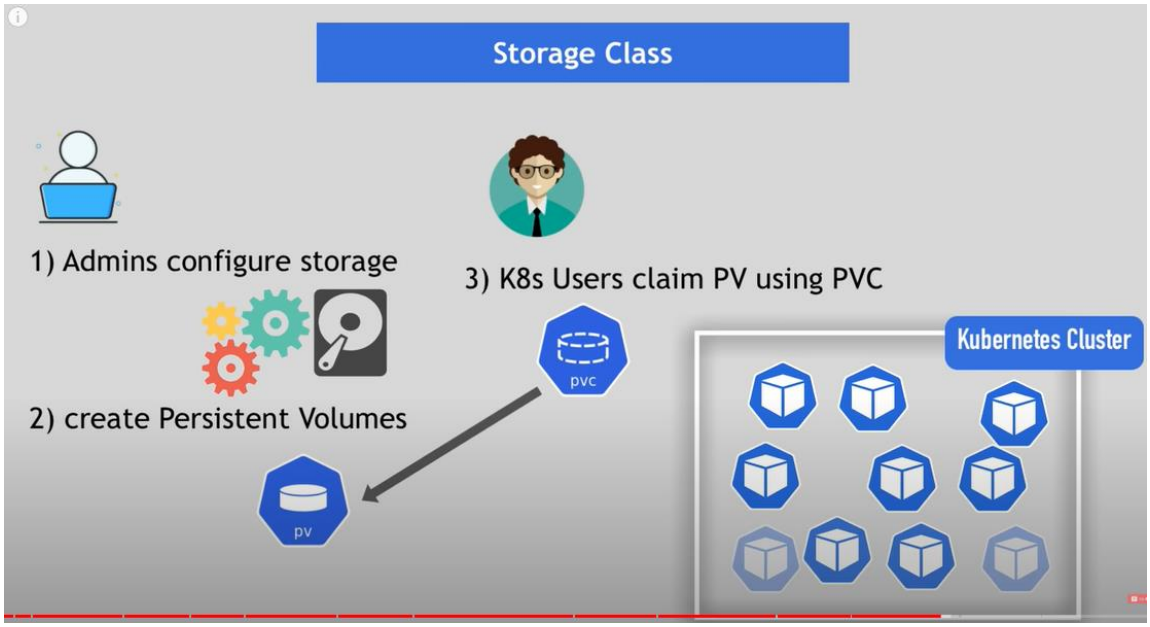

PersistentVolumeClaim component
Use that PVC in Pods configuration
apiVersion: v1
kind: Pod
metadata:
name: mypod
spec:
containers:
- name: myfrontend
image: nginx
volumeMounts:
- mountPath: "/var/www/html"
name: mypd
volumes:
- name: mypd
persistentVolumeClaim:
claimName: pvc-name
pvc
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-name
spec:
storageClassName: manual
volumeMode: Filesystem
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi


Levels of Volume abstractions
Kubernetes Cluster
pvc
storage: 10Gi
Claims must be in the same namespace!
5Gi
10Gi
8Gi
2:49:03 / 3:36:54 • Persisting Data in K8s with Volumes

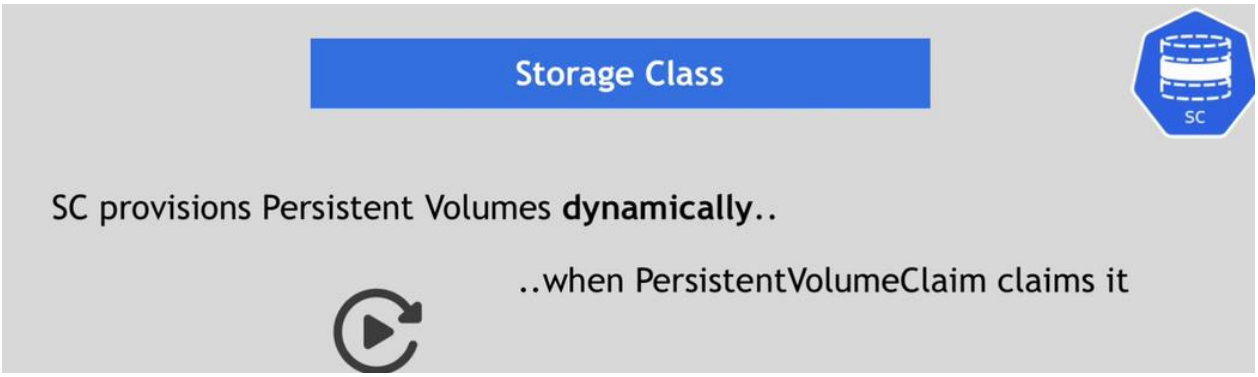


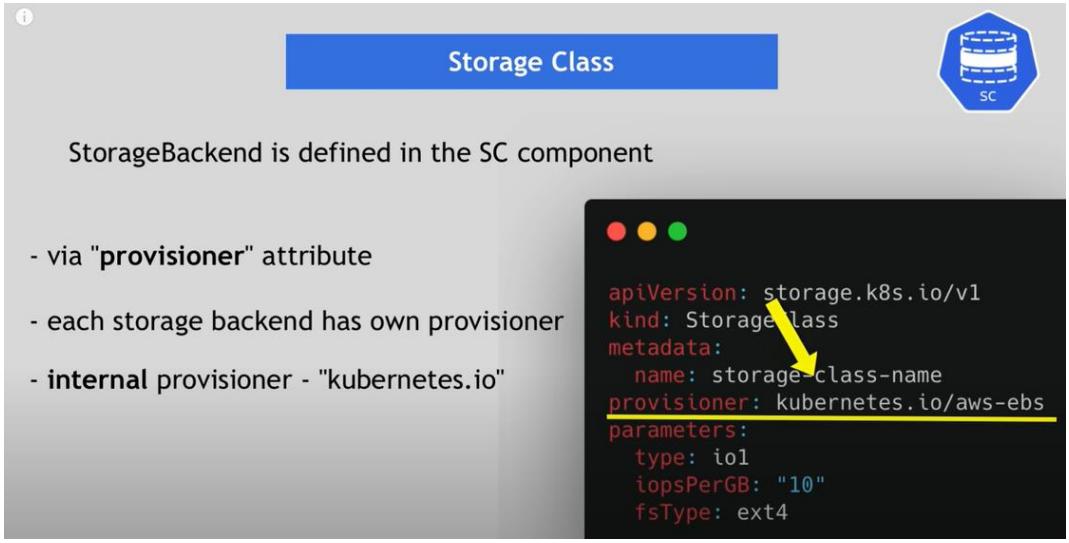
DIFFERENT VOLUME TYPES





Storage class – 3rd party to provision PV (Persistant Volume) when PVC (Persistant Volume claim) claims it





also, there might be external provisioner !!

Storage Class

StorageBackend is defined in the SC component

- via "**provisioner**" attribute
- each storage backend has own provisioner
- **internal** provisioner - "kubernetes.io"
- **external** provisioner
- configure **parameters** for storage we want to request for PV

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class

Another abstraction level

- abstracts underlying storage provider
- parameters for that storage

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class usage

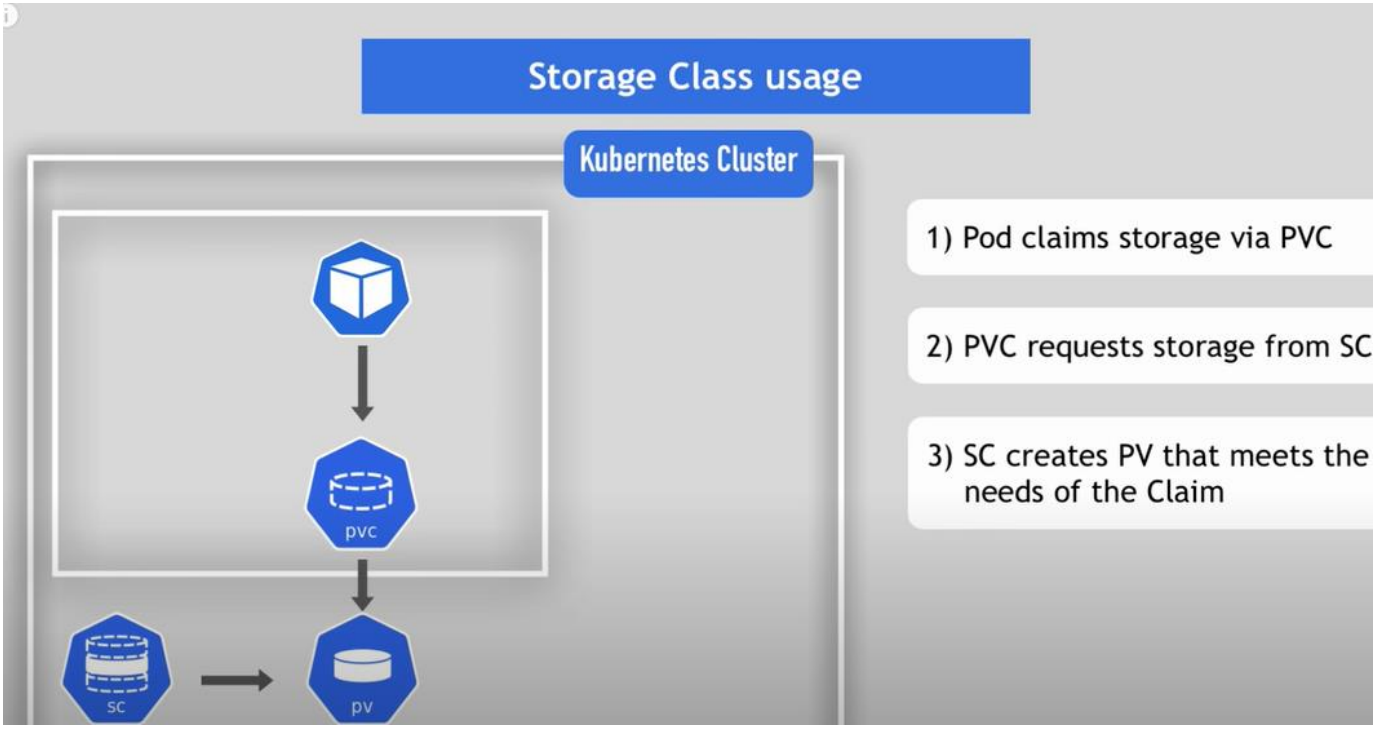
Requested by PersistentVolumeClaim

Storage Class Config

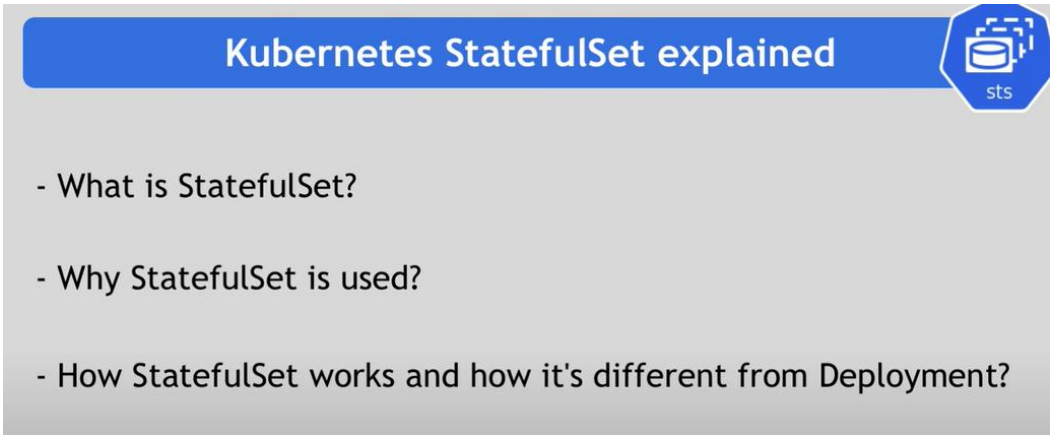
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

PVC Config

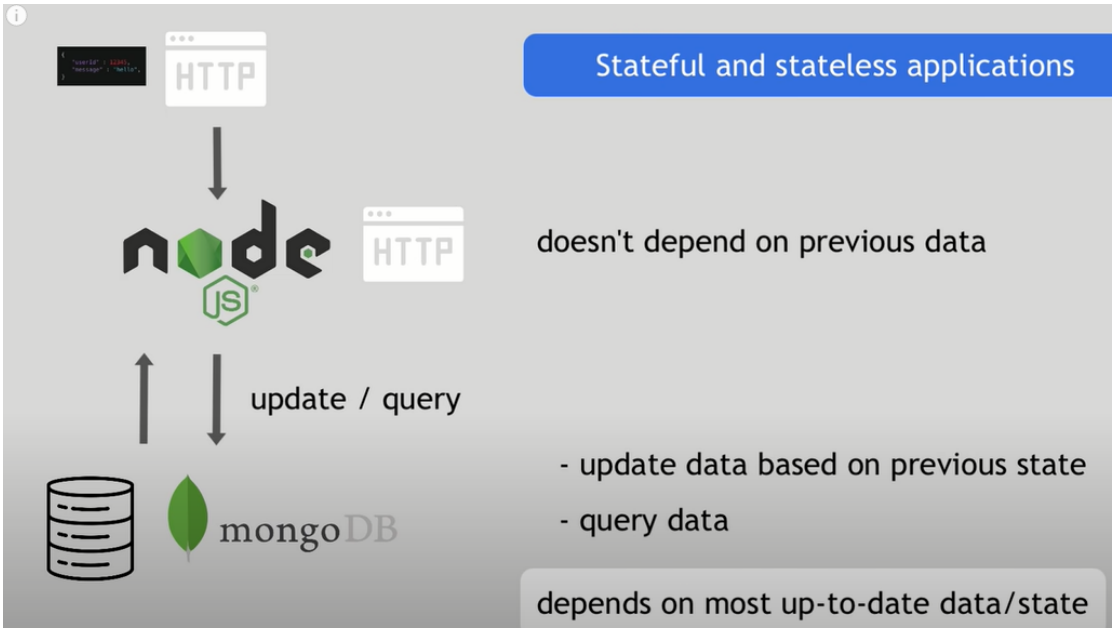
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: mypvc
spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 100Gi
    storageClassName: storage-class-name
```

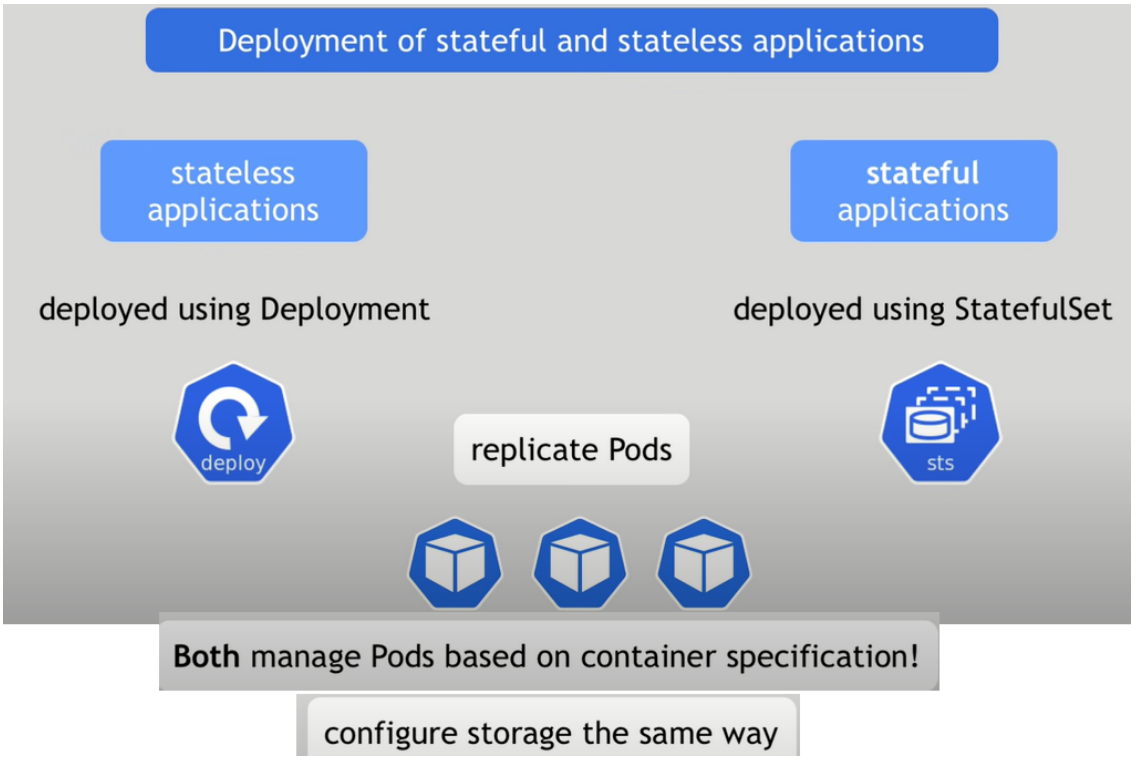


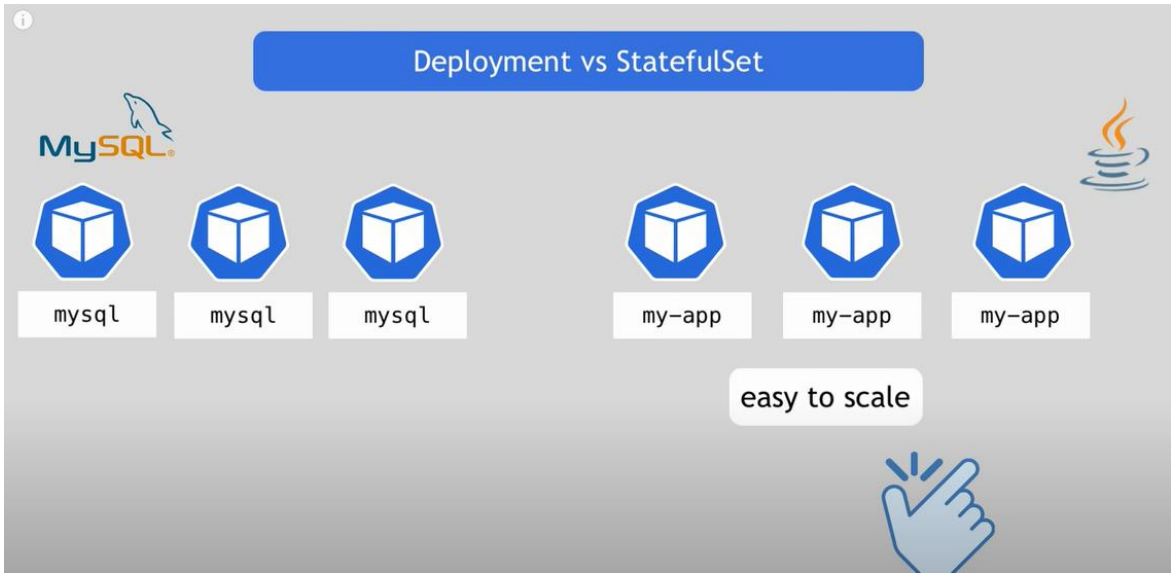
StatefulSet

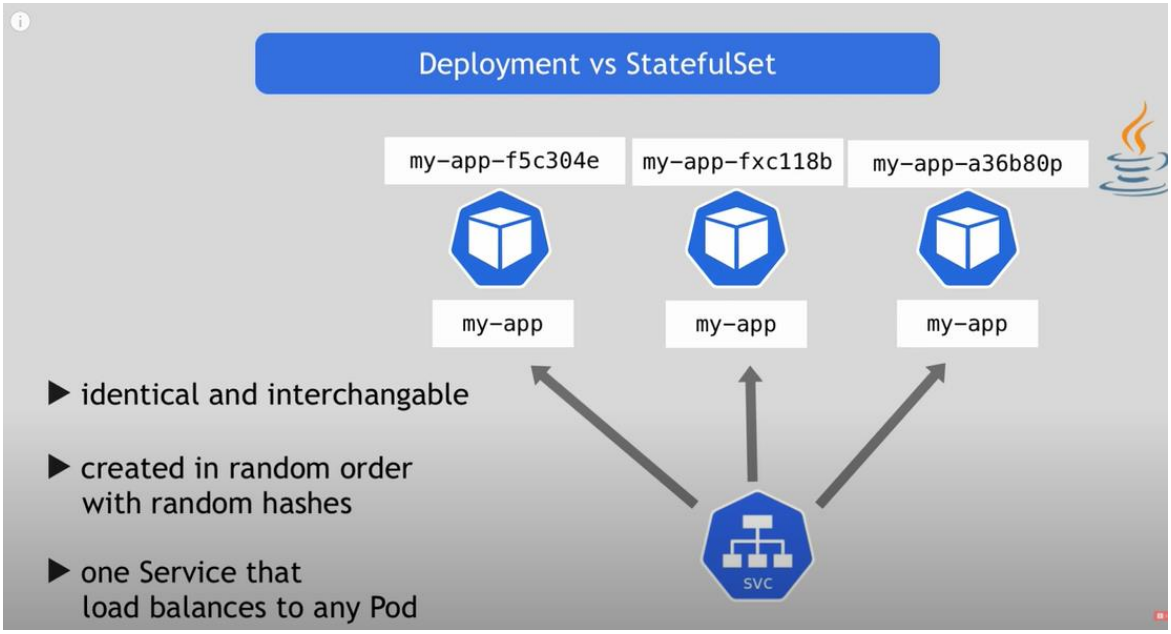


Usage:


Stateful and stateless applications
HTTP
HTTP
doesn't depend on previous data
update / query
mongoDB
- update data based on previous state
- query data
depends on most up-to-date data/state


Deployment of stateful and stateless applications
stateless applications
stateful applications
deployed using Deployment
deployed using StatefulSet
deploy
replicate Pods
sts
Both manage Pods based on container specification!
configure storage the same way


Deployment vs StatefulSet
MySQL
mysql
mysql
mysql
my-app
my-app
my-app
easy to scale


Deployment vs StatefulSet
my-app-f5c304e
my-app-fxc118b
my-app-a36b80p
my-app
my-app
my-app
identical and interchangable
created in random order with random hashes
one Service that load balances to any Pod
SVC

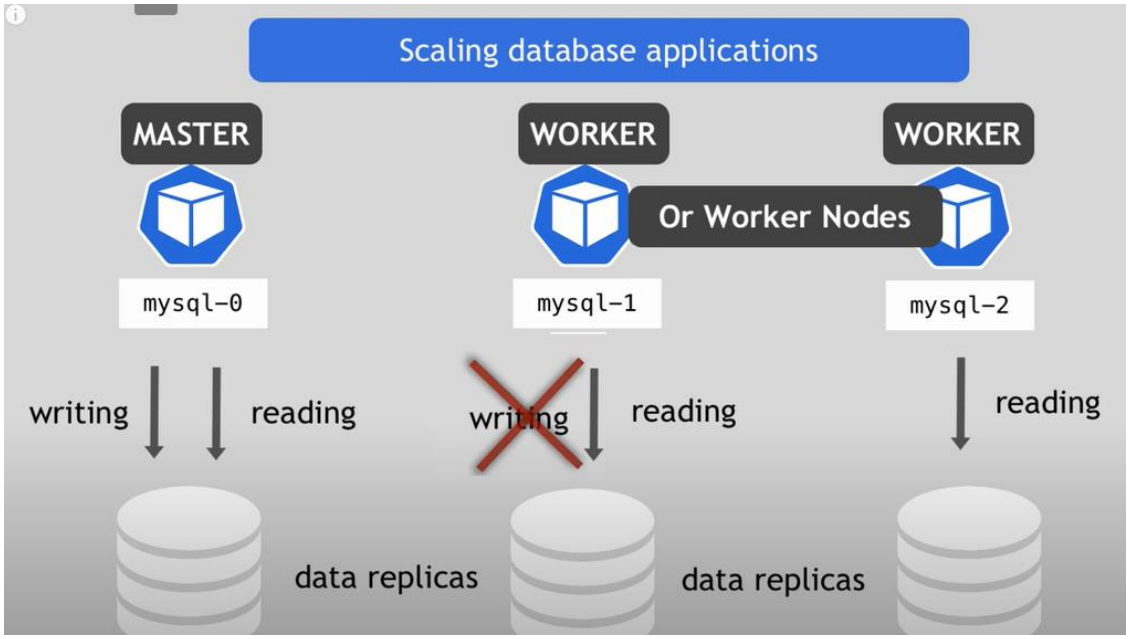
Deployment vs StatefulSet

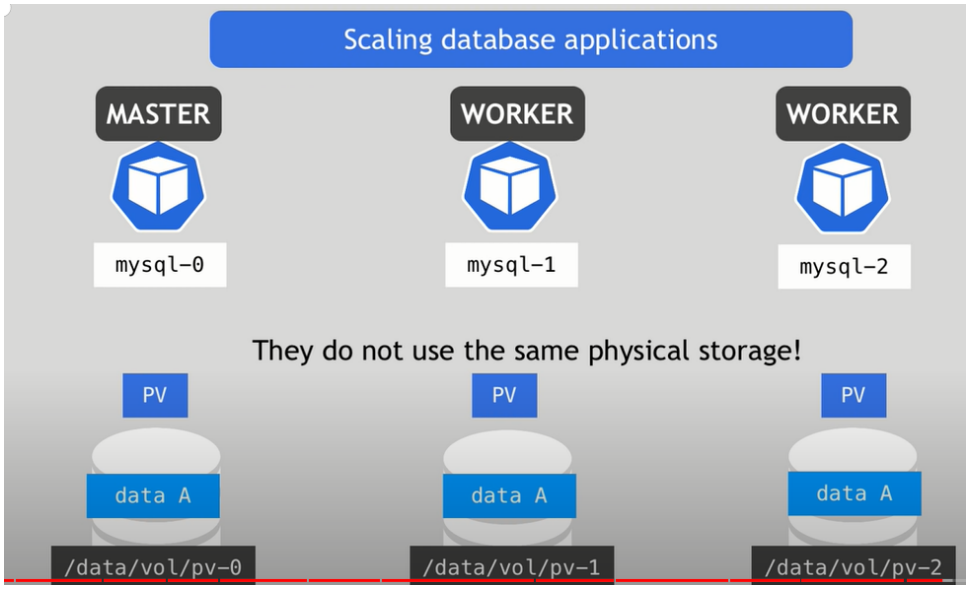
Pod Identity

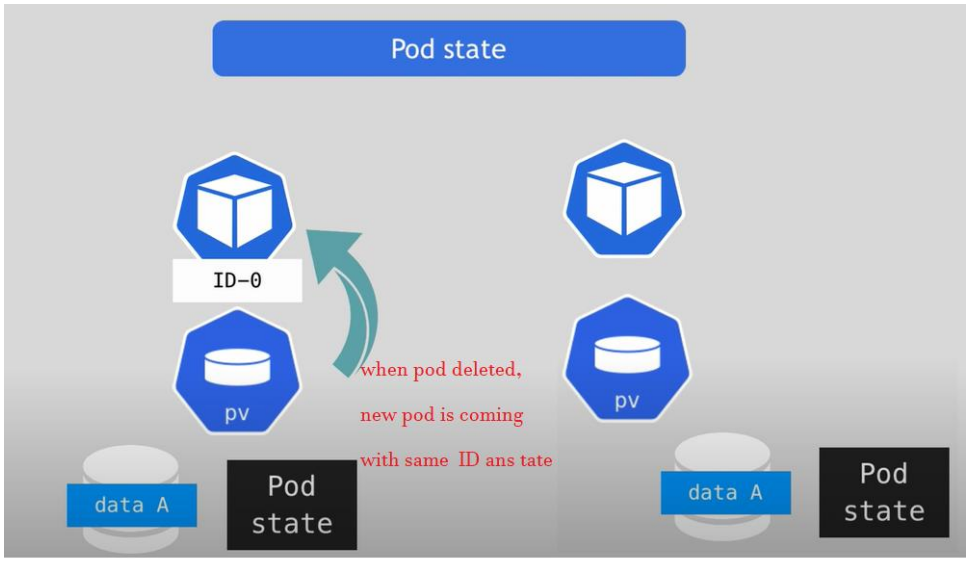
- sticky identity for each pod

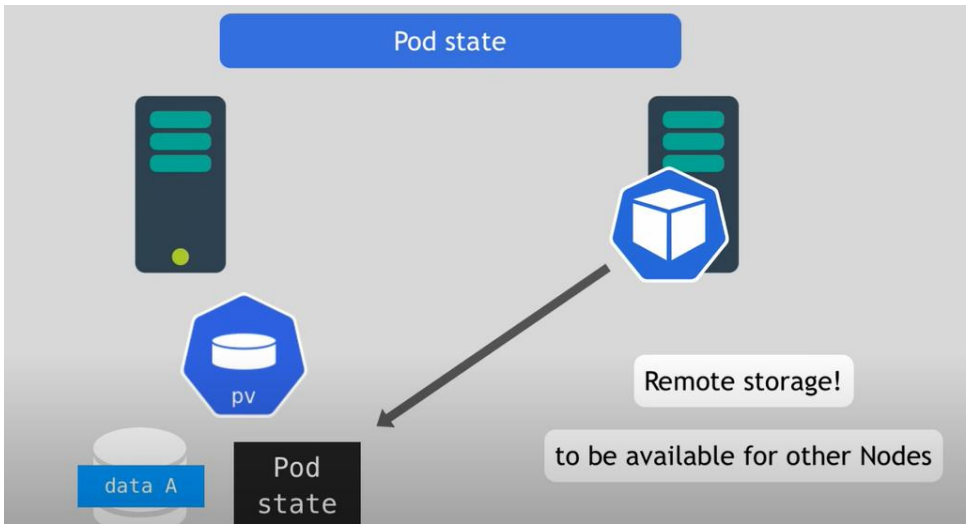
- created from **same specification**, but **not interchangeable**!

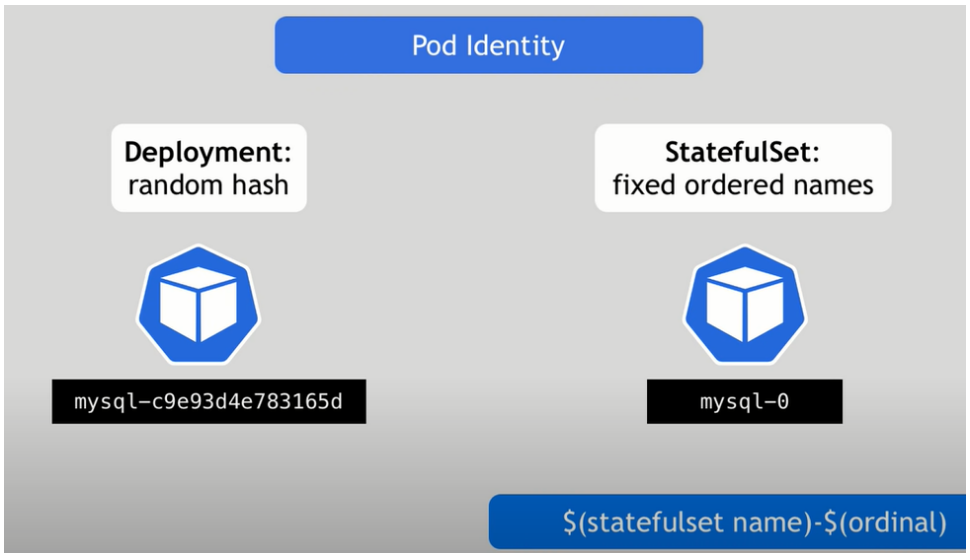
- persistent identifier across any re-scheduling

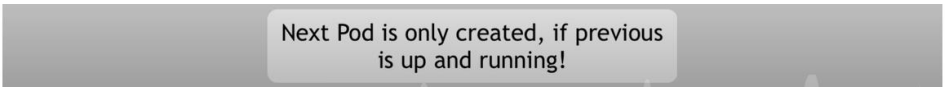

Scaling database applications
MASTER
WORKER
WORKER
Or Worker Nodes
mysql-0
mysql-1
mysql-2
writing
reading
writing
reading
reading
data replicas
data replicas

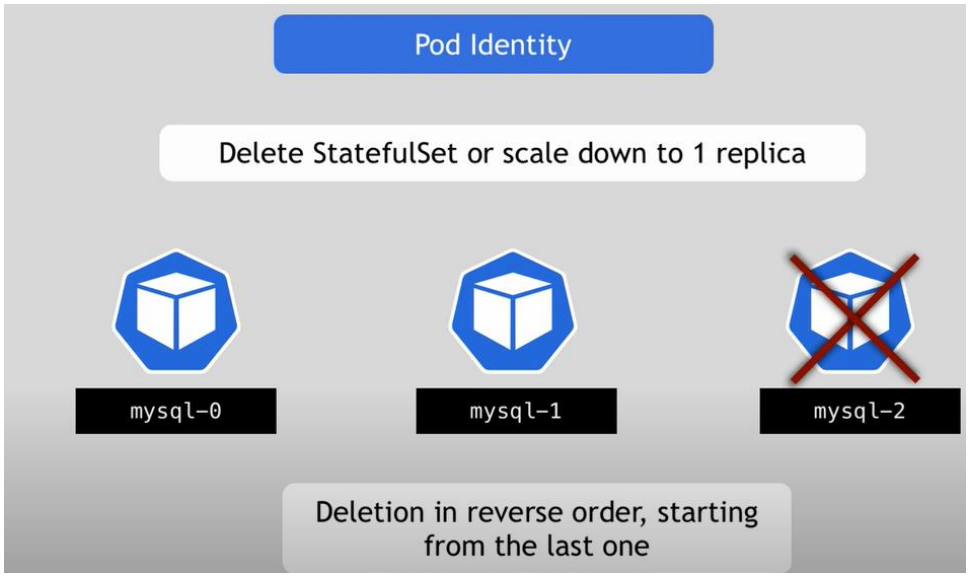

Scaling database applications
MASTER
WORKER
WORKER
mysql-0
mysql-1
mysql-2
They do not use the same physical storage!
PV
PV
PV
data A
data A
data A
/data/vol/pv-0
/data/vol/pv-1
/data/vol/pv-2

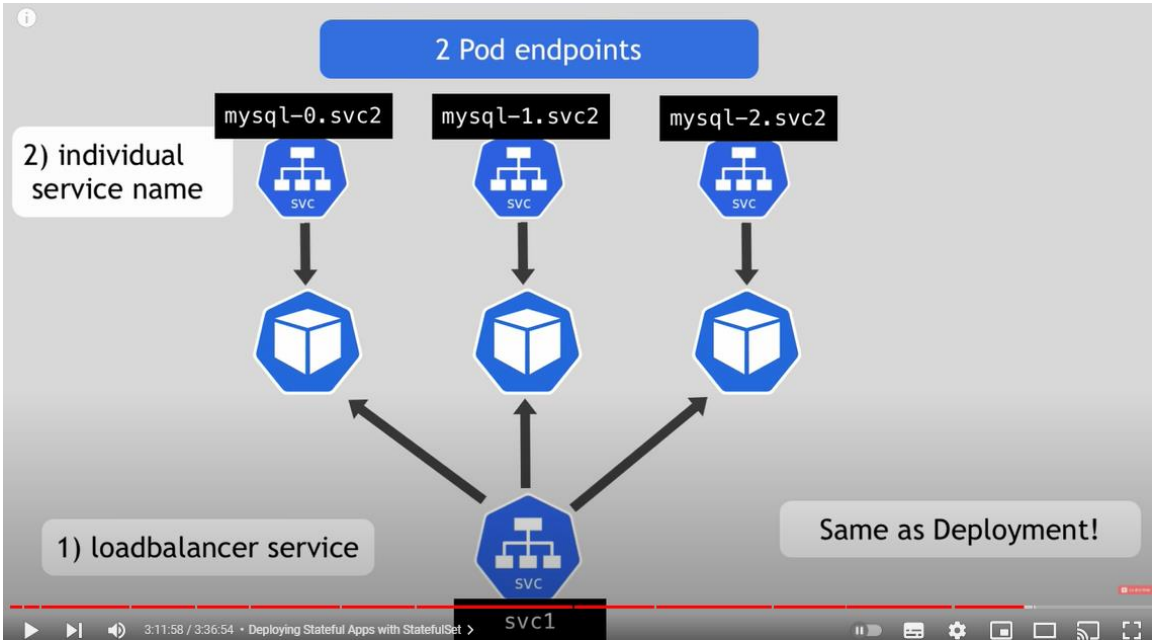

Pod state
ID-0
pv
when pod deleted,
new pod is coming
with same ID ans tate
pv
data A
Pod state
data A
Pod state

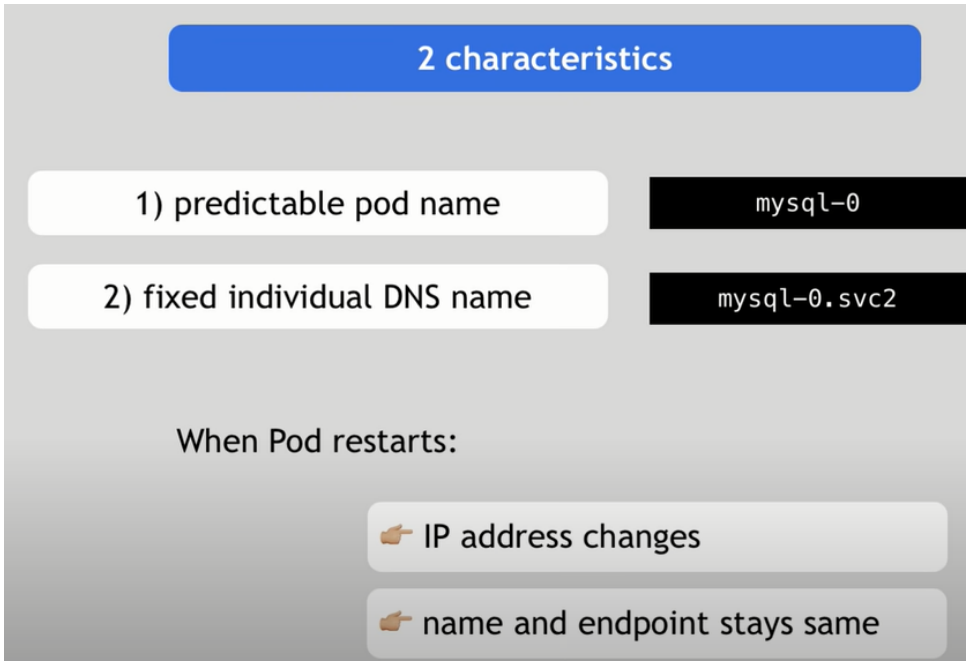

Pod state
pv
Remote storage!
to be available for other Nodes
data A
Pod state

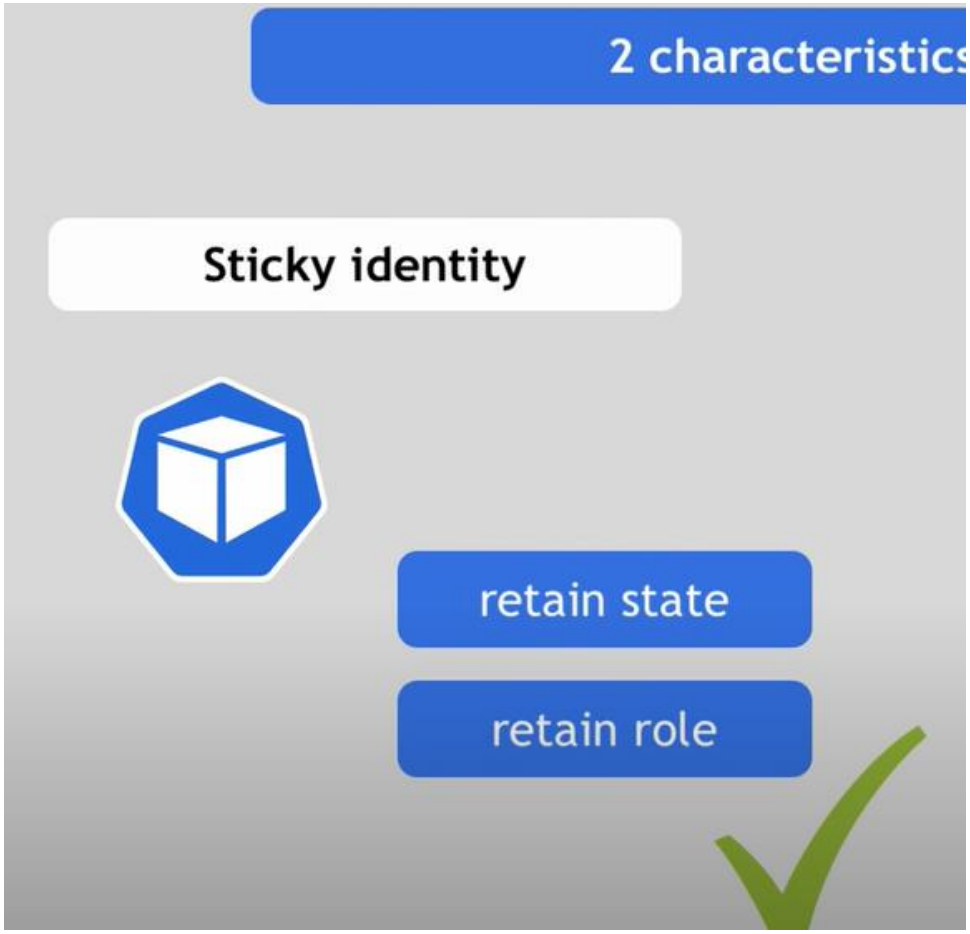

Pod Identity
Deployment:
random hash
StatefulSet:
fixed ordered names
mysql-c9e93d4e783165d
mysql-0
$(statefulset name)-$(ordinal)

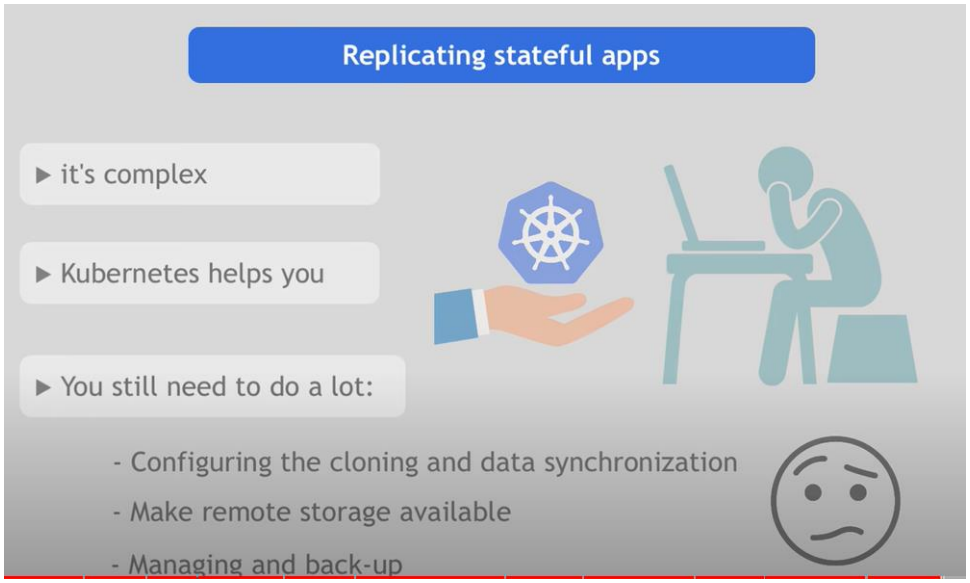

Next Pod is only created, if previous
is up and running!


Pod Identity
Delete StatefulSet or scale down to 1 replica
mysql-0
mysql-1
mysql-2
Deletion in reverse order, starting from the last one


2 Pod endpoints
mysql-0.svc2
mysql-1.svc2
mysql-2.svc2
2) individual service name
svc
svc
svc
1) loadbalancer service
svc
svc1
Same as Deployment!
3:11:58 / 3:36:54 • Deploying Stateful Apps with StatefulSet >


2 characteristics
1) predictable pod name
mysql-0
2) fixed individual DNS name
mysql-0.svc2
When Pod restarts:
IP address changes
name and endpoint stays same


2 characteristics
Sticky identity
retain state
retain role

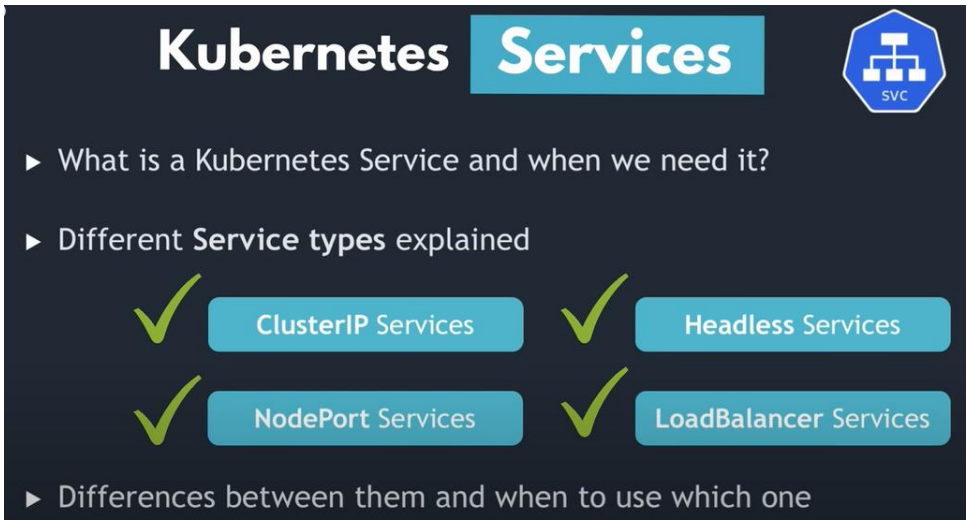


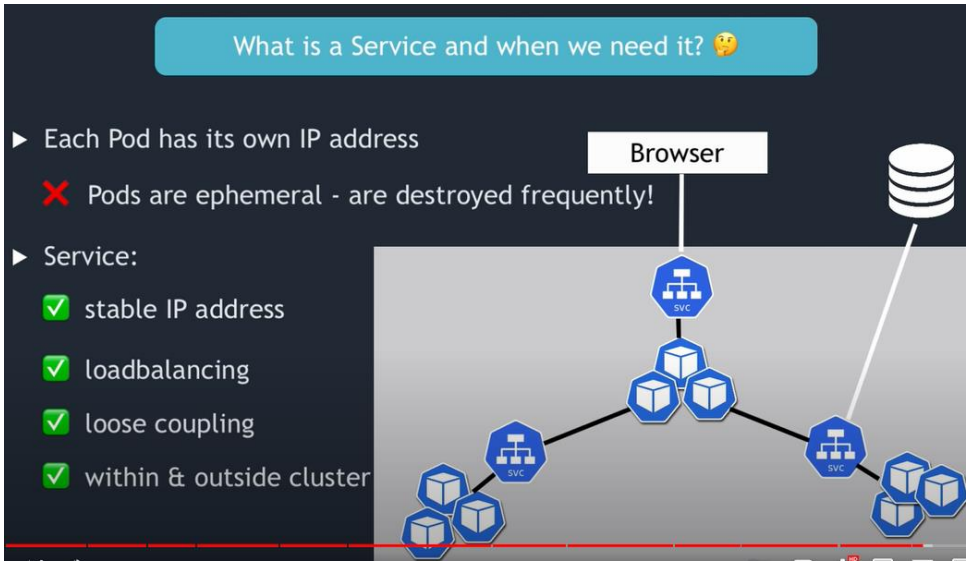
Services

Types are :

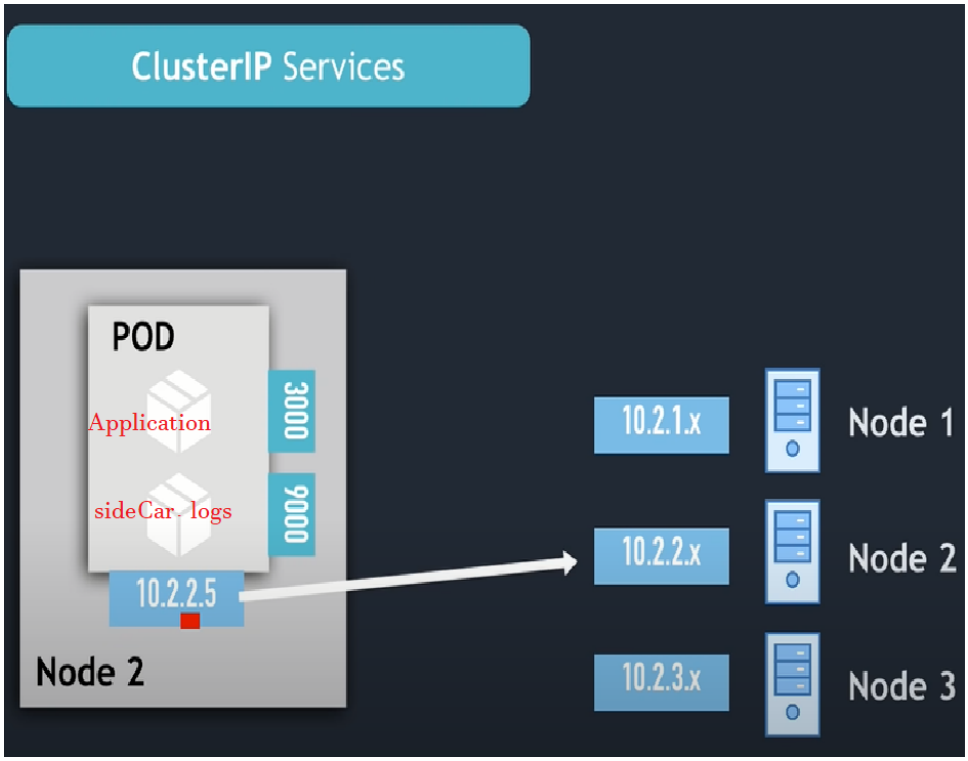
ClusterIP service (default)

Headless Service



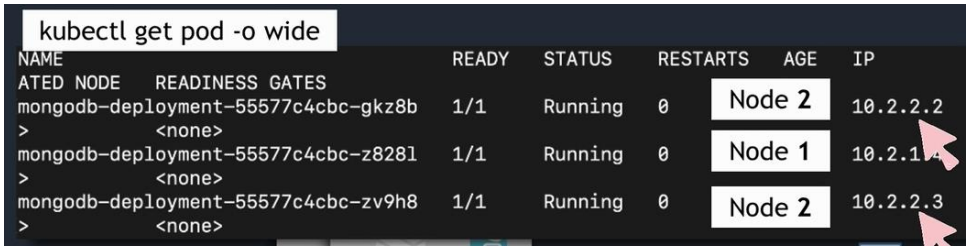


ClusterIP services

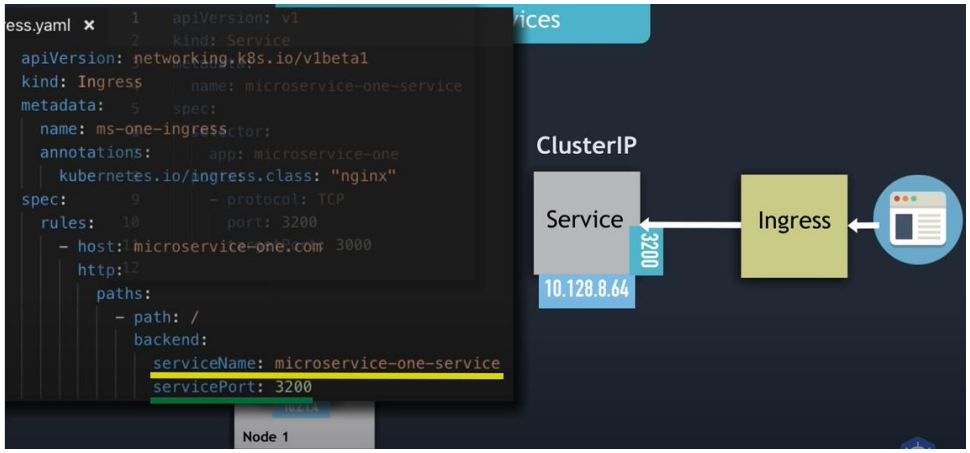


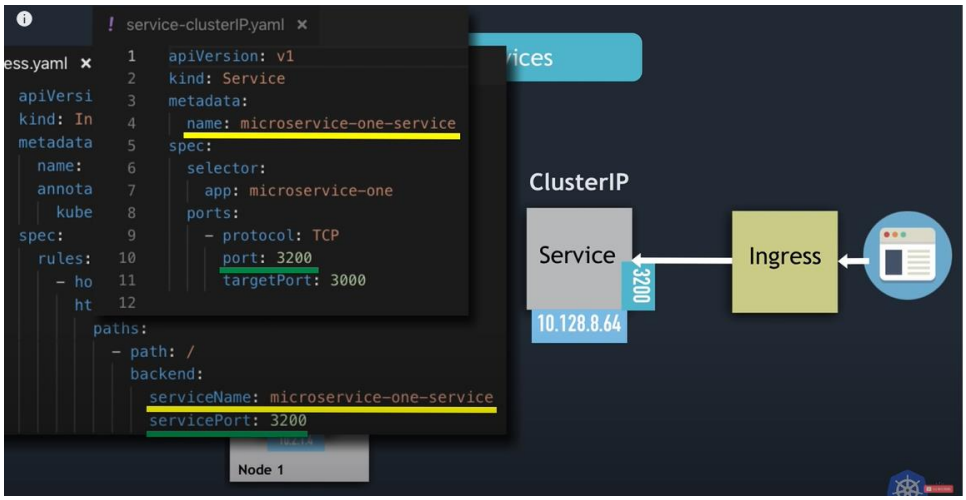
kubectl get pod -o wide

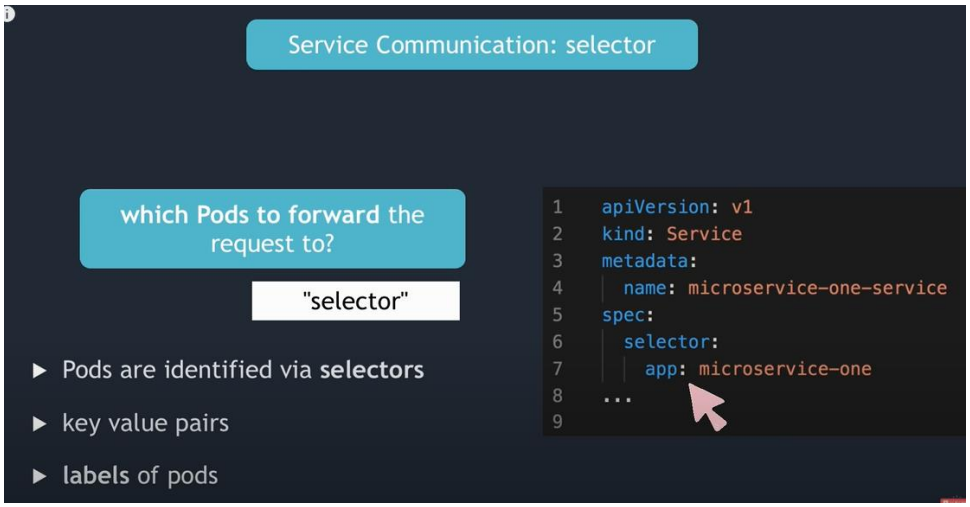
```
NAME                                    READY   STATUS    RESTARTS   AGE   IP
ATED NODE   READINESS GATES
mongodb-deployment-55577c4cbc-gkz8b     1/1     Running   0          29s   10.2.2.2
>           <none>
mongodb-deployment-55577c4cbc-z828l     1/1     Running   0          29s   10.2.1.4
>           <none>
mongodb-deployment-55577c4cbc-zv9h8     1/1     Running   0          29s   10.2.2.3
>           <none>
```



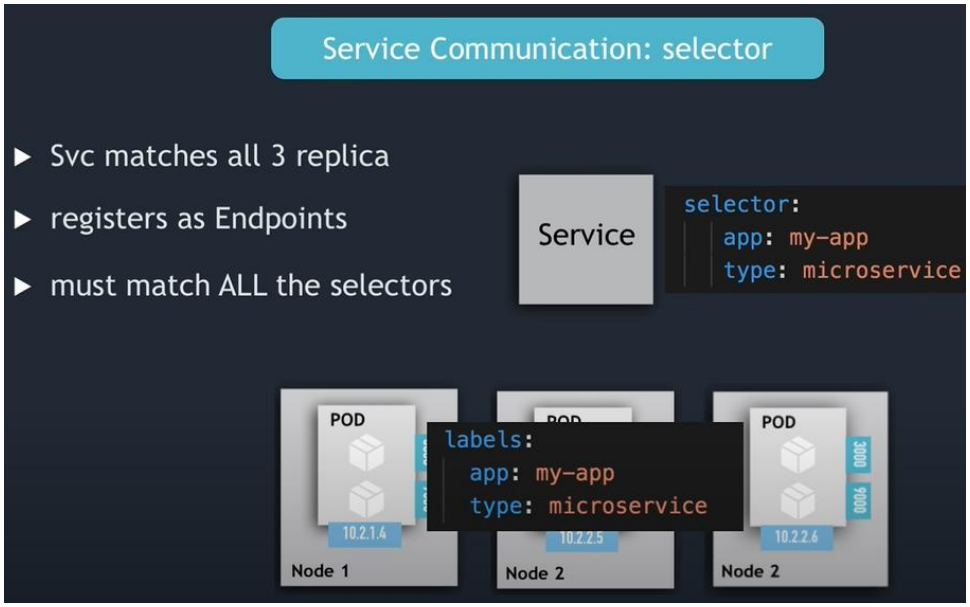
Setting ingress – forward the request to the service

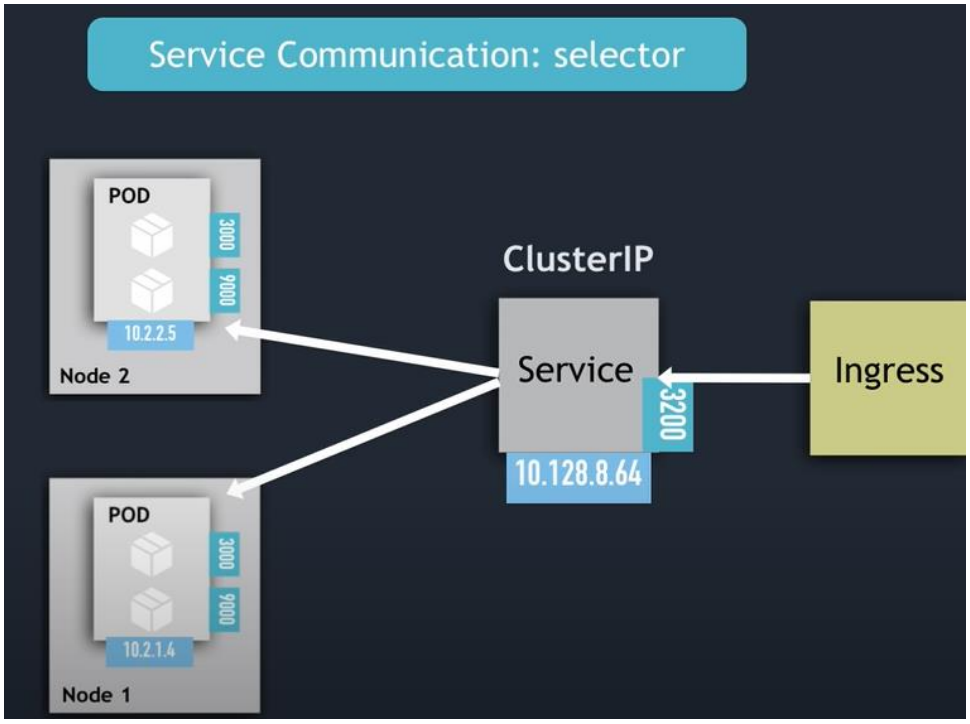

ess.yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: ms-one-ingress
annotations:
kubernetes.io/ingress.class: "nginx"
spec:
rules:
- host: microservice-one.com
http:
paths:
- path: /
backend:
serviceName: microservice-one-service
servicePort: 3200
Node 1
ClusterIP
Service
3200
10.128.8.64
Ingress

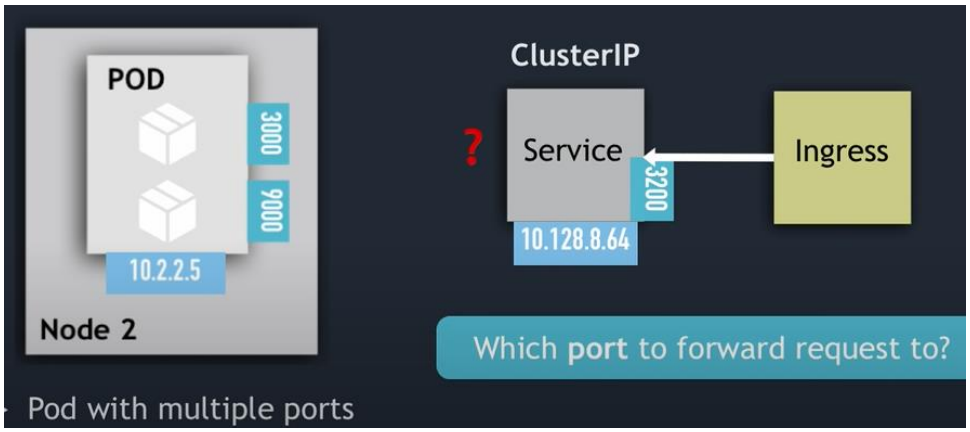

service-clusterIP.yaml
apiVersion: v1
kind: Service
metadata:
name: microservice-one-service
spec:
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
serviceName: microservice-one-service
servicePort: 3200
Node 1
ClusterIP
Service
3200
10.128.8.64
Ingress

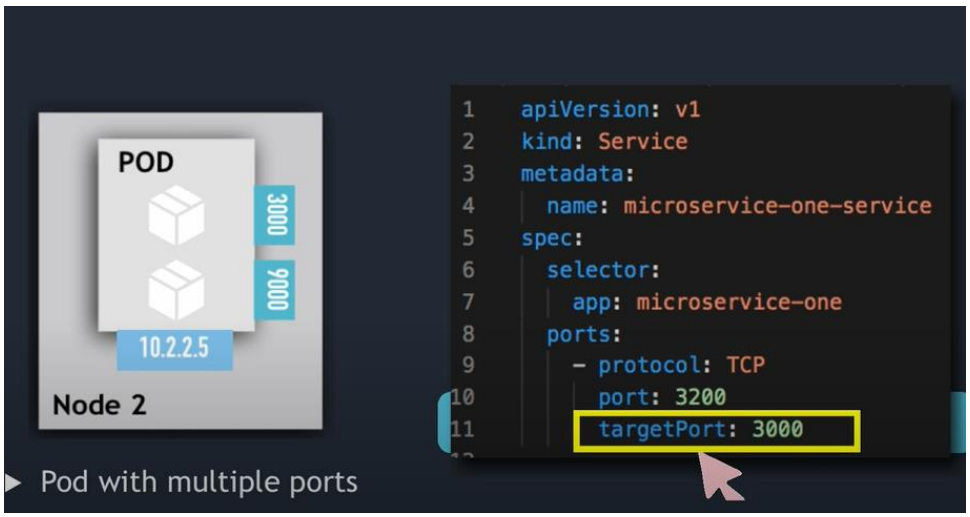

Service Communication: selector
which Pods to forward the request to?
"selector"
Pods are identified via selectors
key value pairs
labels of pods
apiVersion: v1
kind: Service
metadata:
name: microservice-one-service
spec:
selector:
app: microservice-one

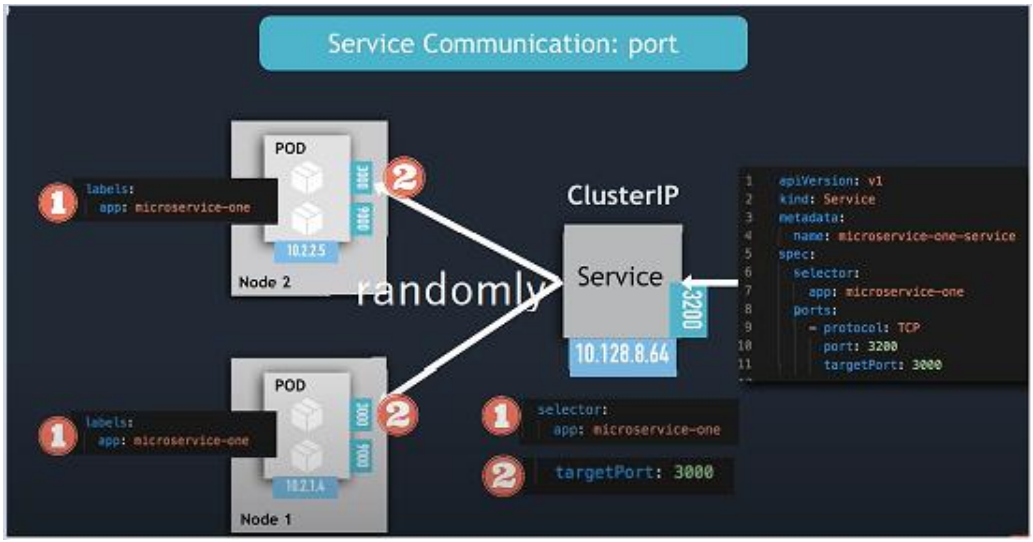

Service Communication: selector
whic
apiVersion: apps/v1
kind: Deployment
metadata:
name: microservice-one
...
spec:
replicas: 2
...
template:
metadata:
labels:
app: microservice-one
apiVersion: v1
kind: Service
metadata:
name: microservice-one-service
spec:
selector:
app: microservice-one
...
Pods are i
key value
labels of pods

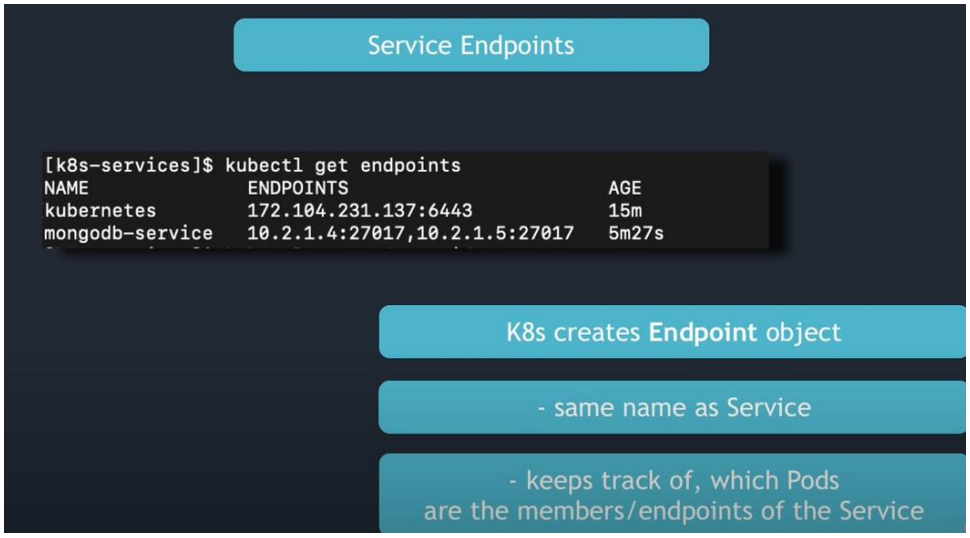

Service Communication: selector
Svc matches all 3 replica
registers as Endpoints
must match ALL the selectors
Service
selector:
app: my-app
type: microservice
POD
10.2.1.4
Node 1
labels:
app: my-app
type: microservice
10.2.2.5
Node 2
POD
3000
9000
10.2.2.6
Node 2

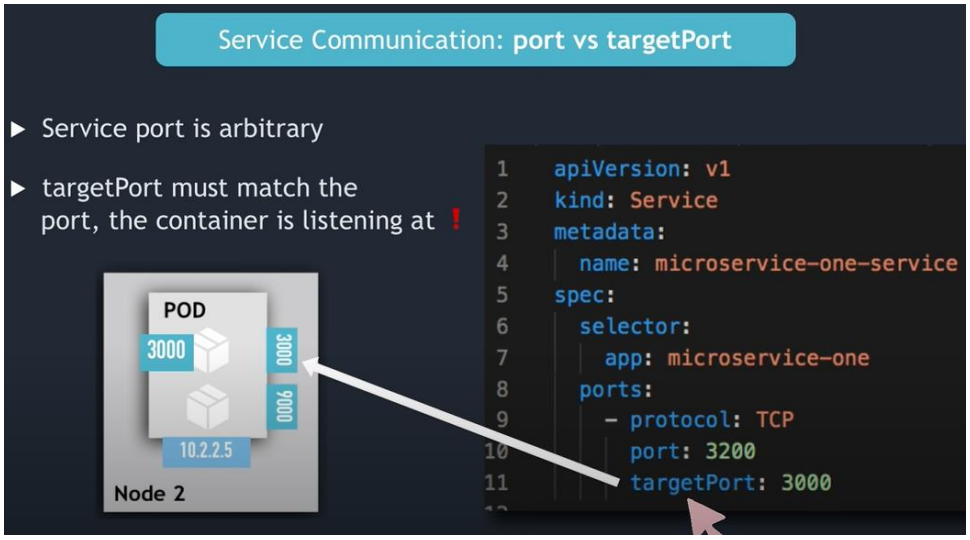

Service Communication: selector
POD
3000
9000
10.2.2.5
Node 2
ClusterIP
Service
3200
10.128.8.64
Ingress
POD
3000
9000
10.2.1.4
Node 1

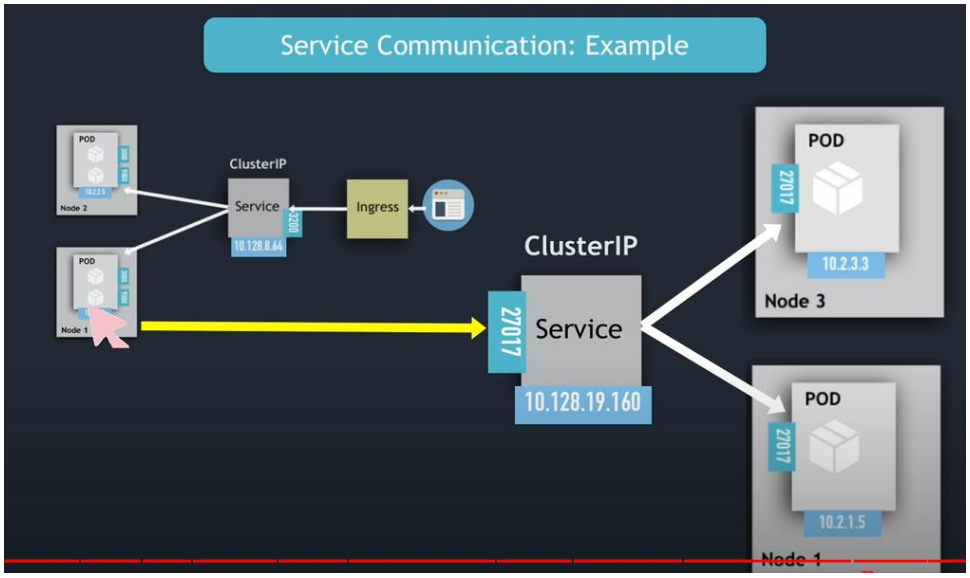

POD
3000
9000
10.2.2.5
Node 2
ClusterIP
?
Service
3200
10.128.8.64
Ingress
Which port to forward request to?
Pod with multiple ports

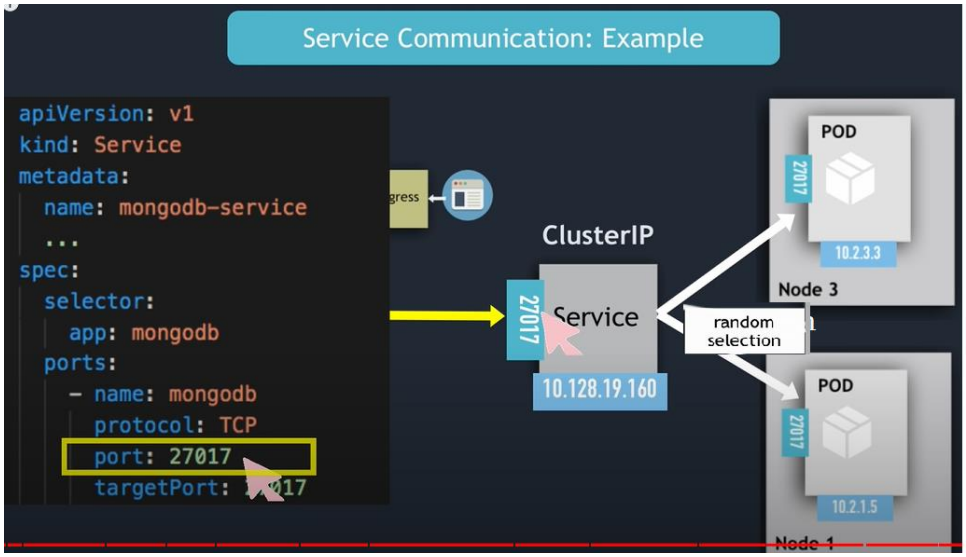

POD
3000
9000
10.2.2.5
Node 2
1 apiVersion: v1
2 kind: Service
3 metadata:
4 name: microservice-one-service
5 spec:
6 selector:
7 app: microservice-one
8 ports:
9 - protocol: TCP
10 port: 3200
11 targetPort: 3000
Pod with multiple ports

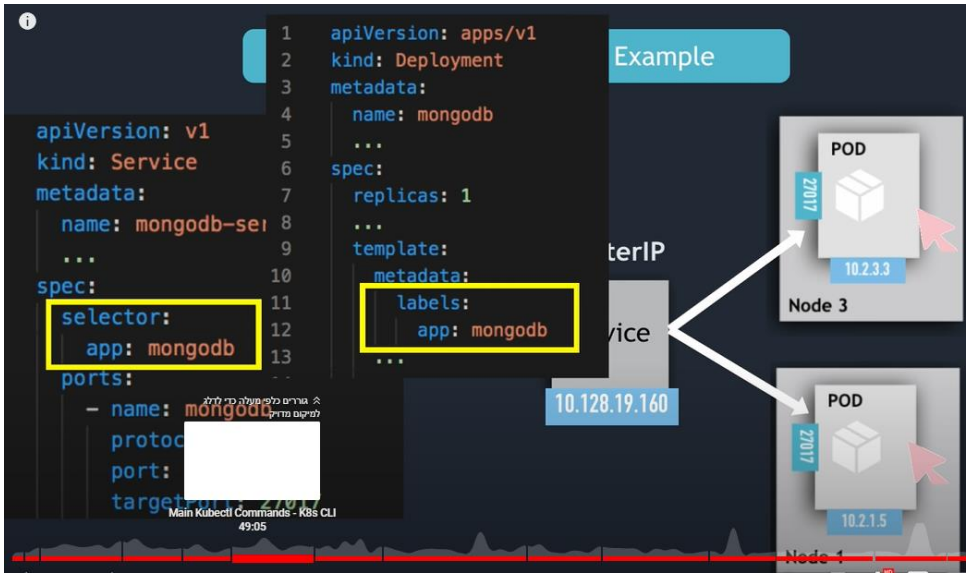

Service Communication: port
POD
labels:
app: microservice-one
3200
9100
10.2.2.5
Node 2
randomly
ClusterIP
Service
3200
10.128.8.64
apiVersion: v1
kind: Service
metadata:
name: microservice-one-service
spec:
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
POD
labels:
app: microservice-one
3000
9000
10.2.1.4
Node 1
selector:
app: microservice-one
targetPort: 3000

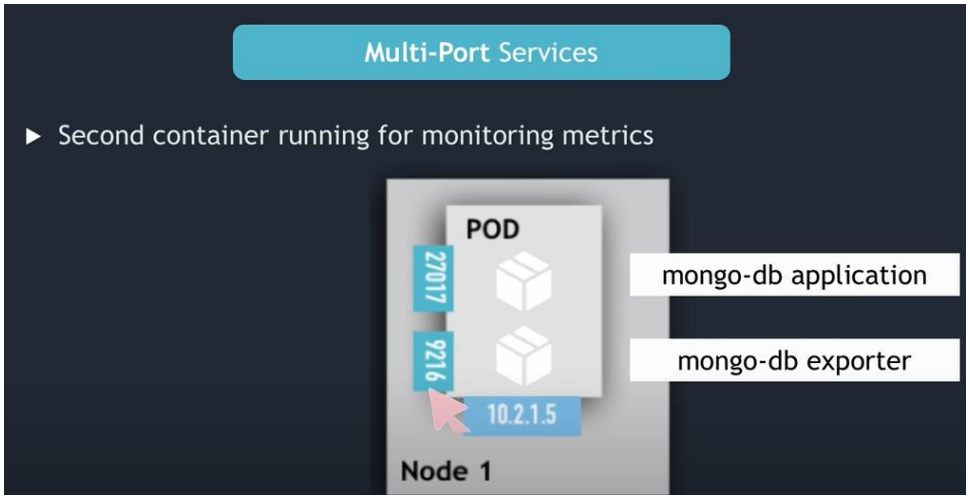

Service Endpoints
[k8s-services]$ kubectl get endpoints
NAME ENDPOINTS AGE
kubernetes 172.104.231.137:6443 15m
mongodb-service 10.2.1.4:27017,10.2.1.5:27017 5m27s
K8s creates Endpoint object
- same name as Service
- keeps track of, which Pods
are the members/endpoints of the Service

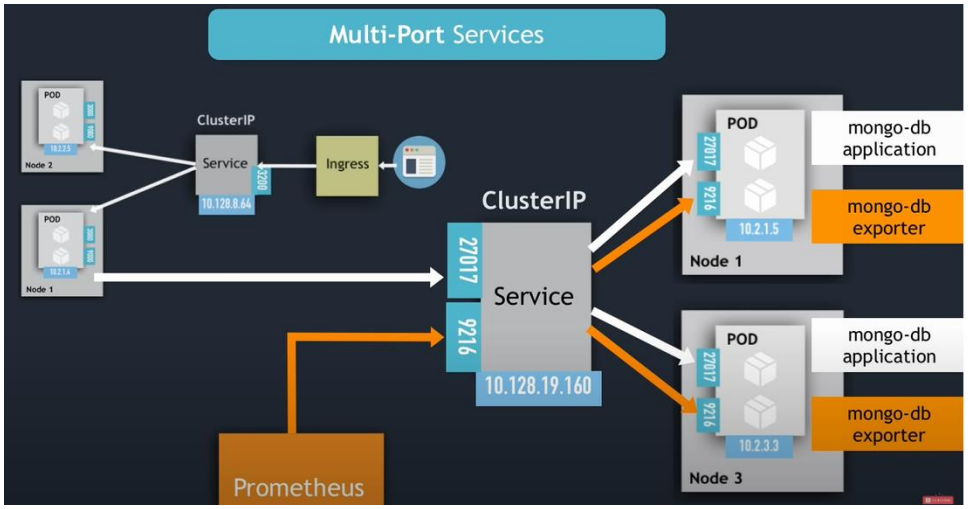

Service Communication: port vs targetPort
Service port is arbitrary
targetPort must match the
port, the container is listening at !
POD
3000
3000
9000
10.2.2.5
Node 2
apiVersion: v1
kind: Service
metadata:
name: microservice-one-service
spec:
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000

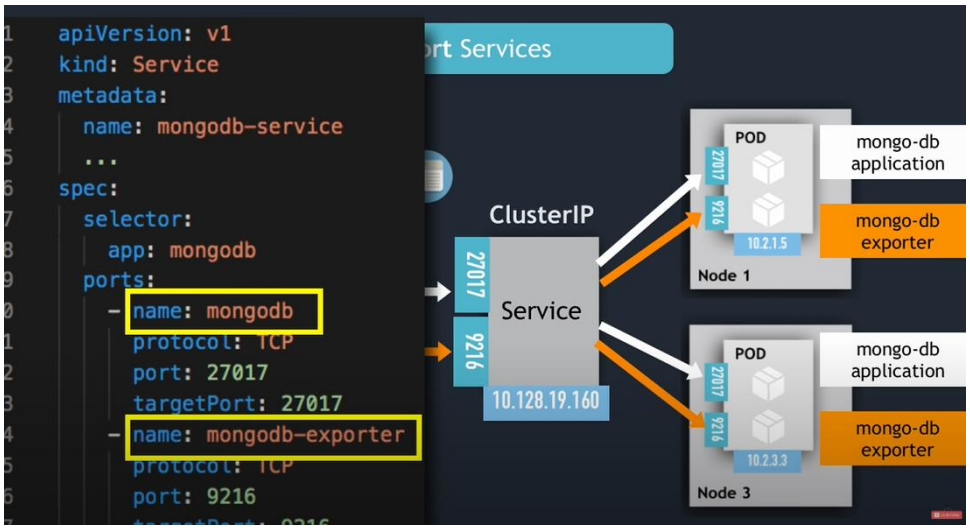

Service Communication: Example
POD
Node 2
ClusterIP
Service
Ingress
10.128.8.64
POD
Node 1
ClusterIP
27017
Service
10.128.19.160
POD
27017
10.2.3.3
Node 3
POD
27017
10.2.1.5
Node 1


Service Communication: Example
apiVersion: v1
kind: Service
metadata:
name: mongodb-service
...
spec:
selector:
app: mongodb
ports:
- name: mongodb
protocol: TCP
port: 27017
targetPort:
ClusterIP
27017
Service
10.128.19.160
random selection
POD
27017
10.2.3.3
Node 3
POD
27017
10.2.1.5
Node 1


apiVersion: apps/v1
kind: Deployment
metadata:
name: mongodb
...
spec:
replicas: 1
...
template:
metadata:
labels:
app: mongodb
...
apiVersion: v1
kind: Service
metadata:
name: mongodb-se
...
spec:
selector:
app: mongodb
ports:
- name: mongodb
protoc
port:
targetPort: 27017
Example
terIP
vice
10.128.19.160
POD
27017
10.2.3.3
Node 3
POD
27017
10.2.1.5
Node 1
Main Kubectl Commands - K8s CLI
49:05


Multi-Port Services
Second container running for monitoring metrics
POD
27017
9216
10.2.1.5
mongo-db application
mongo-db exporter
Node 1

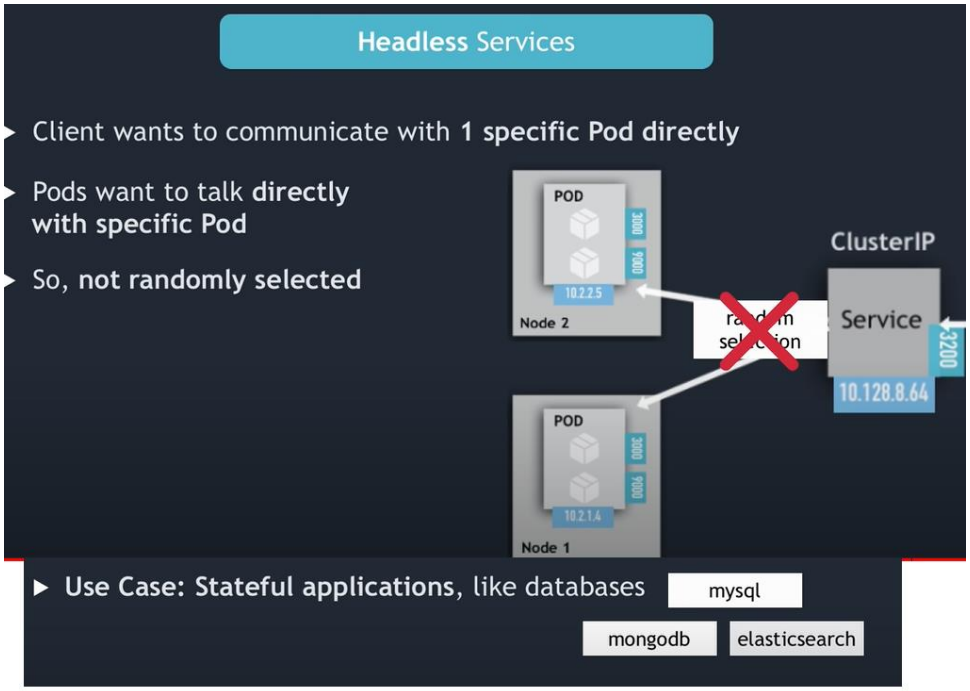

Multi-Port Services
POD
Node 2
POD
Node 1
ClusterIP
Service
Ingress
10.128.8.64
ClusterIP
27017
9216
Service
10.128.19.160
Prometheus
POD
10.2.1.5
Node 1
POD
10.2.3.3
Node 3
mongo-db application
mongo-db exporter

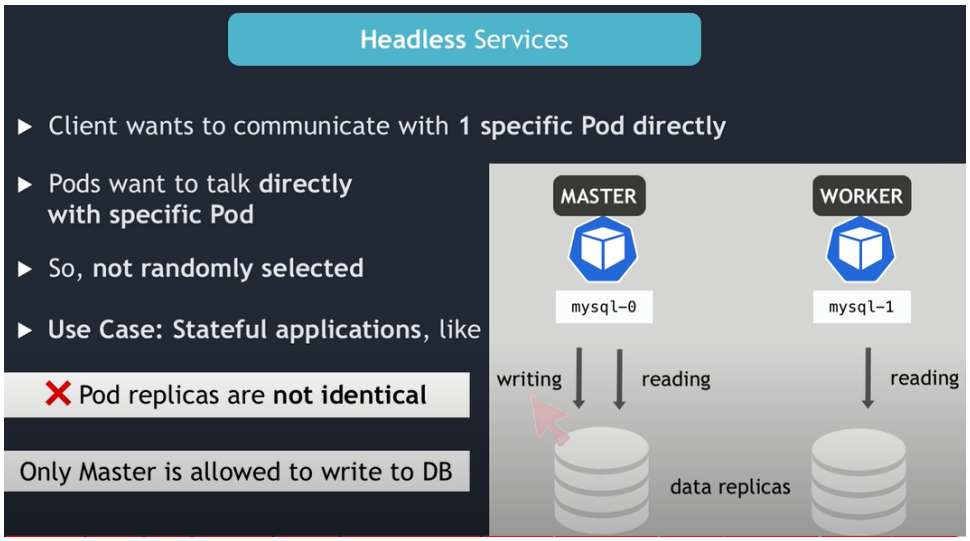

apiVersion: v1
kind: Service
metadata:
name: mongodb-service
...
spec:
selector:
app: mongodb
ports:
- name: mongodb
protocol: TCP
port: 27017
targetPort: 27017
- name: mongodb-exporter
protocol: TCP
port: 9216
ClusterIP
27017
9216
Service
10.128.19.160
POD
10.2.1.5
Node 1
POD
10.2.3.3
Node 3
mongo-db application
mongo-db exporter

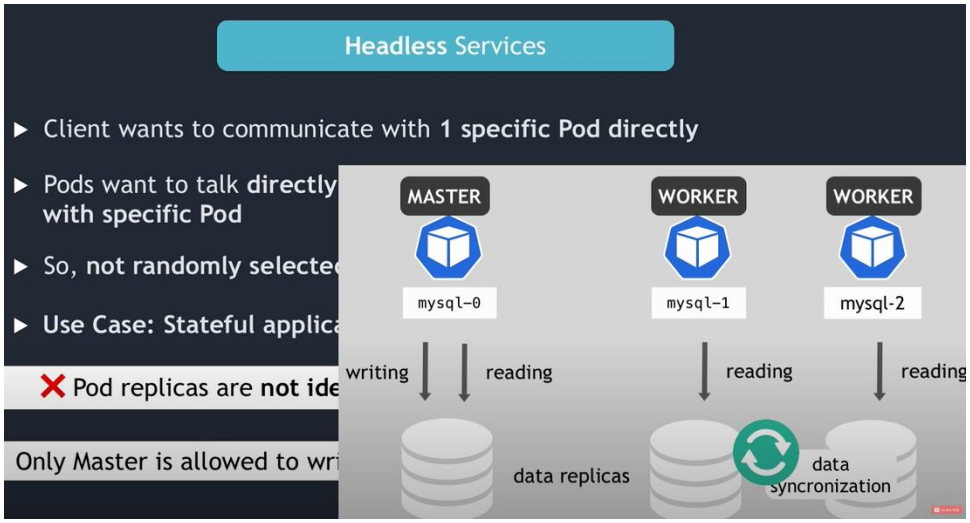
Headless Service

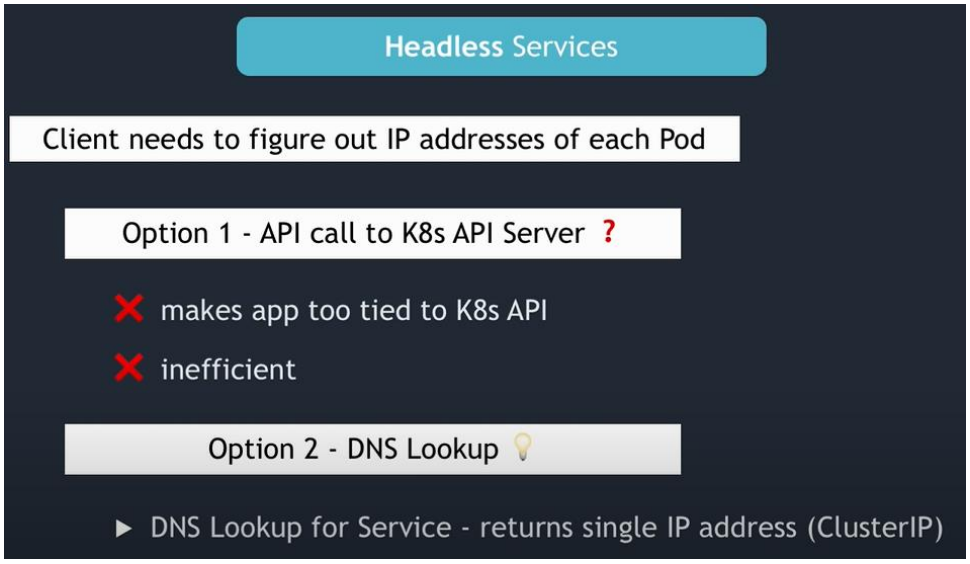
Use case:

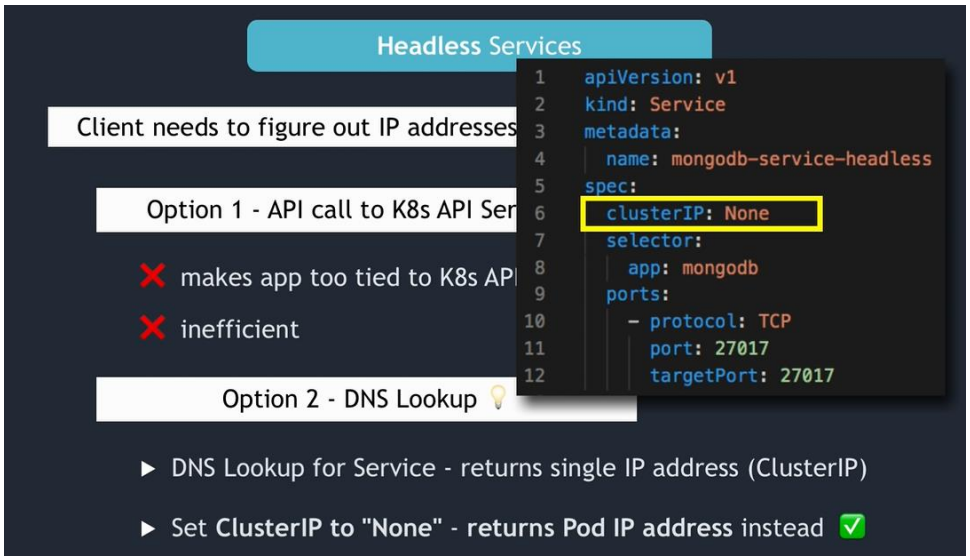
1. Client need to connect directly to the pod (DB worker pod)
2. DB Worker pods to master pod (for synchronization)
3. Master pod to perform writecommands to DB

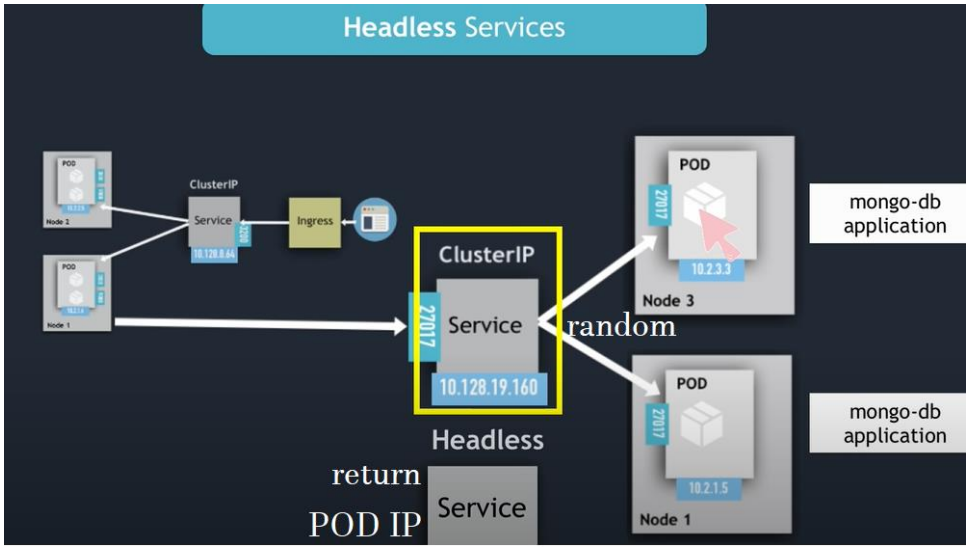


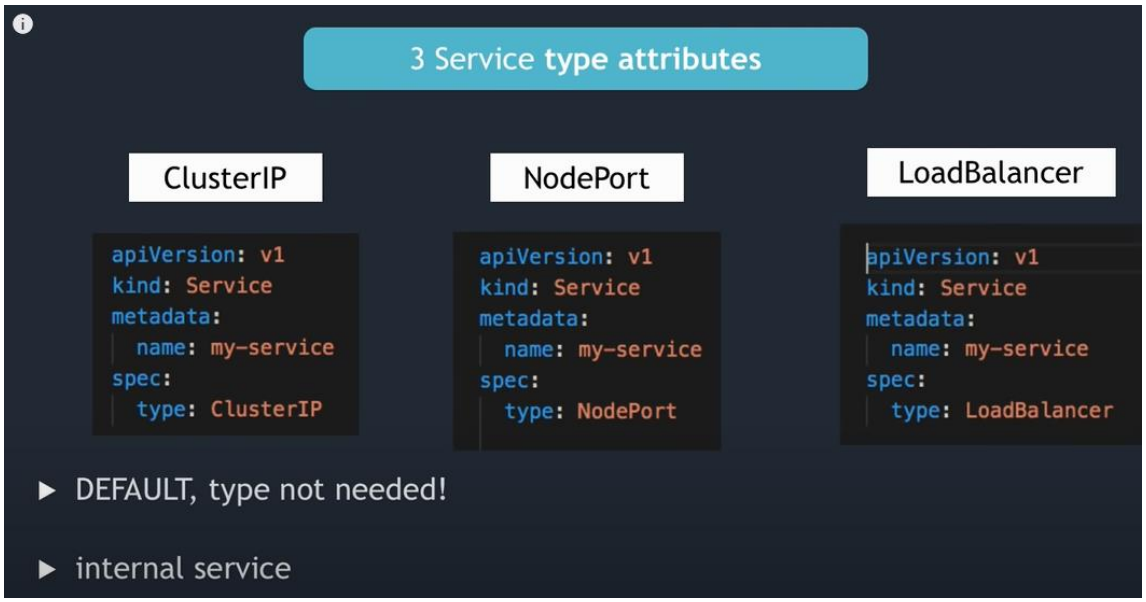



Headless Services
Client wants to communicate with 1 specific Pod directly
Pods want to talk directly
with specific Pod
So, not randomly selecte
Use Case: Stateful applic
Pod replicas are not ide
Only Master is allowed to wr
MASTER
WORKER
WORKER
mysql-0
mysql-1
mysql-2
writing
reading
reading
reading
data replicas
data
syncronization

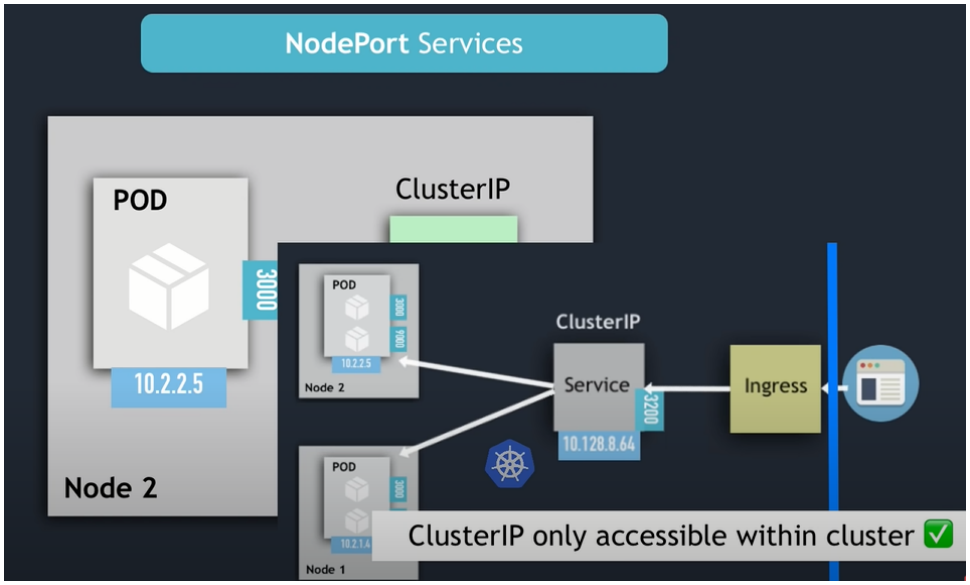

Headless Services
Client needs to figure out IP addresses of each Pod
Option 1 - API call to K8s API Server ?
makes app too tied to K8s API
inefficient
Option 2 - DNS Lookup
DNS Lookup for Service - returns single IP address (ClusterIP)


Headless Services
Client needs to figure out IP addresses
Option 1 - API call to K8s API Ser
makes app too tied to K8s API
inefficient
Option 2 - DNS Lookup
apiVersion: v1
kind: Service
metadata:
name: mongodb-service-headless
spec:
clusterIP: None
selector:
app: mongodb
ports:
- protocol: TCP
port: 27017
targetPort: 27017
DNS Lookup for Service - returns single IP address (ClusterIP)
Set ClusterIP to "None" - returns Pod IP address instead

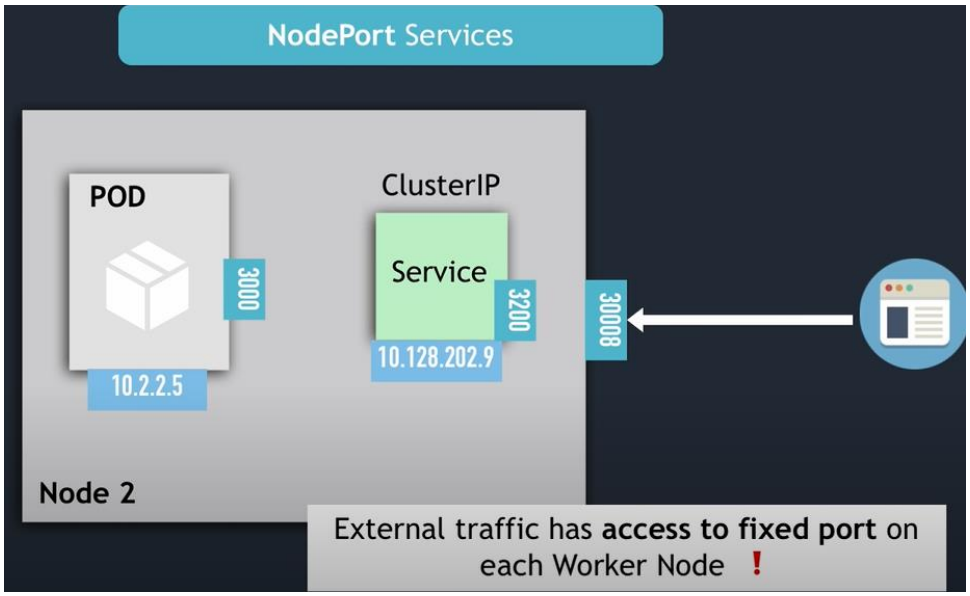

Headless Services
ClusterIP
Service
Ingress
ClusterIP
Service
27017
10.128.19.160
random
POD
27017
10.2.3.3
Node 3
mongo-db application
POD
27017
10.2.1.5
Node 1
mongo-db application
Headless
return
POD IP
Service

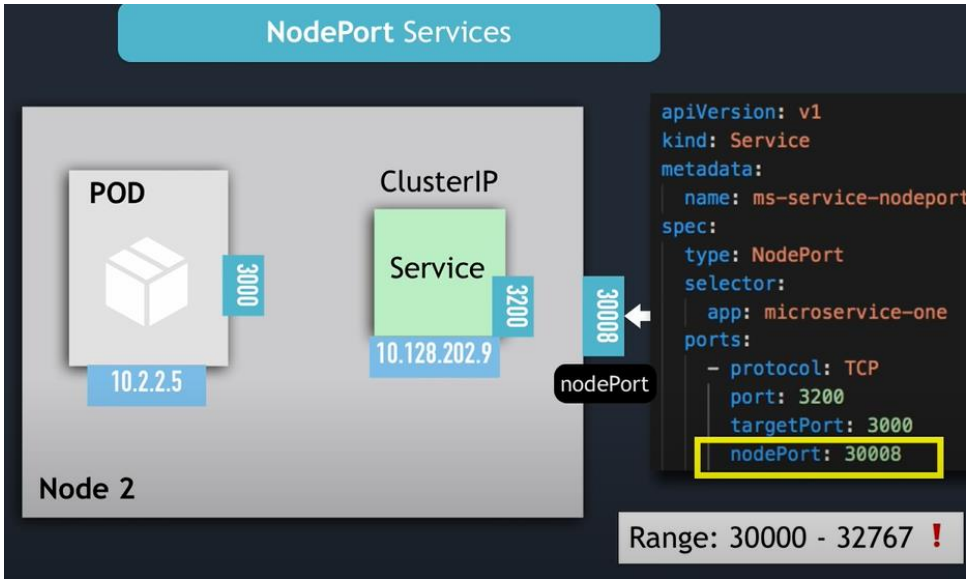

3 Service type attributes
ClusterIP
NodePort
LoadBalancer
apiVersion: v1
kind: Service
metadata:
name: my-service
spec:
type: ClusterIP
apiVersion: v1
kind: Service
metadata:
name: my-service
spec:
type: NodePort
apiVersion: v1
kind: Service
metadata:
name: my-service
spec:
type: LoadBalancer
DEFAULT, type not needed!
internal service

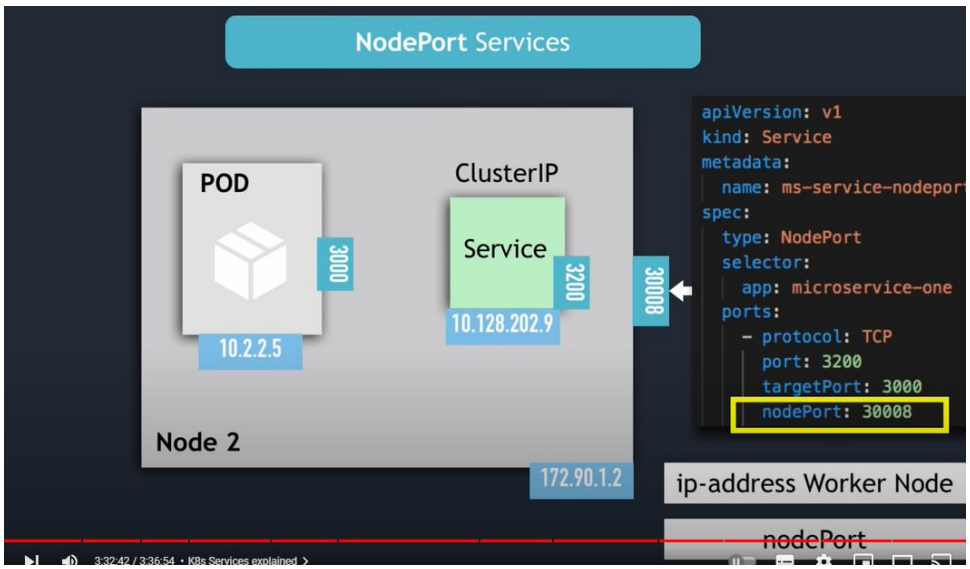
ClusterIP

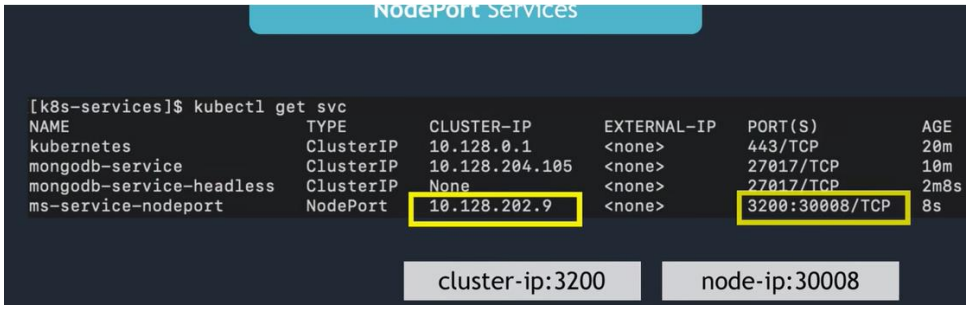

NodePort Services
POD
10.2.2.5
3000
ClusterIP
POD
Node 2
ClusterIP
Service
10.128.8.64
3200
Ingress
POD
Node 1
Node 2
ClusterIP only accessible within cluster

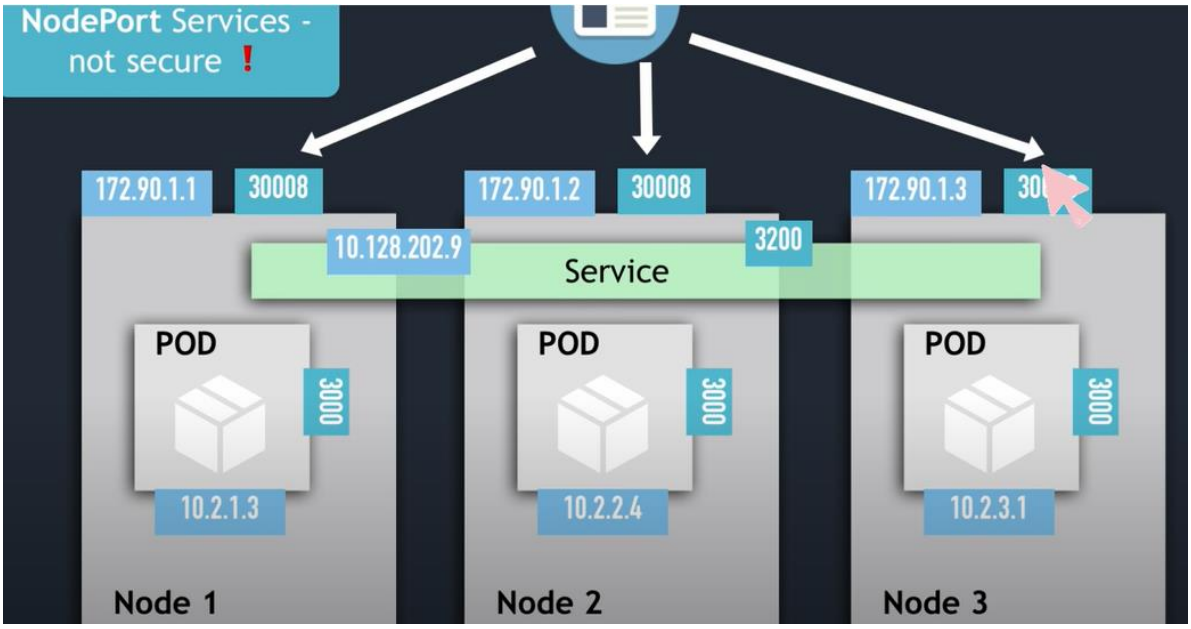
NodePort

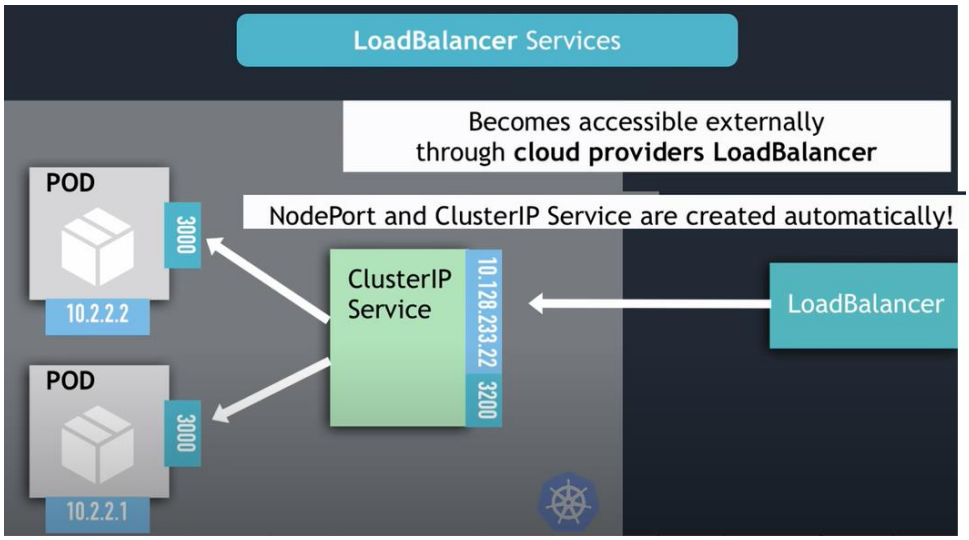

NodePort Services
POD
3000
10.2.2.5
ClusterIP
Service
3200
10.128.202.9
30008
Node 2
External traffic has access to fixed port on each Worker Node !

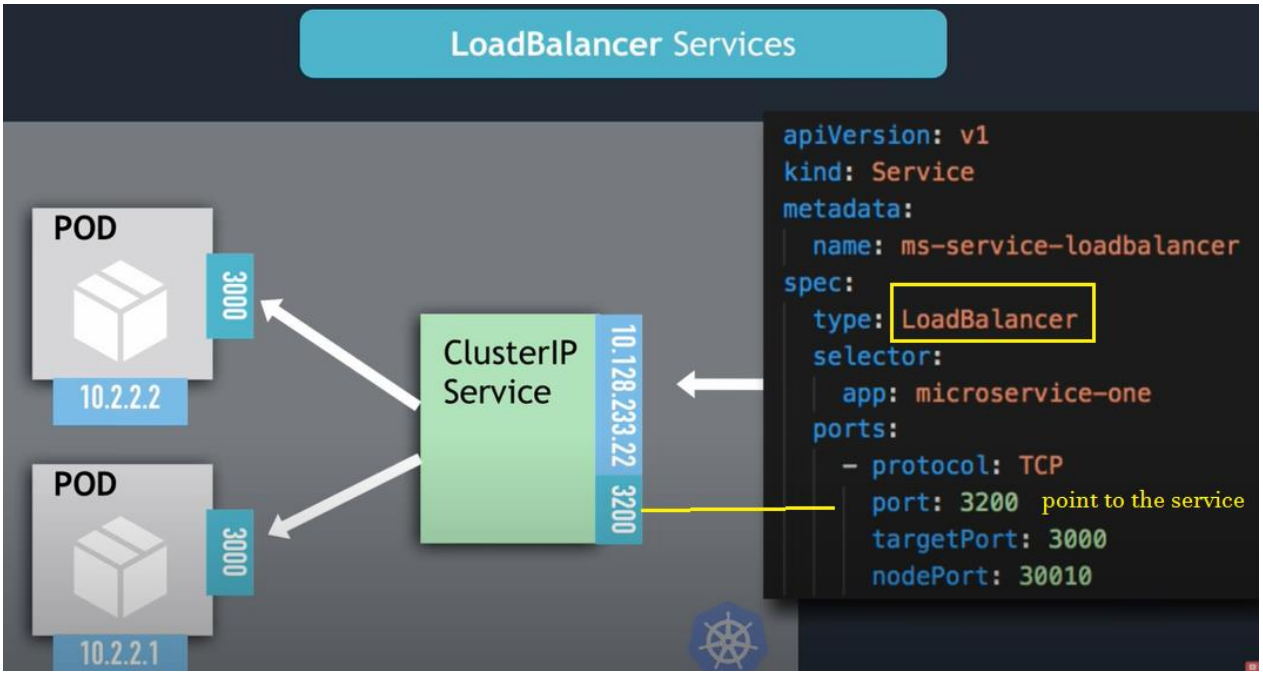

NodePort Services
POD
3000
10.2.2.5
ClusterIP
Service
3200
10.128.202.9
30008
nodePort
Node 2
apiVersion: v1
kind: Service
metadata:
name: ms-service-nodeport
spec:
type: NodePort
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
nodePort: 30008
Range: 30000 - 32767 !

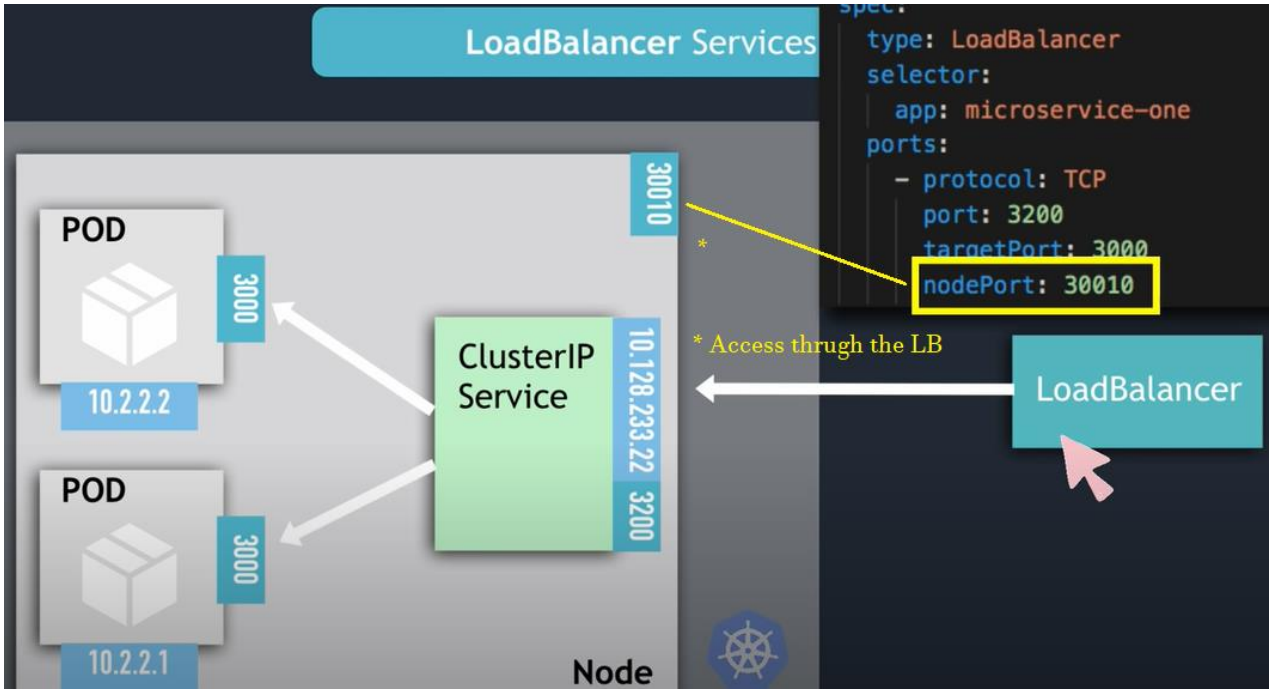

NodePort Services
POD
3000
10.2.2.5
ClusterIP
Service
3200
10.128.202.9
30008
Node 2
172.90.1.2
ip-address Worker Node
nodePort
apiVersion: v1
kind: Service
metadata:
name: ms-service-nodeport
spec:
type: NodePort
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
nodePort: 30008

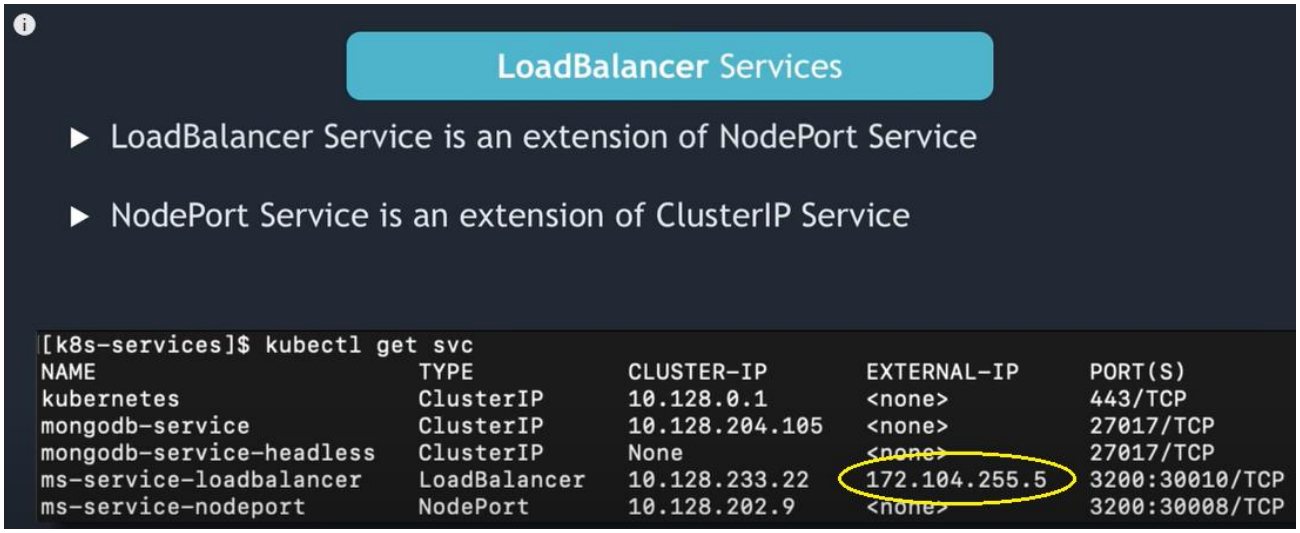

NodePort Services
[k8s-services]$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.128.0.1 <none> 443/TCP 20m
mongodb-service ClusterIP 10.128.204.105 <none> 27017/TCP 10m
mongodb-service-headless ClusterIP None <none> 27017/TCP 2m8s
ms-service-nodeport NodePort 10.128.202.9 <none> 3200:30008/TCP 8s
cluster-ip:3200
node-ip:30008

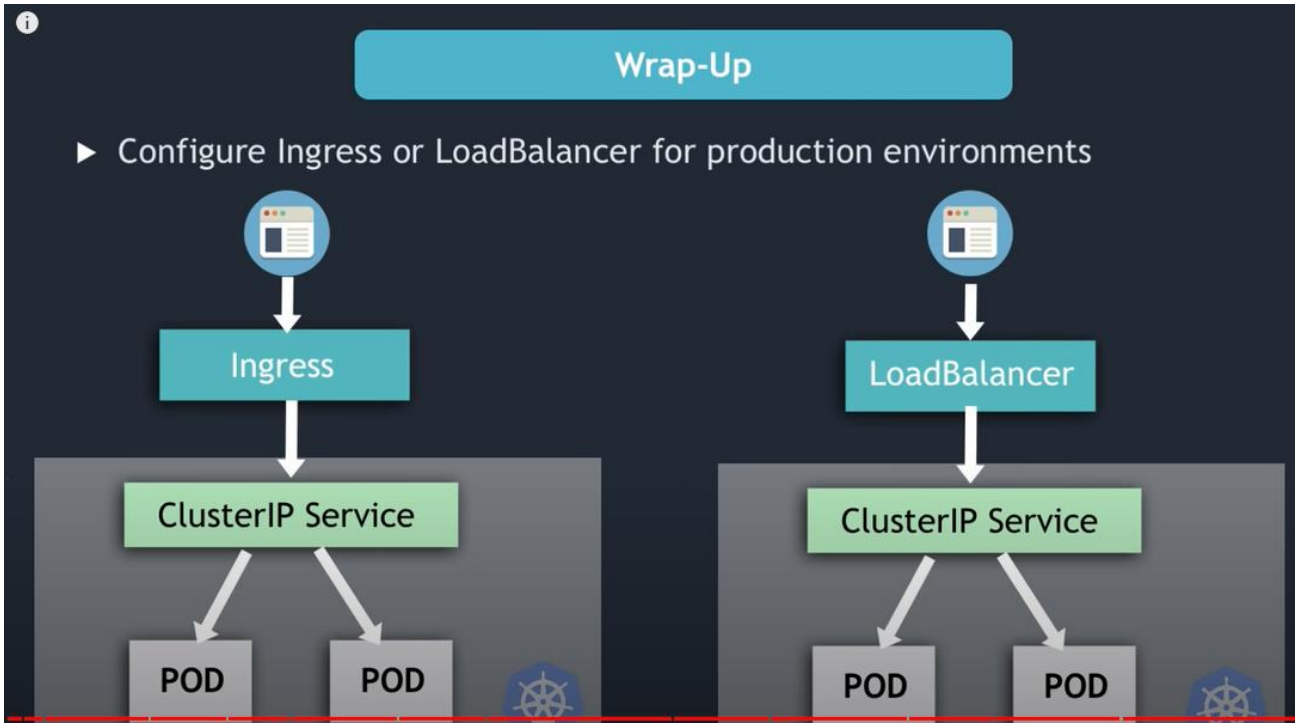

NodePort Services - not secure !
172.90.1.1
30008
172.90.1.2
30008
172.90.1.3
10.128.202.9
3200
Service
POD
3000
10.2.1.3
10.2.2.4
10.2.3.1
Node 1
Node 2
Node 3


LoadBalancer Services
Becomes accessible externally through cloud providers LoadBalancer
NodePort and ClusterIP Service are created automatically!
POD
3000
10.2.2.2
POD
3000
10.2.2.1
ClusterIP Service
10.128.233.22
3200
LoadBalancer


LoadBalancer Services
POD
3000
10.2.2.2
POD
3000
10.2.2.1
ClusterIP Service
10.128.233.22
3200
apiVersion: v1
kind: Service
metadata:
name: ms-service-loadbalancer
spec:
type: LoadBalancer
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
point to the service
targetPort: 3000
nodePort: 30010


LoadBalancer Services
type: LoadBalancer
selector:
app: microservice-one
ports:
- protocol: TCP
port: 3200
targetPort: 3000
nodePort: 30010
30010
*
* Access thrugh the LB
LoadBalancer
POD
3000
10.2.2.2
POD
3000
10.2.2.1
ClusterIP Service
10.128.233.22
3200
Node


LoadBalancer Services
▶ LoadBalancer Service is an extension of NodePort Service
▶ NodePort Service is an extension of ClusterIP Service
[k8s-services]$ kubectl get svc
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


Wrap-Up
▶ Configure Ingress or LoadBalancer for production environments
Ingress
ClusterIP Service
POD
POD
LoadBalancer
ClusterIP Service
POD
POD

HELM

HPA – scaling Horizontally

In Kubernetes, you can use the Horizontal Pod Autoscaler (HPA) to automatically scale the number of replicas of a deployment or a statefulset based on metrics such as CPU utilization or memory usage.

To set up an HPA, you need to define a metric to monitor and set a target value for the metric. For example, you can configure the HPA to monitor CPU utilization and set a target value of 70%.

Once the HPA is set up, Kubernetes will automatically monitor the metric and adjust the number of replicas accordingly. If the CPU utilization goes above the target value, Kubernetes will add more replicas, and if it goes below the target value, Kubernetes will reduce the number of replicas.

Here is an example manifest for setting up an HPA:

yamlCopy code

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app-deployment
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 70
```

In this example, the HPA is configured to monitor CPU utilization and adjust the number of replicas of a deployment named **`my-app-deployment`**. The minimum number of replicas is set to 2, and the maximum number of replicas is set to 10. The target CPU utilization is set to 70%.

With this configuration, Kubernetes will automatically adjust the number of replicas of the deployment based on the CPU utilization, ensuring that the application can handle changes in workload automatically.