9 What is Kubernetes?

- Open source container orchestration tools
- Developed by google
- Helps us manage containerized applications in different environments like physical machine, virtual machine or cloud environment

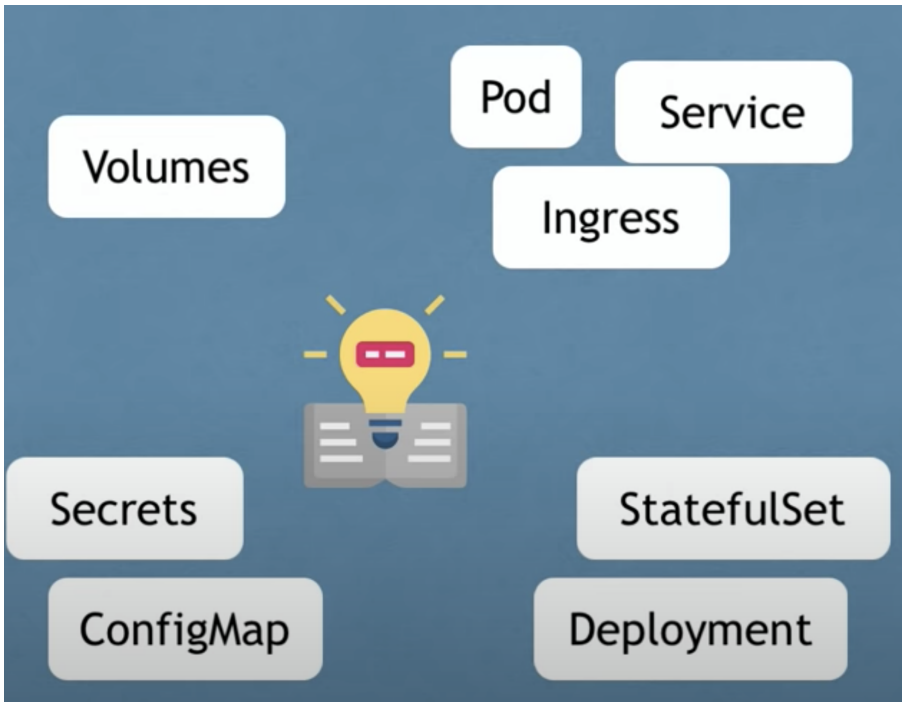
What problem does k8s solve?

- Trend for monolith to microservices
- Increased usage of containers
- Demand for proper managing those hundreds of containers

What features does k8s provide?

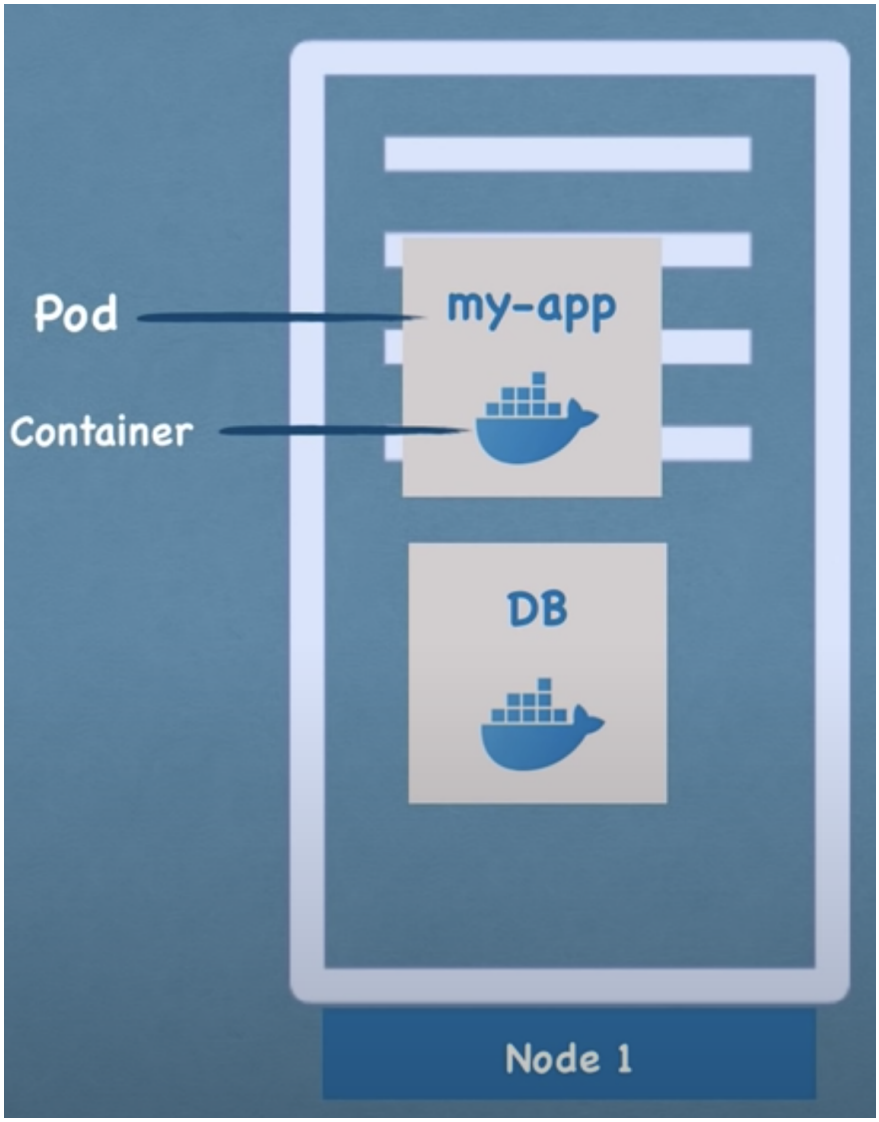
- High availability or no downtime
- Scalability or high performance
- Disaster recovery - backup and restore

Kubernetes components:



Nodes and Pods

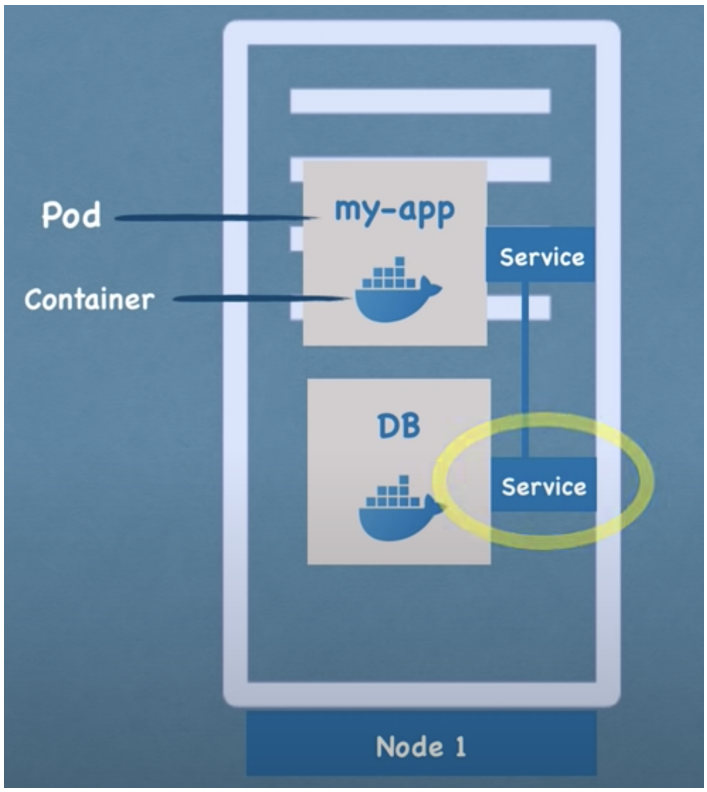
- Smallest unit of k8s
- Abstraction over container
- Usually 1 application per pod
- One pod can have more than one container but usually in one pod apart from main application, it will have helper services or side services in other containers



- Kubernetes offers out of the box a virtual network which means each pod gets its own ip address. Its not the container but the pod gets the ip address
- Each pod can communicate with each other using that ip address which is an internal ip address
- Pods in the kubernetes cluster are ephemeral. They can die very easily.
- For ex. when we lose a db pod because the container crashed inside or the node where the pod is running ran out of resources. When that happens the pod will crash and then restarts. In this case, a new ip will get assigned to the new pod.
- It will be difficult to manage ip addresses of pods every time it restarts. So there is another k8s component called Service is used

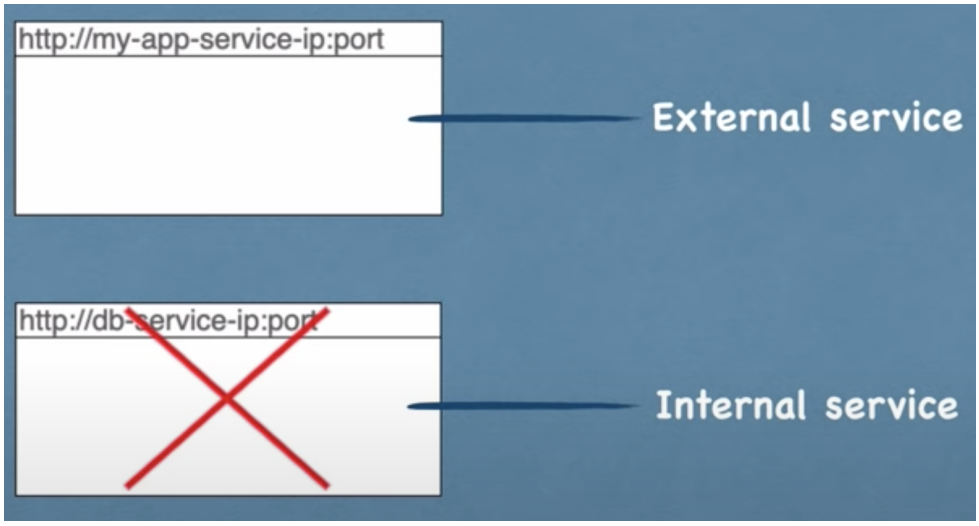
Service and Ingress:

- Permanent IP address that can be attached to each pod



- In above diagram, myapp will have its own service and db will have its own service
- Lifecycle of service and pod are not connected. So even if pod dies, the service and its ip address will stay

- To make our application available in browser, we will need to create external service
- External service is a service that opens the communication from external sources
- We can specify internal or external service while creating one.



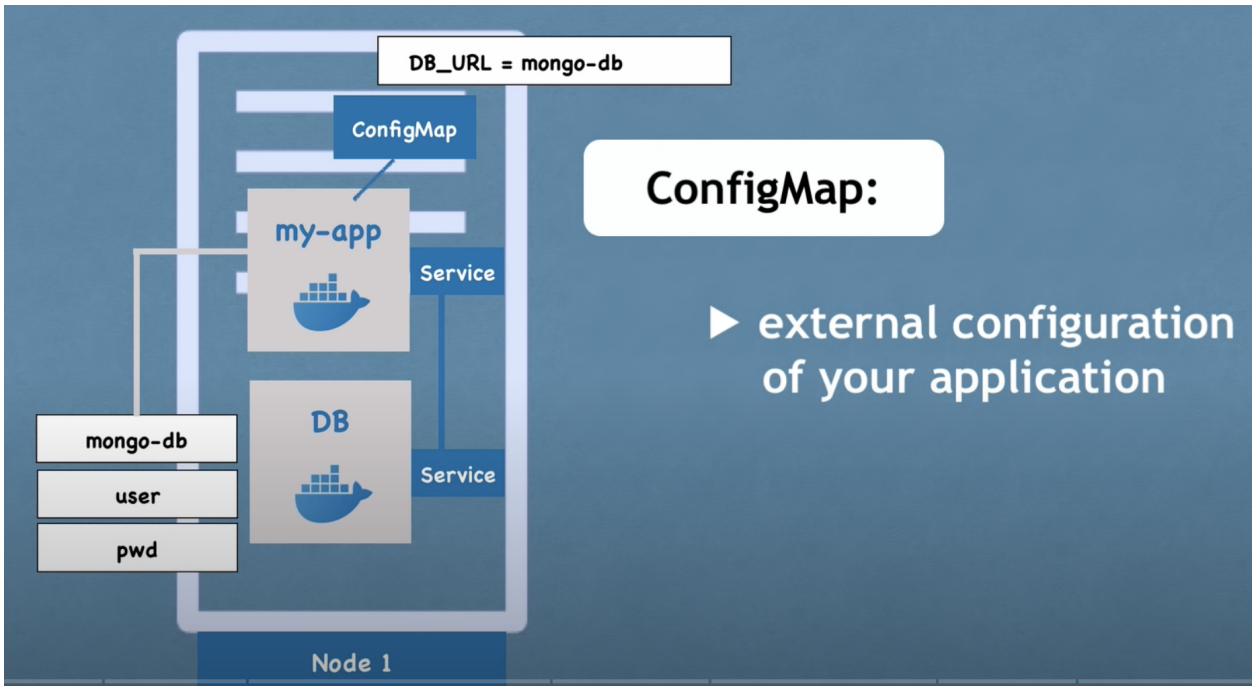
- We cannot expose external service with http schema and IP & port. So k8s provide another component called Ingress to communicate via secure protocol and domain name.
- So instead of service the request first goes to ingress then it does the forwarding to the service

Config Map and Secret

- Pods communicate with each other using a service
- For ex. An application have a database endpoint called mongodb-service that the app uses to communicate with the database. To communicate with the database we usually store the config in configuration file or external environment variables. But its usually inside the built image of the application.
- If the endpoint of the db service or service name changes to mongo-db, we would have to adjust the url in the application.
- In this case, we will need to rebuild the application and push it to the repository and then we will have to pull the new image in our pod and then restart the server
- So, Kubernetes have a component called Config map.

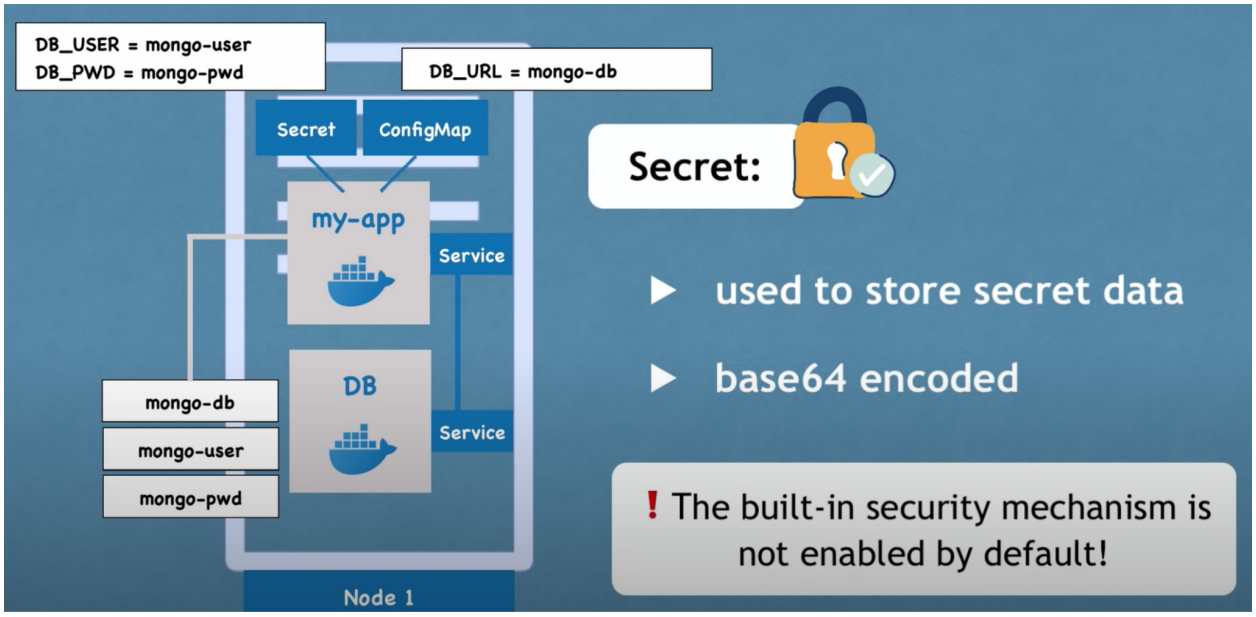
ConfigMap - It is an external configuration for our application

- It is used to store configuration like database url, username, password
- In kubernetes, we can simply connect the configmap to a pod, so that the pod gets the data that config map contains
- Any change in the ConfigMap will immediately reflect in the pod. We will not need to go through the above cycle after any config change.
- Pod of the external configuration can be database username or password which may also change during app deployment process
- Putting password or any type of credentials in ConfigMap is insecure
- So for this purpose kubernetes have another component called Secret



Secret - It is similar to ConfigMap but is used to store secret data like credentials in base64 encoded format

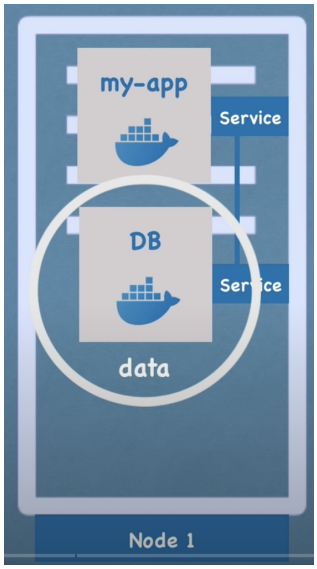
- Database username can be put into ConfigMap but secret data like password, Certificates can be put into Secret
- Just like ConfigMap, we can connect Secret to a pod so that pod can see those data and use it



NOTE: We can use data from ConfigMap and Secret in our application pods using environmental variables or even as properties files

Volumes:

Data Storage in Kubernetes

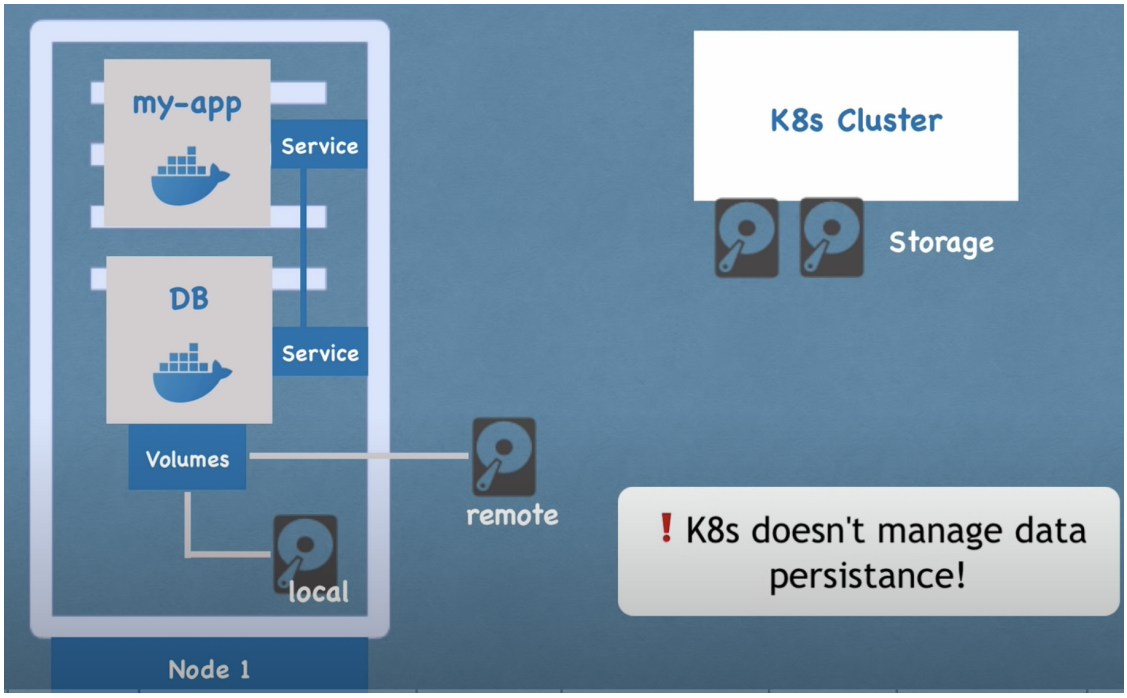


In the above setup, we have a database pod that the application uses and it has some data or it generates some data.

If the database container or the pod gets restarted, the data would be gone

In kubernetes, we have another component called volumes.

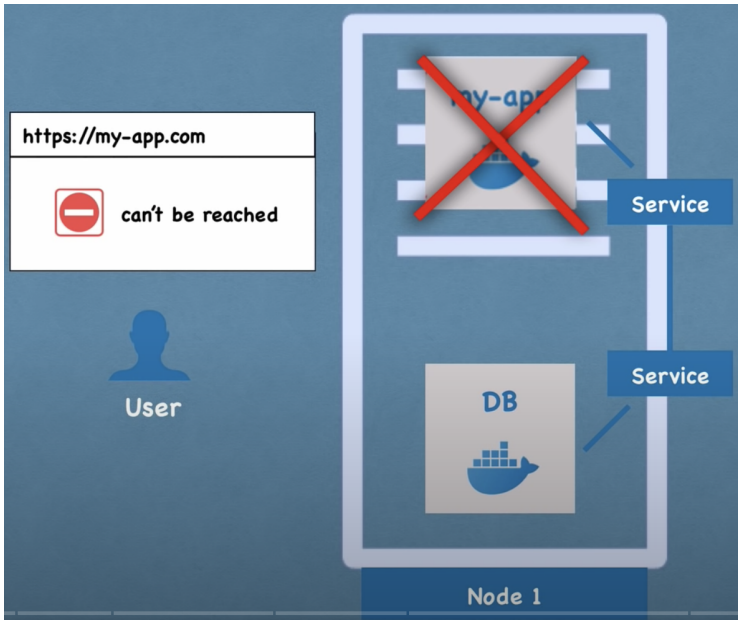
- It basically attaches a physical storage on a hard drive to our pod and that storage could be in our local machine meaning on the same server node where the pod is running or it could be remote storage meaning outside of the kubernetes cluster.
- It could be cloud storage or on premise storage which is not part of the kubernetes cluster
- In this case, even after the database pod or container gets restarted the data will remain persisted



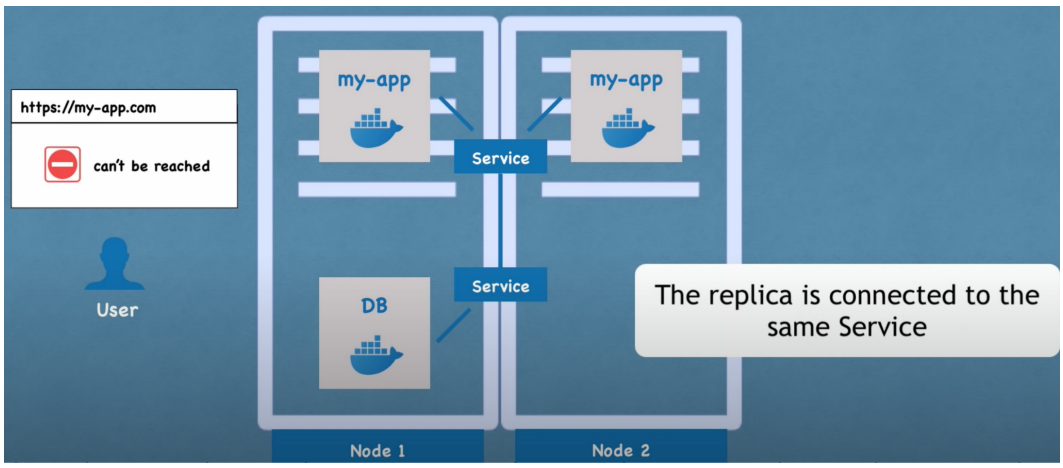
Deployment and Stateful Set

Deployment

- When a application service goes down, it will not be accessible to user
- To make our application more available, we can keep more replica of our application



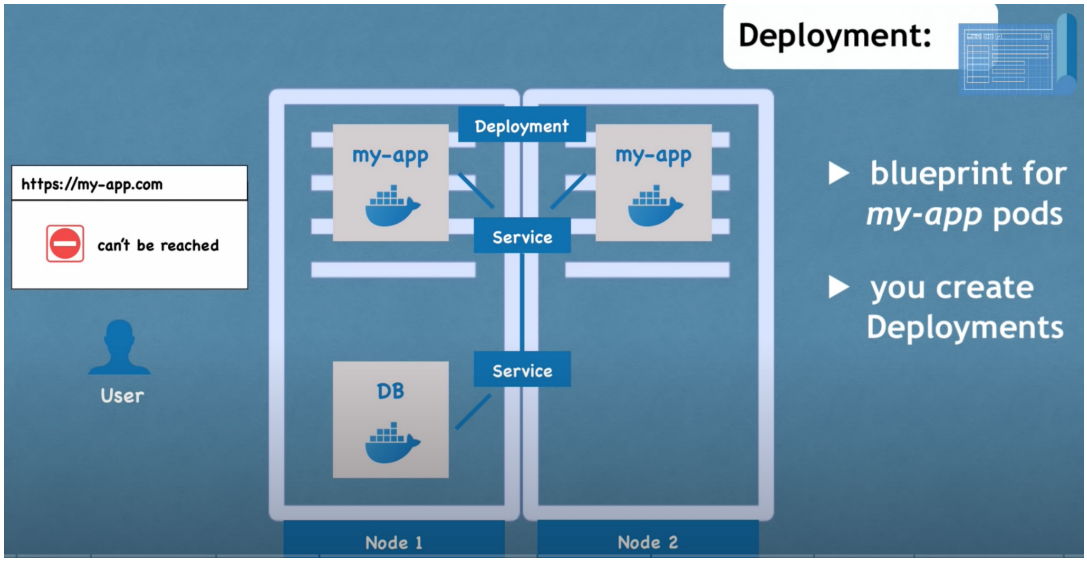
Two node with replicated application server
Note - Service is like a persistent static IP address with a dns name. So that we don't need to adjust endpoint when the pod dies



Service has two functionality:

- Permanent IP
- Load Balancer

- In order to create the 2nd replication of the application pod, we would not create a second pod. But instead we define a blueprint for our application pod and specify the number of replica we want to run
- That blueprint is called Deployment



NOTE:

- Pod is a layer of abstraction on top of containers
- Deployment is another abstraction on top of Pods

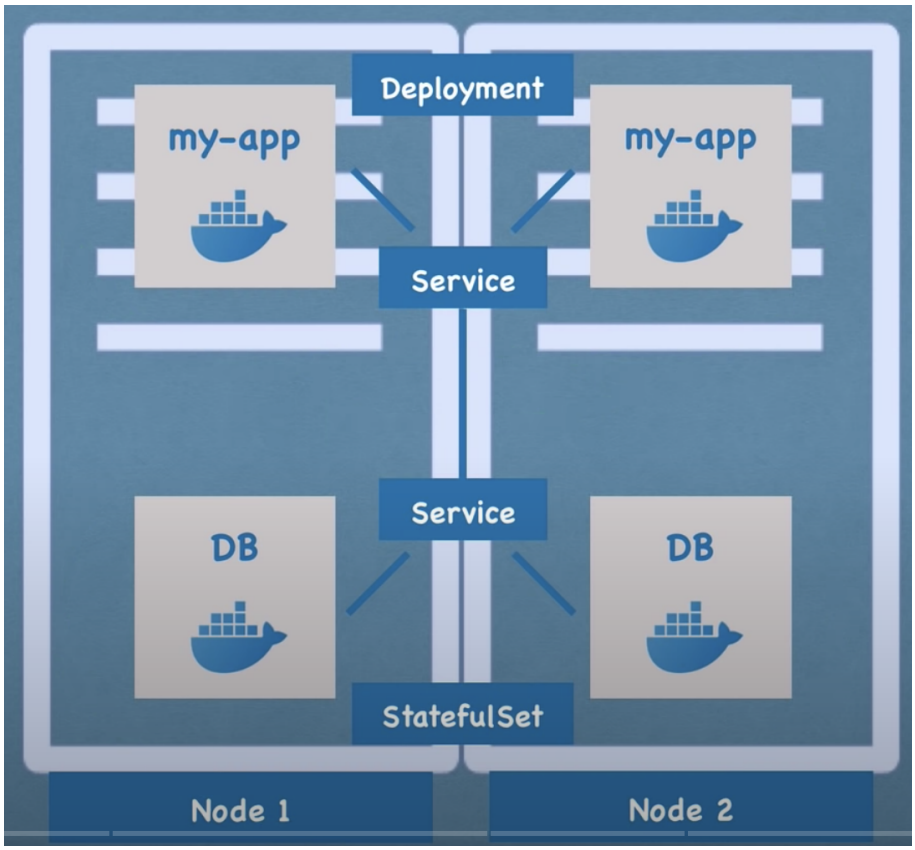
When an application pod dies, the kubernetes will send the traffic to a replicated pod. It will also spin up a new pod based on the replication value in the deployment file.

When a database pod dies, the application will not be accessible. So we need a database replica as well. However, we cannot replicate a database using a deployment. The reason for that is that the database has a state which is its data.
It means, if we have clones or replica of a database, then there would need some kind of mechanism that manages which pods are currently writing to that shared storage which pods are reading from the storage in order to avoid the data inconsistency and that mechanism in addition to replicating feature is offered by another kubernetes component called Stateful Set.

Stateful Set

- It is meant specifically for applications like databases. So mysql, mongodb, elastic search or any other stateful applications or databases should be created using stateful sets and not deployments
- It is similar to Deployment but for stateful services
- Similar to deployment, it will also take care of replicating the pods and scaling them up and down
- It will make sure database reads and writes are synchronized, so that no database inconsistencies are offered
- Deploying database applications using Stateful Set in kubernetes cluster can be somewhat tedious

- So its a good practice to host database applications outside the kubernetes cluster and just have the deployment and stateless applications that replicate and scale with no problem inside the kubernetes cluster and communicate with remote database



Here, we have two replica of my-app and two replica of database and both are load balanced

If one of the node goes down, the second node will completely take over and serve the traffic until the two replica gets recreated

Worker machines in kubernetes cluster

- Running one node with two applications pods running in same node

- Kubernetes cluster can have multiple nodes and each node have multiple application pods with containers running on that node.
- The way kubernetes does this is there must be three process must be installed on every nodes that are used to schedule and manage those pods

Process:

Kubelet

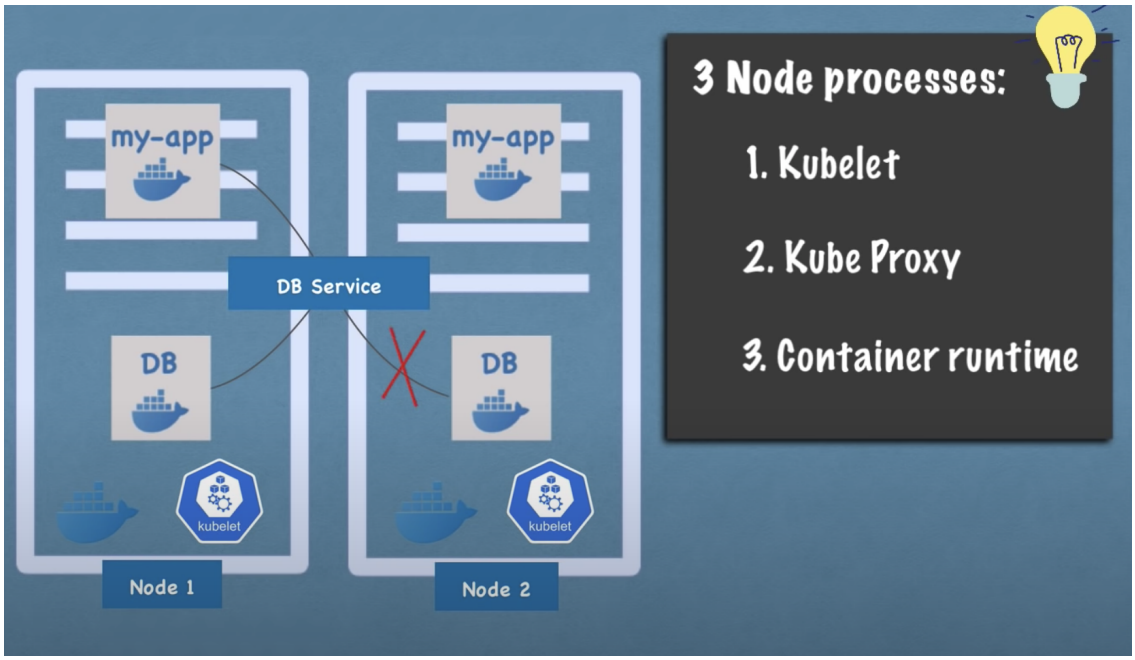
- The first process that needs to run on every node is container runtime. For ex: docker. The process that schedules those pods and containers underneath is **kubelet.**
- Kubelet is a process of kubernetes and it interacts with both container runtime and the machine the node itself
- Kubelet is responsible for taking the configuration and running a pod or starting a pod with a container inside and assigning resources from that node to that container like CPU, RAM and storage resources
- Kubernetes cluster is usually made of multiple nodes which also must have container runtime and kubelet services installed. We can have hundreds of those worker nodes which can run other pods, containers or replicas of the existing pods like myapp or db pod in this example

Services:

- The way communication between pods works is via Services which is some kind of load balancer which catches the request directed to the pod and application like database for ex. And forwards it to the respective pod

Kube proxy

- This is another process which is responsible for forwarding the request from services to pods. It must be installed on every node
- Kube proxy have intelligent forwarding logic inside. It makes sure that the communication works in a performant way with low overhead. For ex: If an application myapp replication is making a request to the database, instead of forwarding request to any replica it will forward in the same node where the app pod initiated the request. This way it avoids the network overhead of sending the request to another machine



How to interact with this Cluser?
Q1. Which node the database pod or application pod should be scheduled for?
Q2. When a pod dies, which process monitors it and reschedules/restart it again

Ans- All of these managing processes are done by Master nodes?

Master Node

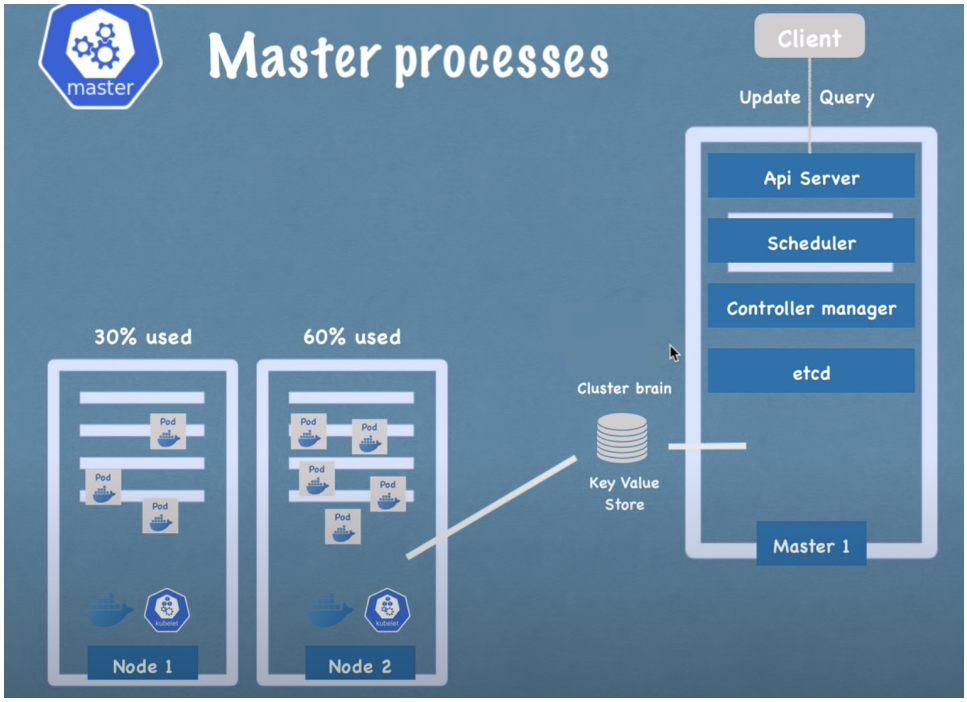
- Master will have completely different set of processes running inside
- These are 4 process running on every master node that controls the cluster state and the worker node as well

Managing services in Master node:

- **API Server**
  - Whenever we want to deploy a new application, we will interact with api server using some client. It could be the ui k8s dashboard, or a CLI like kubelet or k8s API.
  - It is like a cluster gateway which gets the initial request of any updates into the cluster or any queries from the cluster
  - It also acts as a gatekeeper for authentication so that only authenticated users request can get through the cluster
  - It means whenever you want to schedule new pods, deploy new applications, create new service or any other components. You have to talk to the API server on the master node and then the API server will validate your request and if everything is fine then it will forward the request to other processes in order to schedule the pod or create this component that you requested
  - Also, when we query the status of our deployment or the cluster health, we make a request to the API server and it gives us the response
  - Its good for the security as we have only one entry point to the cluster
- **Scheduler**
  - When we send the request to schedule a new pod, the API server after it validates our request will hand it over to the scheduler to start the application pod in one of the worker node. Instead of randomly assigning to any node, scheduler has an intelligent way of deciding on which particular worker node the next pod will be scheduled or the next component will be scheduled
  - First it will check the request on how much resources that the application that we want to schedule needs, how much cpu, how much ram and then it will go through the worker node and see the available resources in each one of them.
  - If it sees that one node is least busy and has the most resources available, it will schedule the new pod in that node.
  - Scheduler just decide on which node the scheduler will be scheduled. The process that actually does the scheduling that actually starts that pod with a container is the kubelet

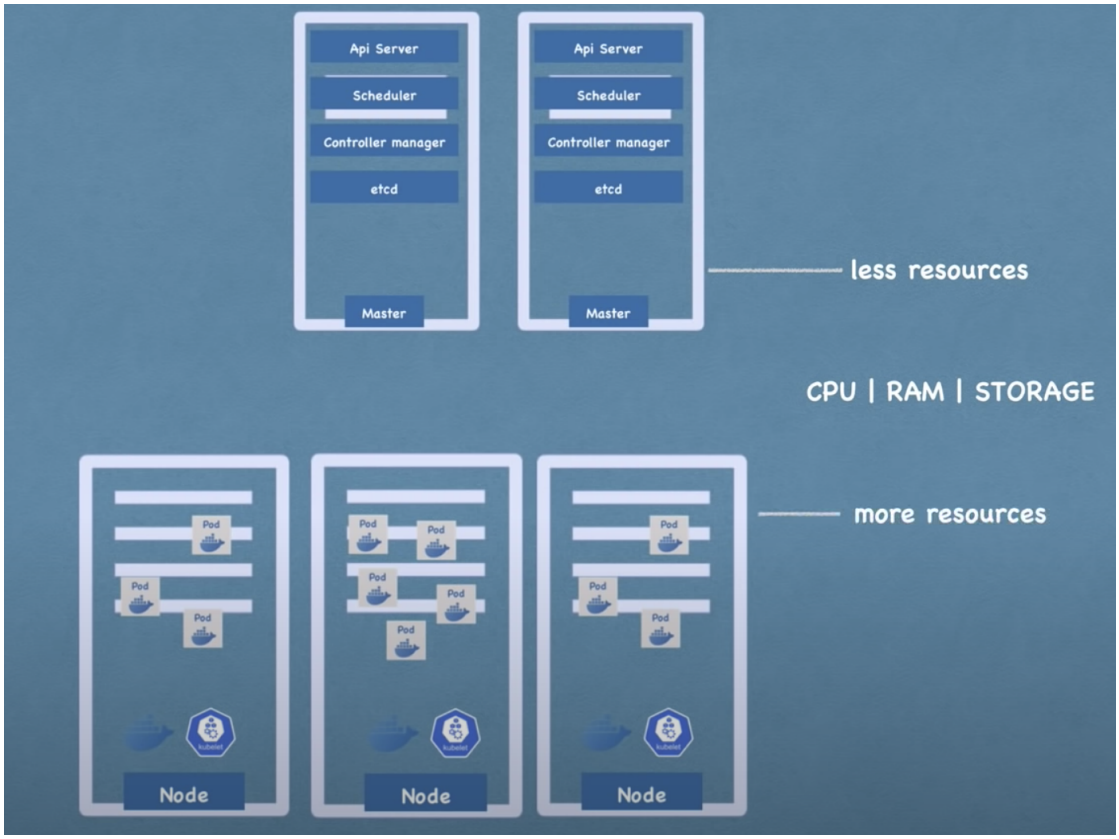
- Kubelet gets the request from the Scheduler and it executes that request on that node

- **Controller Manager**
  - When a pod dies in any node, there must be a way to detect that and reschedules those pods as soon as possible
  - It detects state change like crashing of pods for ex. So whenever that happens, it tries to recover the cluster state as soon as possible.
  - For this, controller manager sends request to scheduler to reschedule those dead pods. The same cycles happens here when the scheduler decides based on the resource calculations that which worker nodes should restart those pods again and makes request to corresponding kubelets on those to restart the pods
- **Etcd**
  - It is a key value store of a cluster state. Its a kind of cluster brain which means every change in the cluster for ex: when a new pod gets scheduled when a pod dies, all of these changes gets saved or updated into the key value store of etcd
  - The reason why etcd store is a cluster brain is because all of this mechanism with scheduler, controller manager etc works because of its data
  - Ex;
    - How does a scheduler know what resources are available on each worker node
    - how does a controller manager knows the cluster state changed in some way for ex: pods died or kubelet started new pod upon the request from scheduler
    - when we make a query request to api server about the cluster health
    - The application deployment state where does the api server gets all these informations.
  - All of the above information will be stored in etcd cluster
  - Application data is not stored in etcd
    - For ex: if a db is running in a cluster, its data will not be stored in the etcd instead it will be stored somewhere else

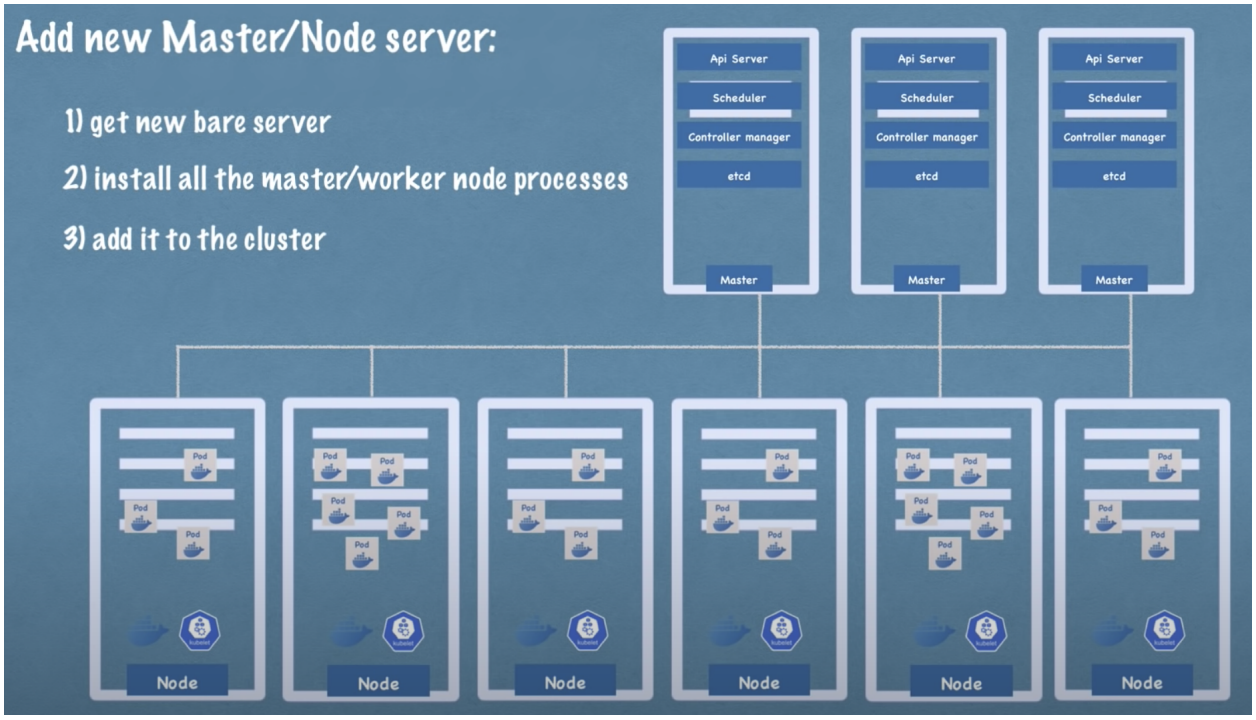


- In practice, k8s cluster is usually made of multiple masters where each master runs its master processes
- Here, Api server will be load balanced and etcd store will be distributed across all master node

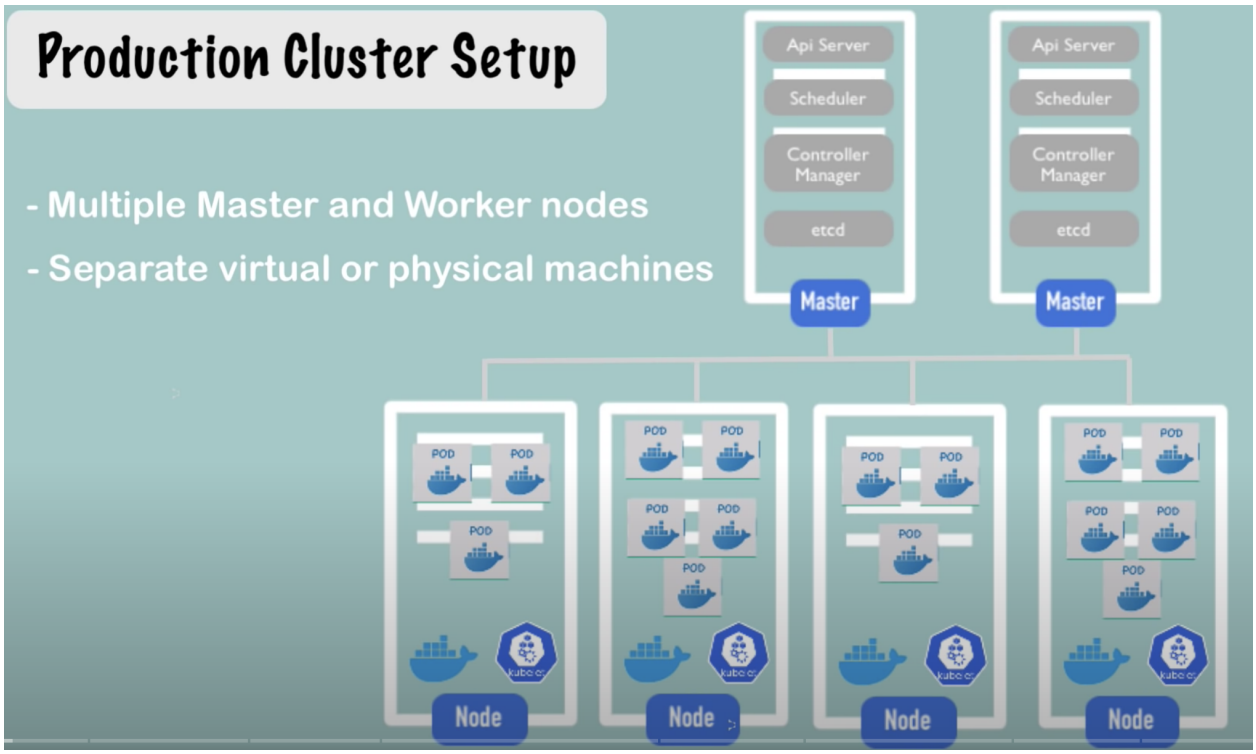
Example Cluster setup:



- Depending on the traffic load, we can increase the number of master and worker node



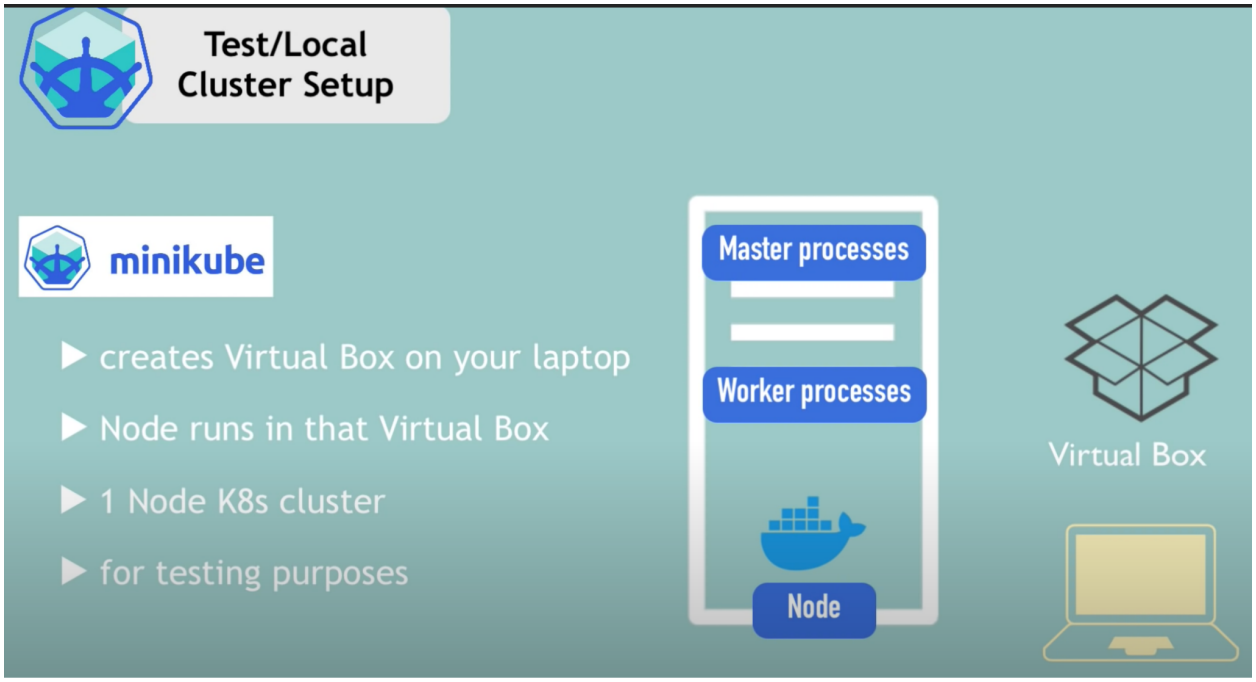
Minikube and Kubectl



Minikube

- Minikube is an Open Source tool that is basically a one node cluster where both master processes and worker process runs on one node
- The node will have docker container runtime preinstalled. So we can run containers/pods with containers in this node
- The way to this in local is through virtual box or some other hypervisor
- Minikube basically creates a virtual box on our laptop
- Node will run in that virtual box

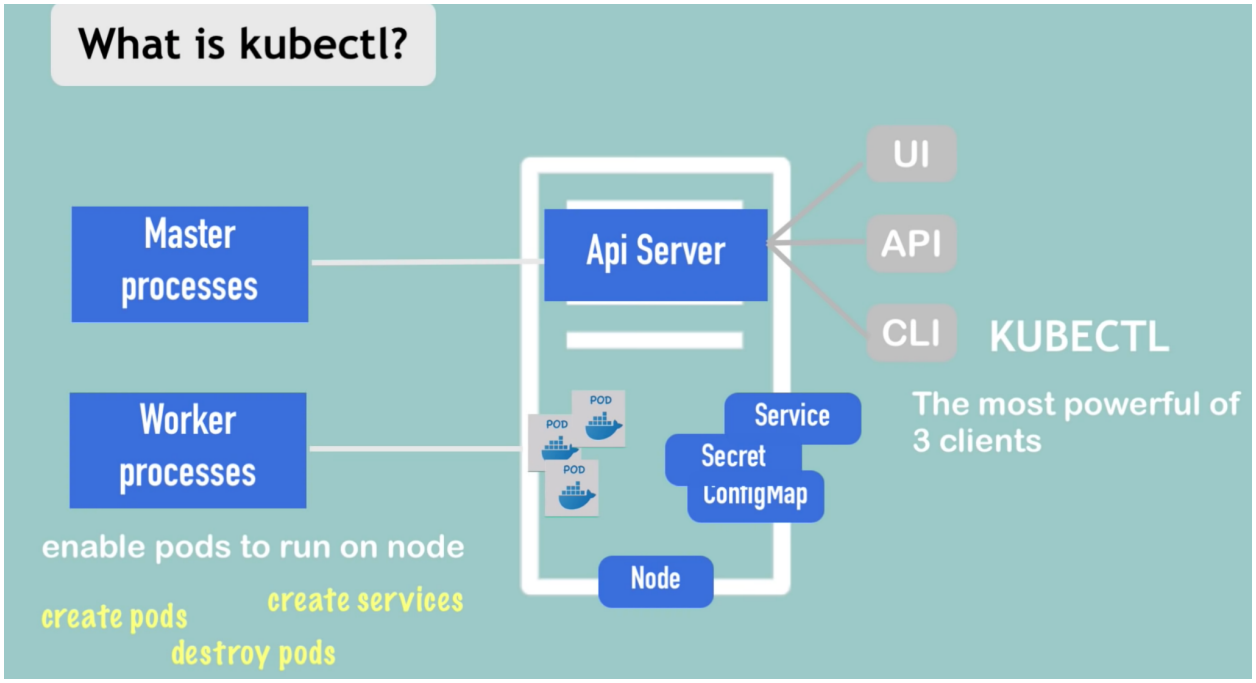
Summary - Minikube is one node kubernetes cluster that runs in a virtual box on our laptop
This can be used for testing in our local setup



To interact with the cluster or to create component/services, K8S provides a tool called kubectl

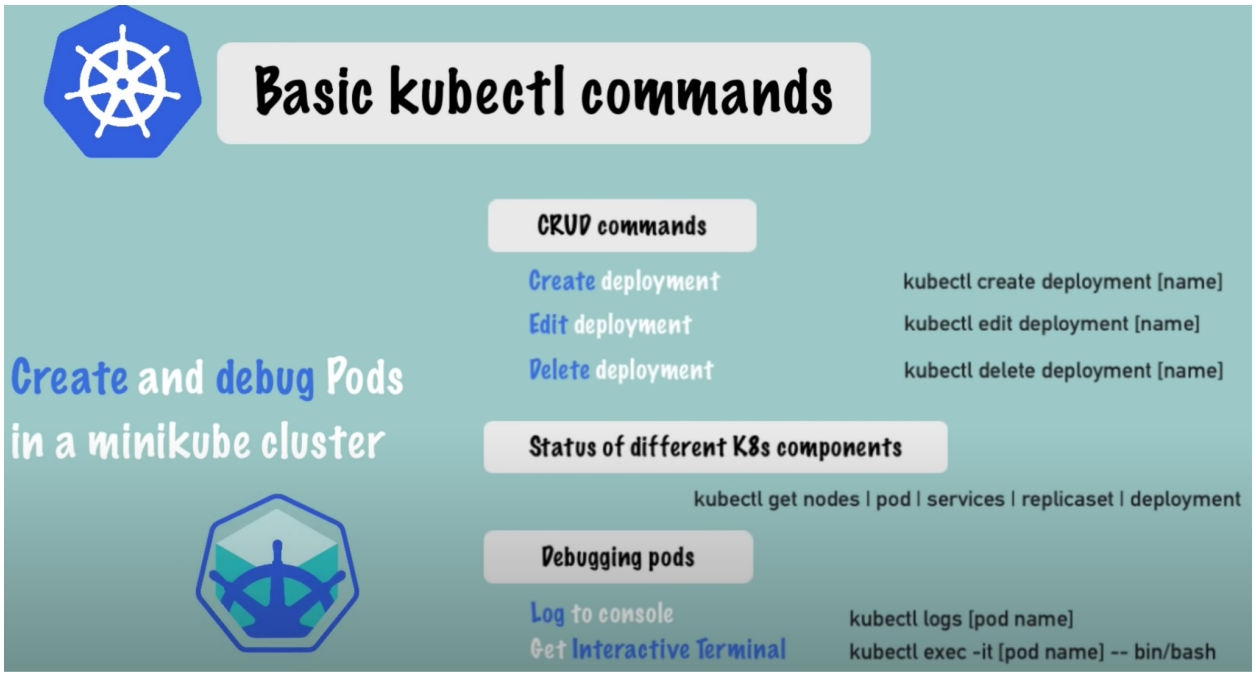
Kubectl

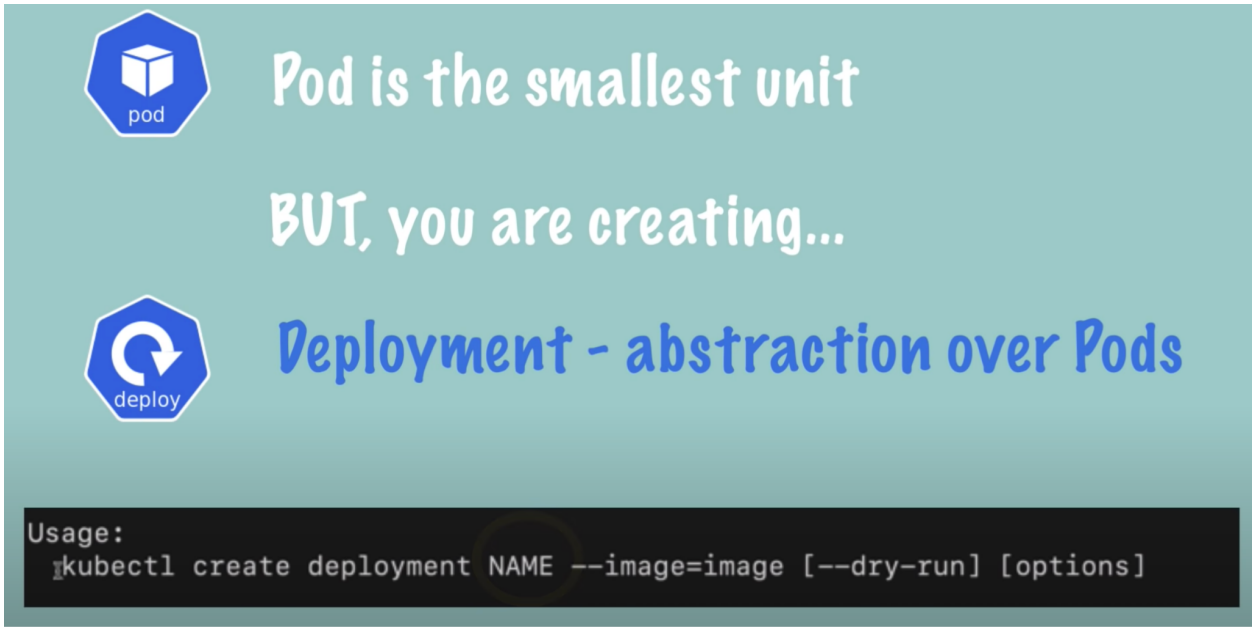
- It is a command line tool for kubernetes cluster
- API server is the entry point for the k8s cluster



Note: Kubectl is not a tool for minikube cluster. It can be used for all type of k8s cluster like remote cluster

Basic Kubectl commands





Layers of Abstraction

Deployment manages a ..

ReplicaSet manages a ..

Pod is an abstraction of ..

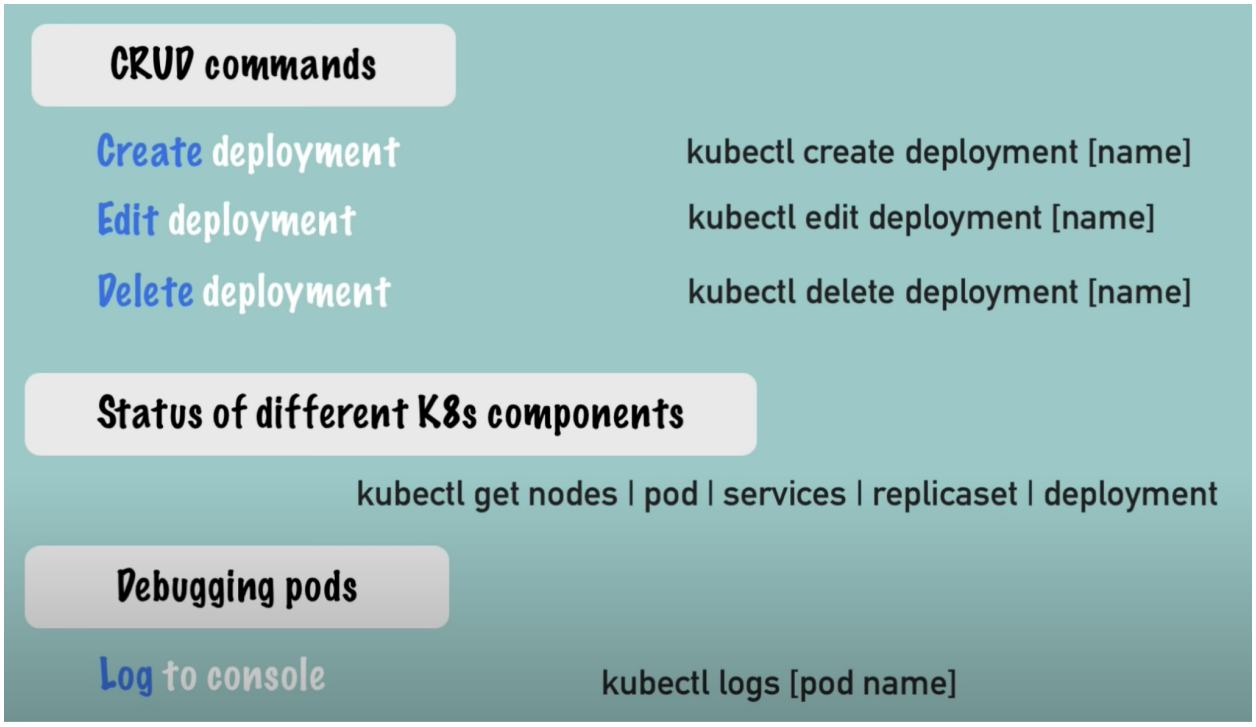
Container

Debugging pods

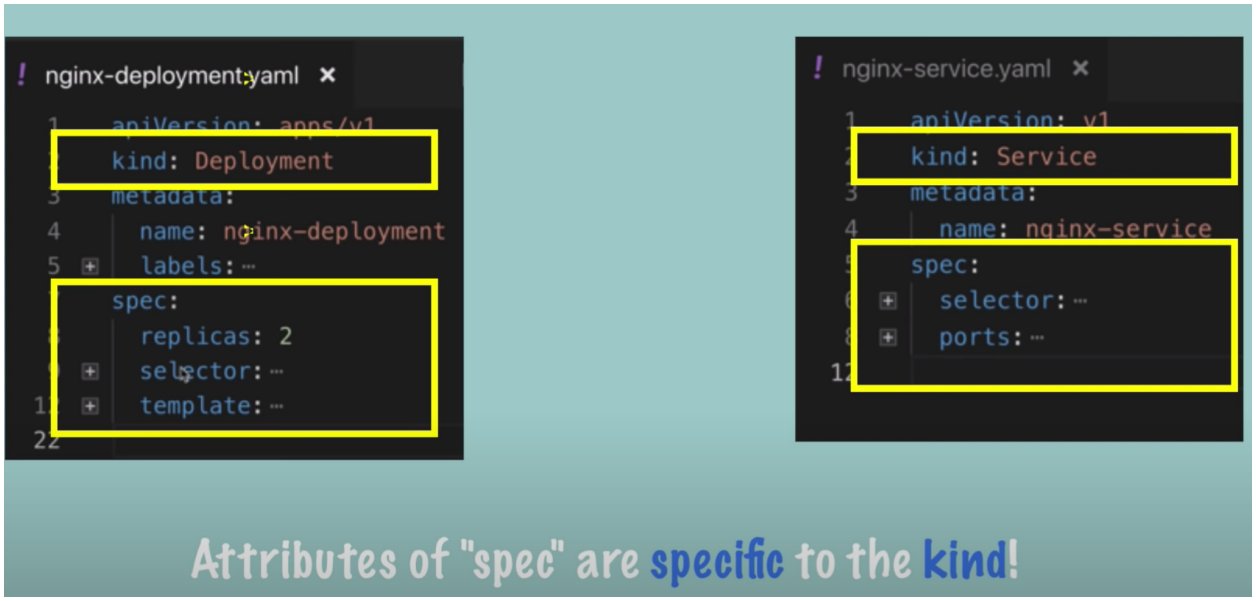
Log to console	kubectl logs [pod name]
Get Interactive Terminal	kubectl exec -it [pod name] -- bin/bash
Get info about pod	kubectl describe pod [pod name]

Use configuration file for CRUD

Apply a configuration file	kubectl apply -f [file name]
Delete with configuration file	kubectl delete -f [file name]



Deployment Types:



There are 3 parts in each configuration file

1. Metadata - It can be added in Deployment file
2. Specification - can be added in Deployment file
3. Status - Automatically generated and added by Kubernetes

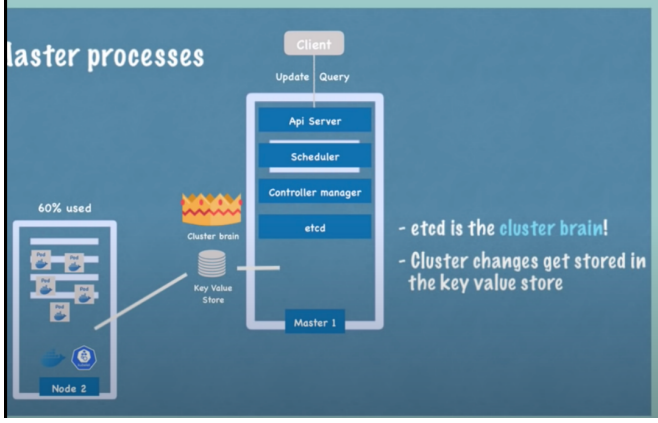
nginx-deployment.yaml

```
1   apiVersion: apps/v1
2   kind: Deployment
3   metadata:
4     name: nginx-deployment
5     labels: ...
    spec:
      replicas: 2
      selector: ...
1     template: ...
22
```

3rd part: status

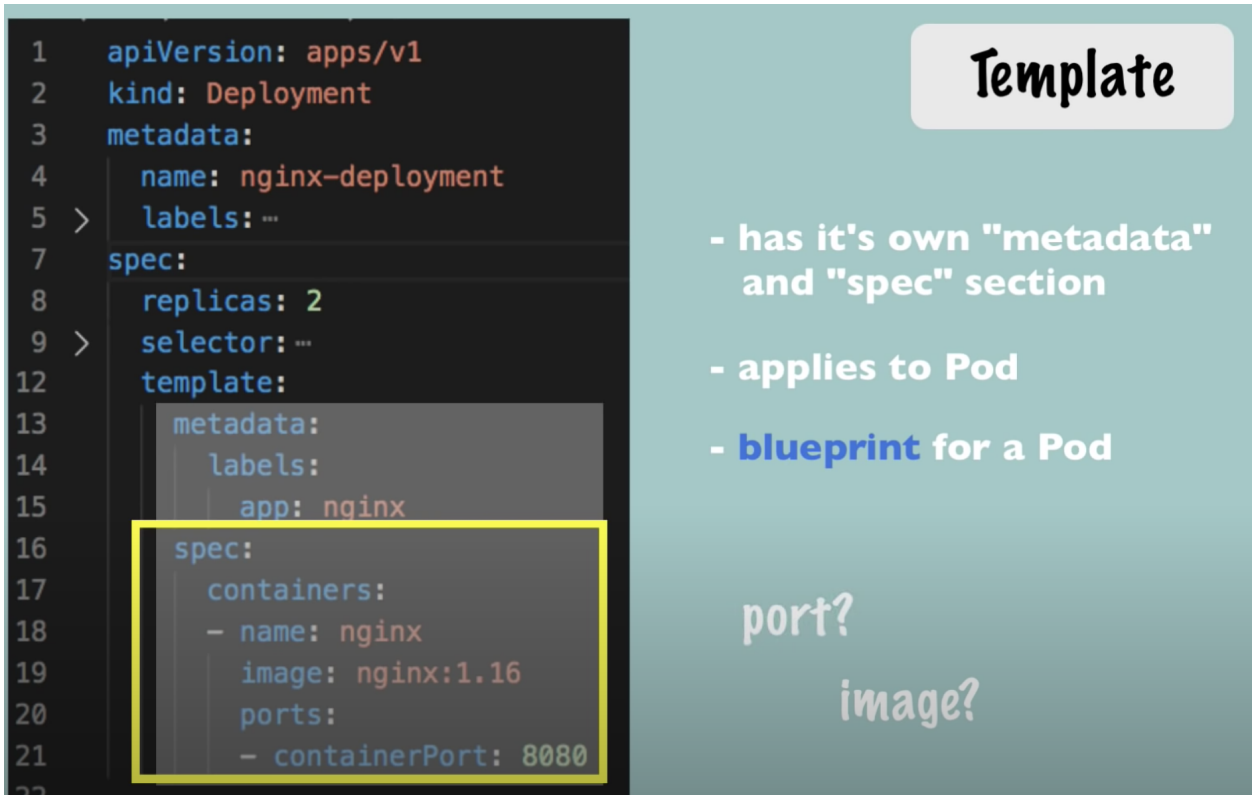
Desired? = Actual?

Where does K8s get this status data?



Etcd holds the current status of any K8s component!

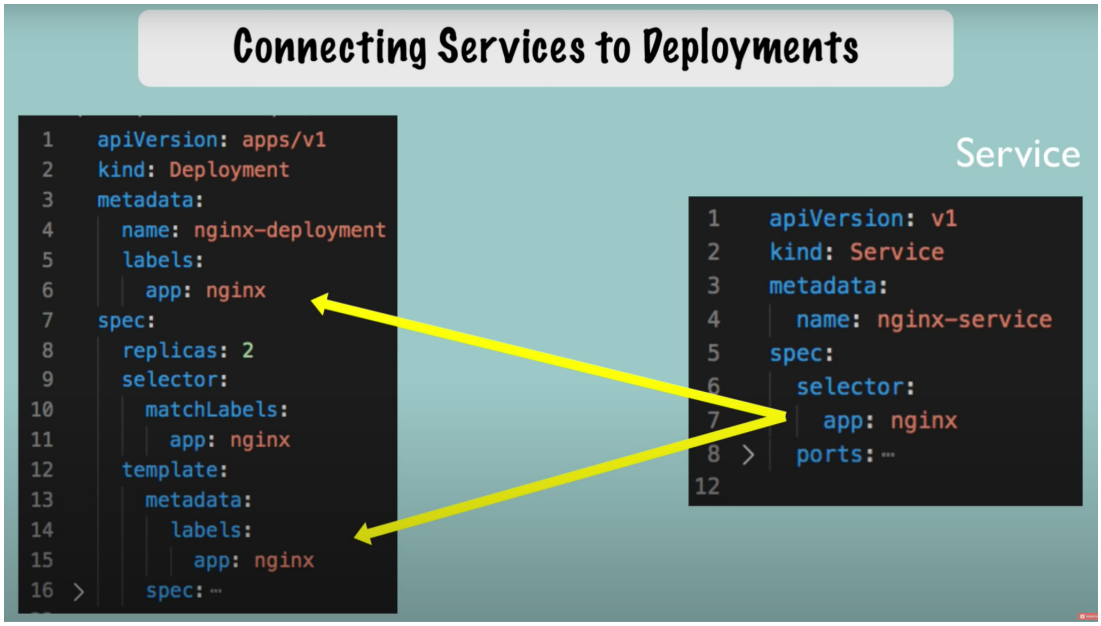
Format of configuration file



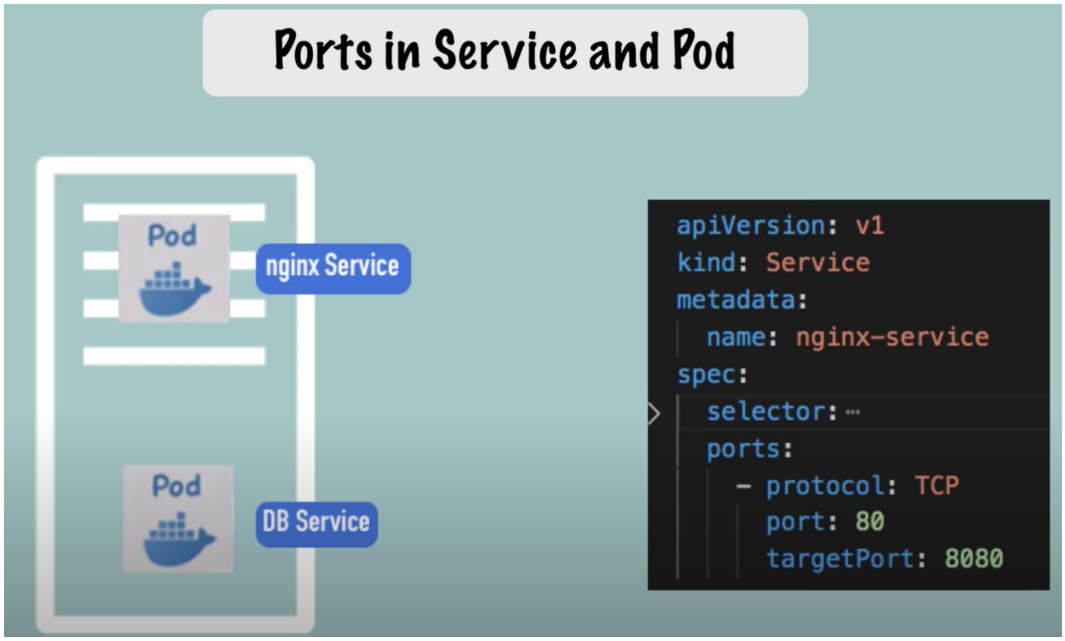
Connecting Components:

- Connecting Deployments to Pods

Labels and Selectors:



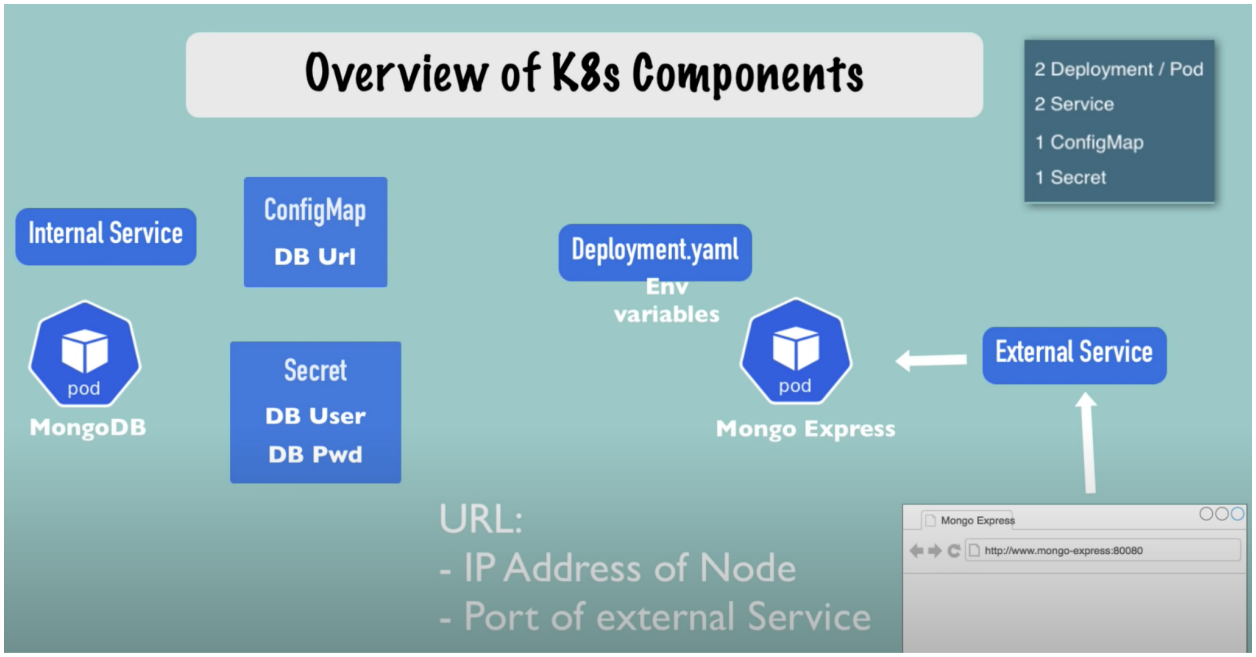
- Metadata contains the labels and the specification part contains the selectors
- In Metadata, we provide components like deployment or pod a key value pair. It could be any key value pair. In this case, it is app: nginx and that label sticks to that component
- Pods gets the label through the template blueprint labeled app nginx and we tell the deployment to connect or to match all the labels with app nginx to create the connection. So this way deployment will know which pod belongs to it
- Deployment has its own label app nginx. These two labels are used by the service selector
- So in service, we define a selector which makes the connection between the service and the deployment or its pods. Service must know which pods are registered with it. Which pods belong to that service. This connection is made through the selector of the label



```
~/Documents — -bash — 89×24
[Documents]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
[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]$ kubectl describe service nginx-service
```

Note: Here Endpoints are the IP and port of the running application pods where the service will forward request
To get detailed pod information:
Kubectl get pod -o wide
To get deployment details:
Kubectl get deployment nginx-deployment -o yaml

Complete Application setup with Kubernetes component:



Creating Secret for Mongo db:

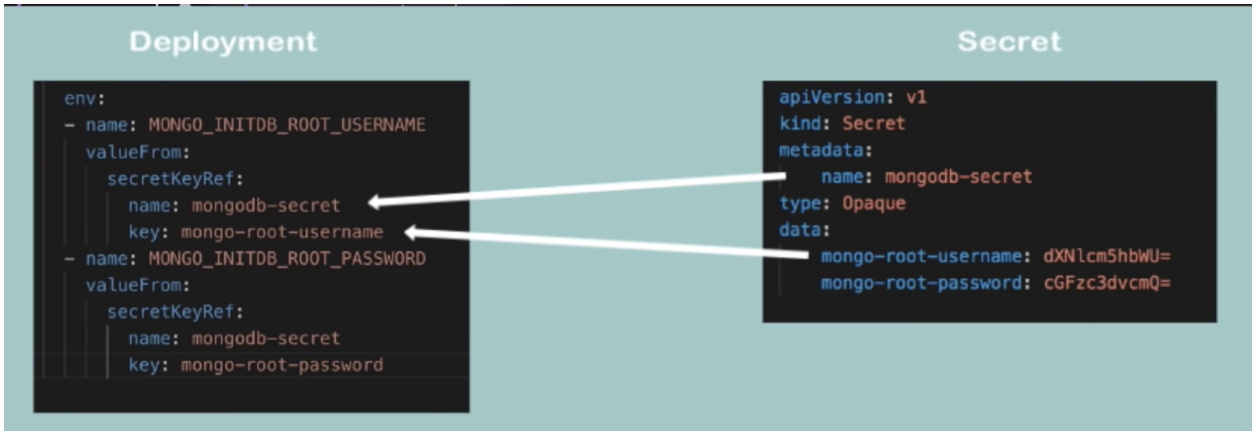
ongo.yaml ! mongo-secret.yaml ×

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

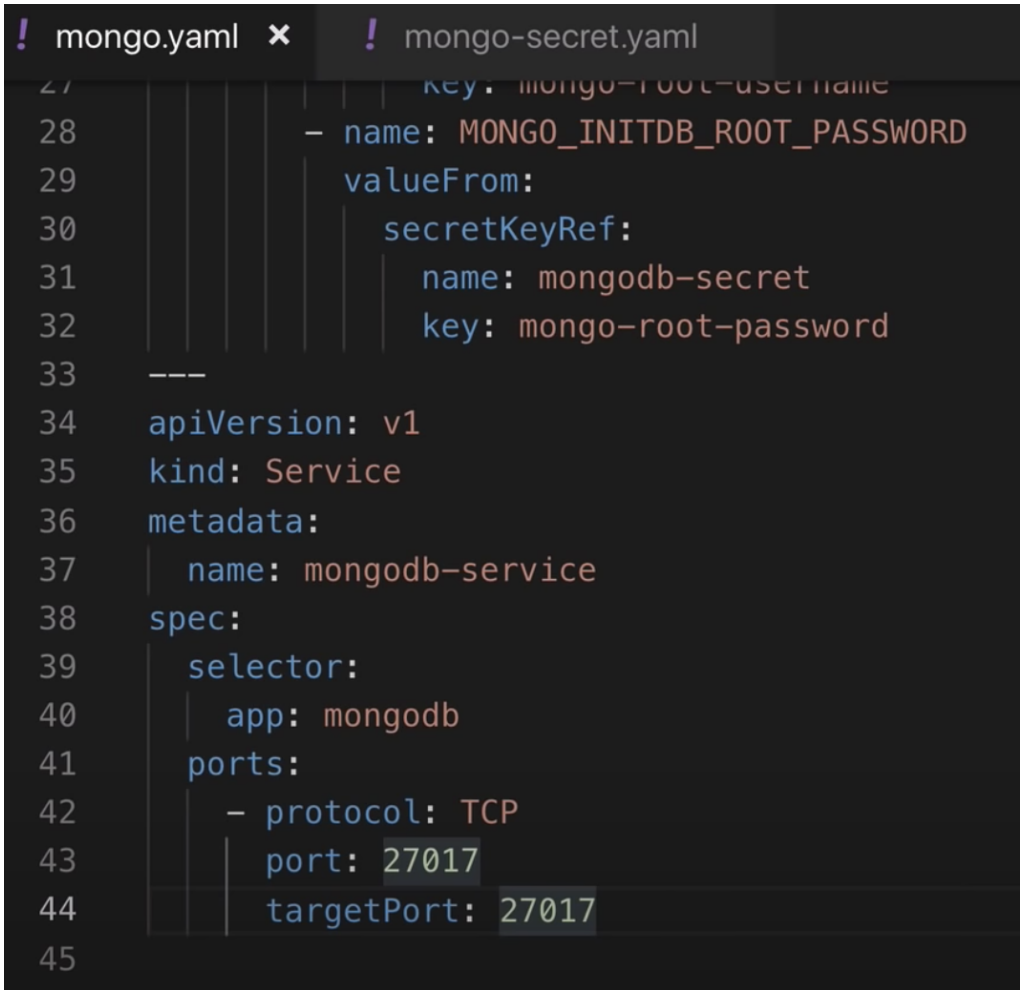
Here, username and password should be base64 encoded

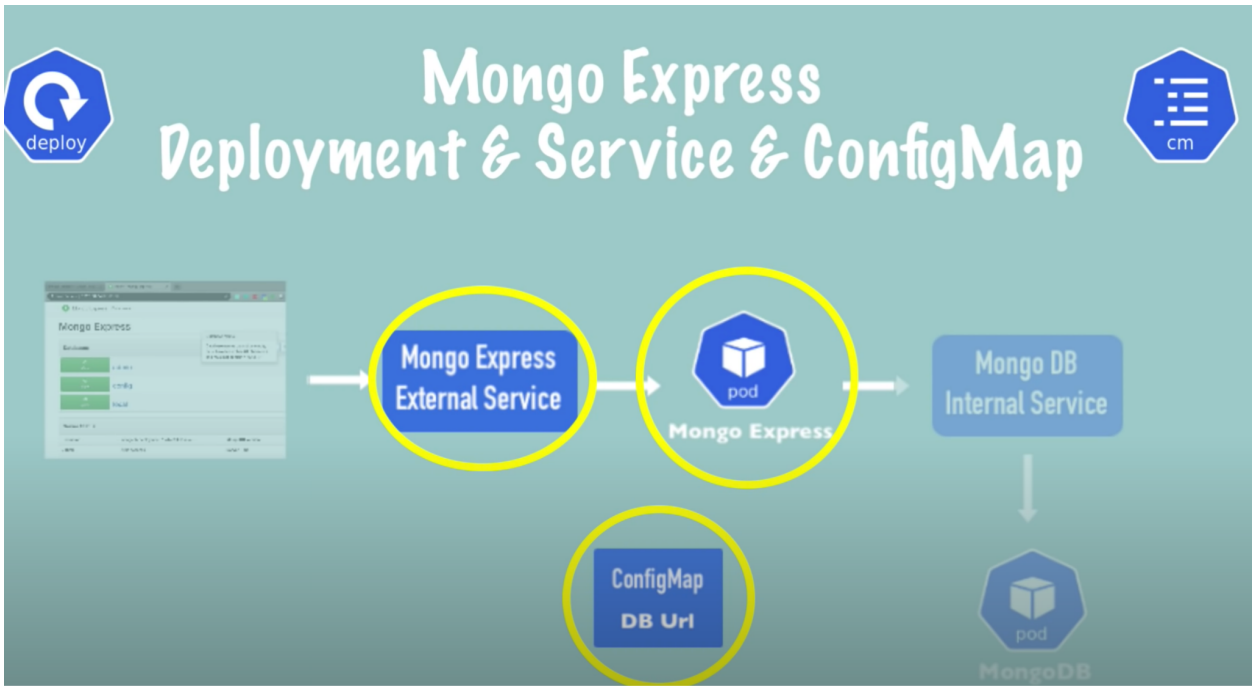
Creating Secret from secret spec file:

```
[k8s-configuration]$ ls
mongo-secret.yaml       mongo.yaml
[k8s-configuration]$ kubectl apply -f mongo-secret.yaml
secret/mongodb-secret created
[k8s-configuration]$
```

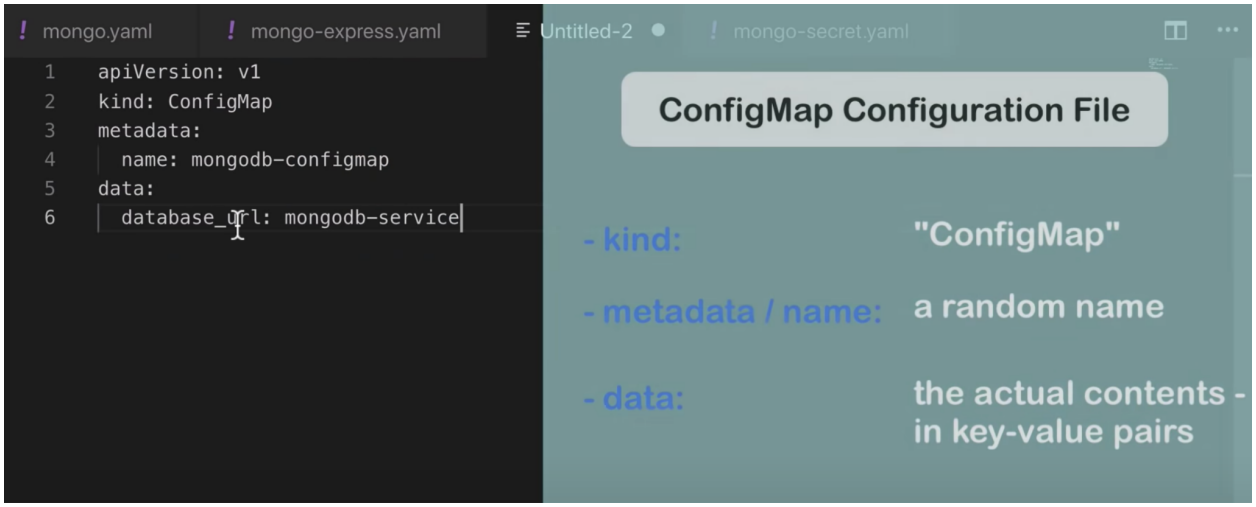


We can add the service in the same file as the deployment.yaml by using 3 hyphens separation:


mongo.yaml
mongo-secret.yaml
key: mongo-root-username
- name: MONGO_INITDB_ROOT_PASSWORD
valueFrom:
secretKeyRef:
name: mongodb-secret
key: mongo-root-password
---
apiVersion: v1
kind: Service
metadata:
name: mongodb-service
spec:
selector:
app: mongodb
ports:
- protocol: TCP
port: 27017
targetPort: 27017

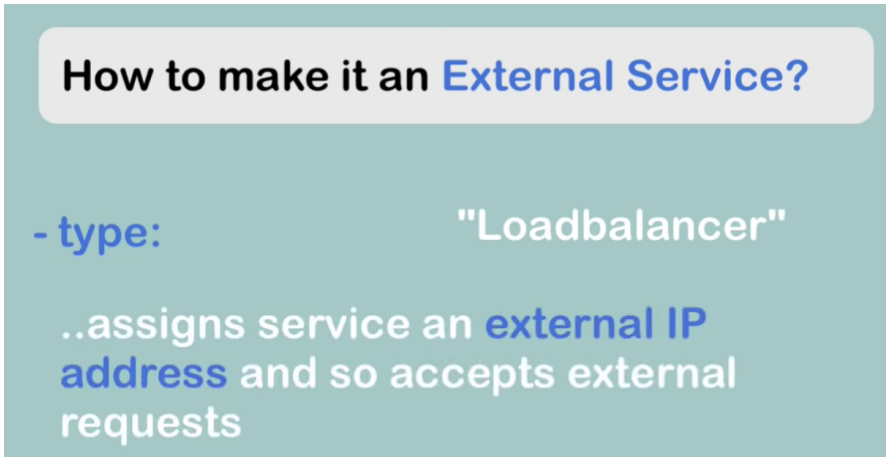


Creating ConfigMap for database URL:



Creating External Service for MongoExpress:

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30000
```

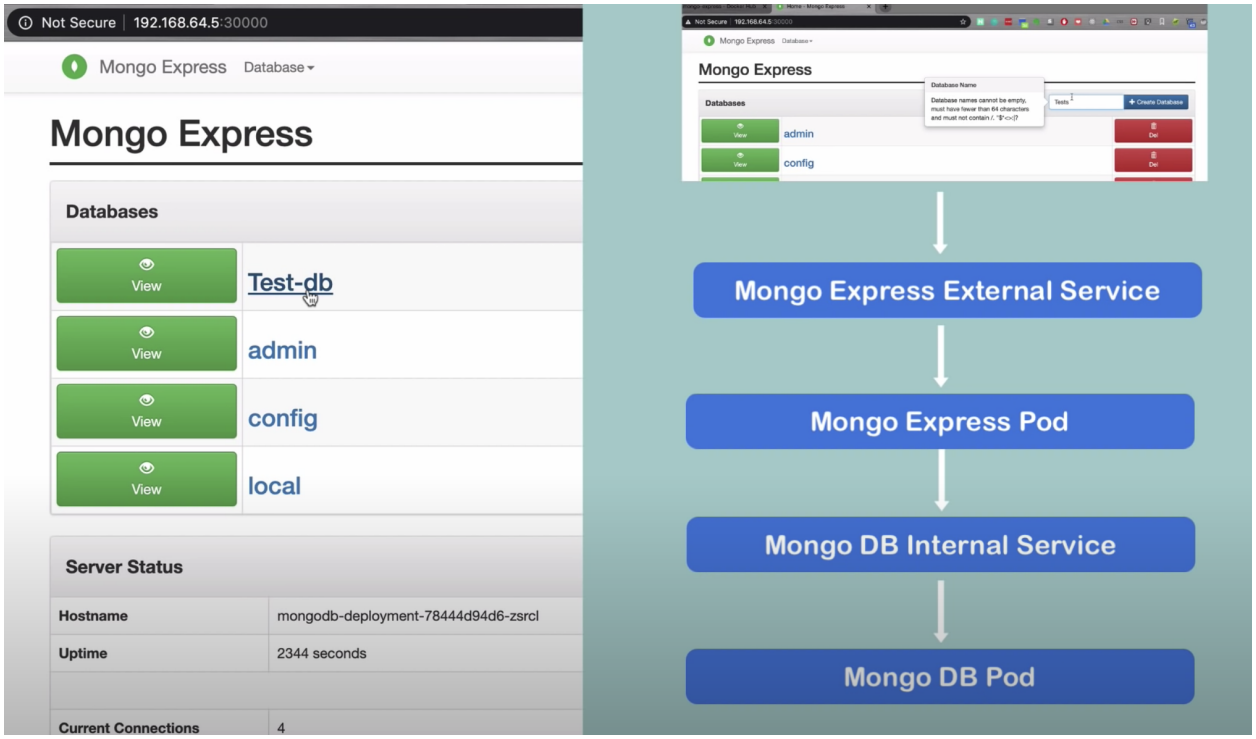



```
[k8s-configuration]$ kubectl apply -f mongo-express.yaml
deployment.apps/mongo-express unchanged
service/mongo-express-service created
[k8s-configuration]$ kubectl get service
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)
      AGE
kubernetes              ClusterIP      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
```

Assigning public IP address for External Service in local

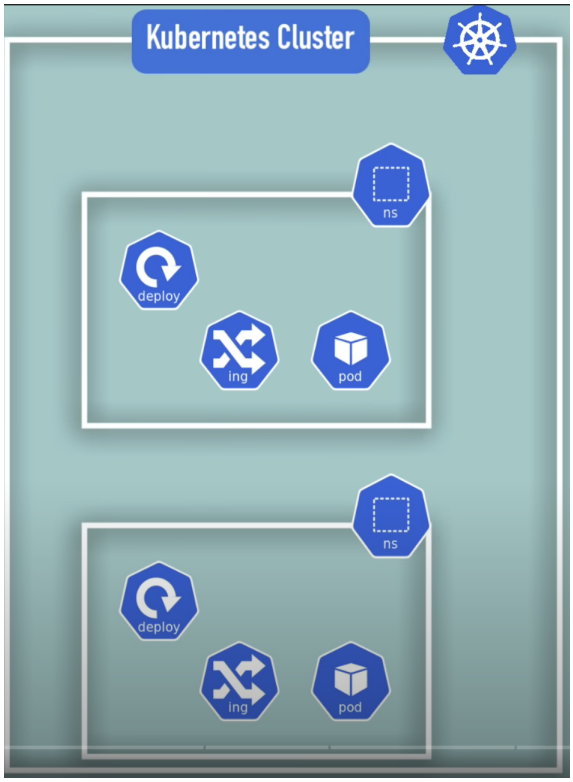
```
[k8s-configuration]$ minikube service mongo-express-service
|-----------|-----------------------|-------------|---------------------------|
| NAMESPACE |         NAME          | TARGET PORT |            URL            |
|-----------|-----------------------|-------------|---------------------------|
| default   | mongo-express-service |             | http://192.168.64.5:30000 |
|-----------|-----------------------|-------------|---------------------------|
  Opening service default/mongo-express-service in default browser...
```

Steps of execution for actions in MongoDB



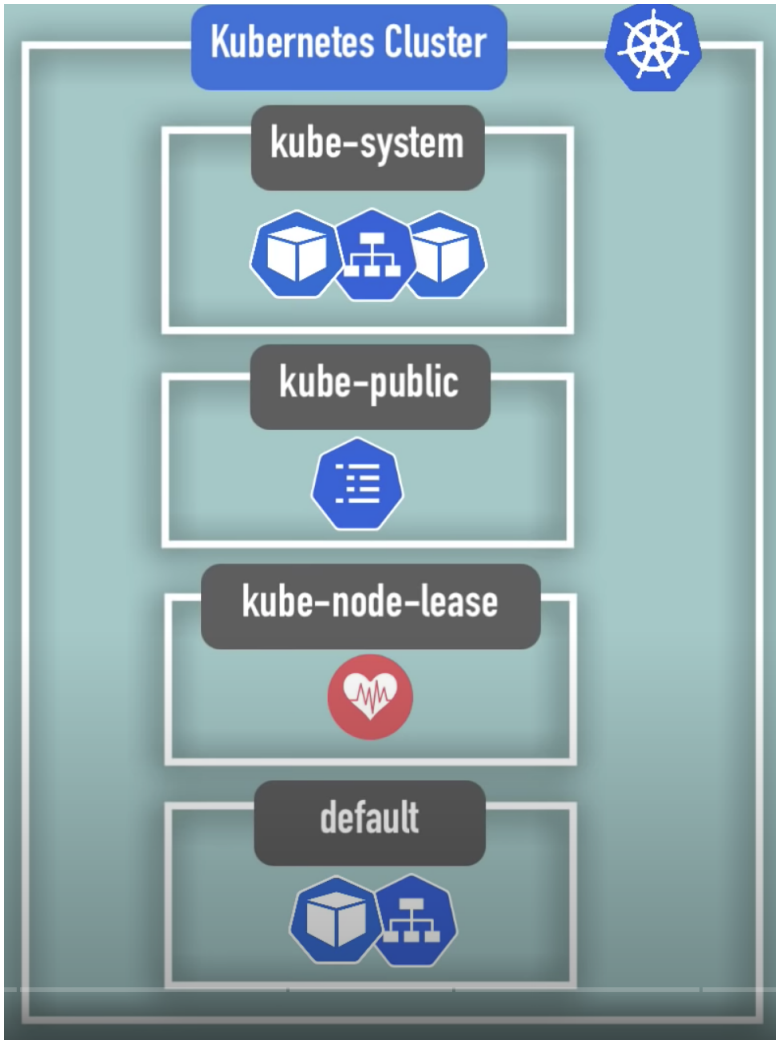
Kubernetes Namespace

- It organises resources in namespace
- Virtual Cluster inside a cluster



Four Namespaces per Default

- Kubernetes-dashboard - specific to minikube cluster (local)
- Kube-system
    - We are not supposed to modify or create in kube-system
    - The components that are deployed in this namespace are the system processes
- Kube-node-lease
    - Hearbeats of nodes
    - Each node has associated lease object in namespace
    - Determines the availability of a node
- Default
    - Newly created re
    - sources are present here unless we mention any new custom namespace

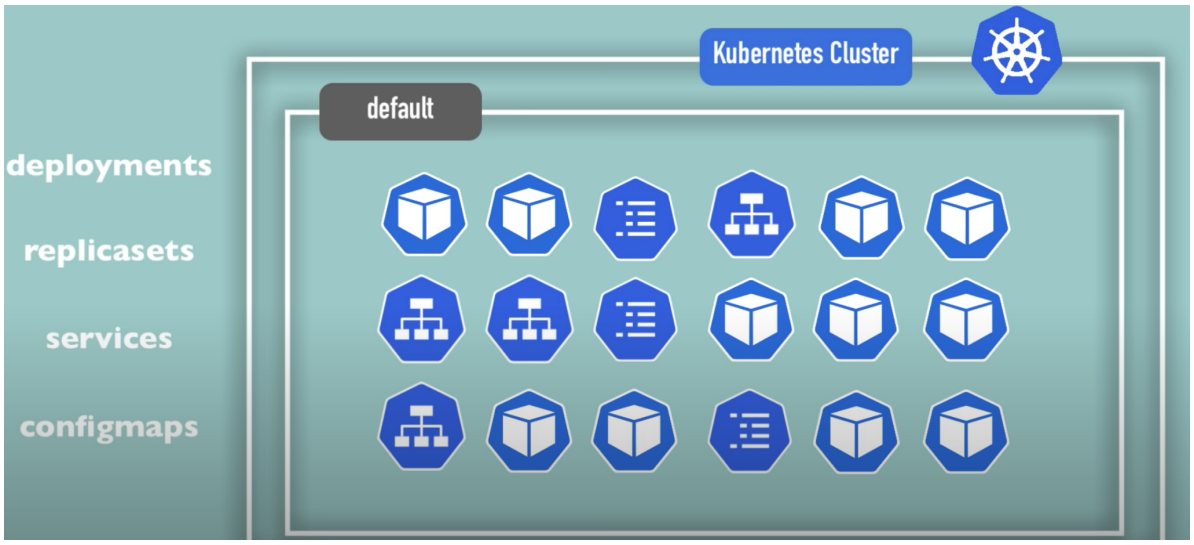


Creating a new namespace

```
[TEST-k8s-configuration]$ kubectl create namespace my-namespace
namespace/my-namespace created
[TEST-k8s-configuration]$ kubectl get namespace
NAME                   STATUS   AGE
default                Active   6d2h
kube-node-lease        Active   6d2h
kube-public            Active   6d2h
kube-system            Active   6d2h
my-namespace           Active   7m41s
```

Why to use a namespace

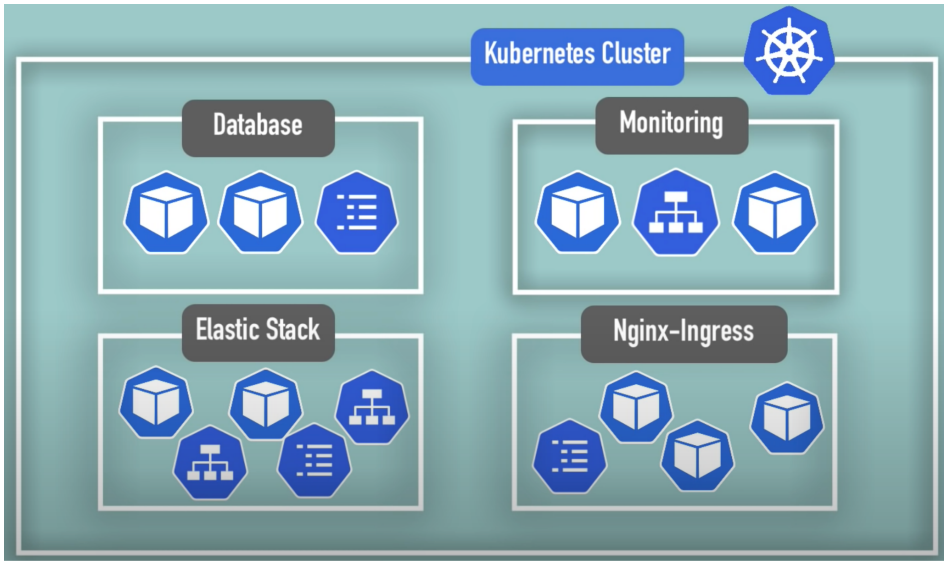
Everything in one namespace



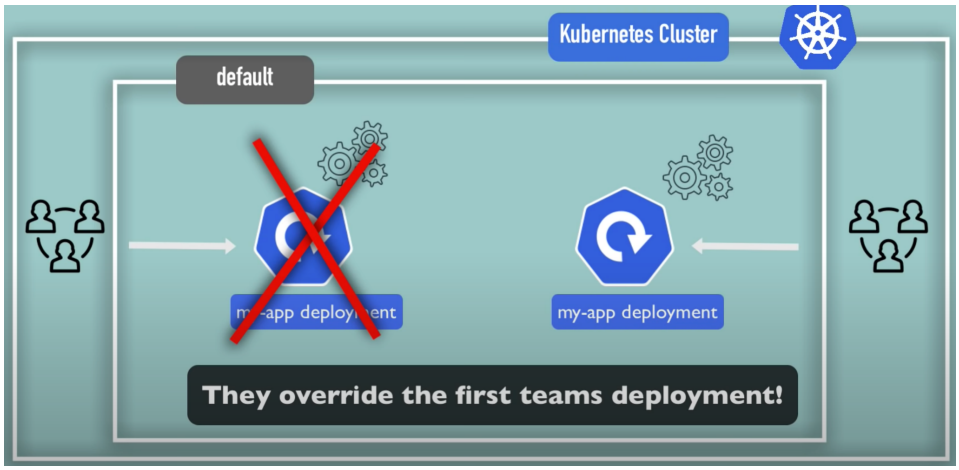
- If we have a complex application, it will become difficult to get the overview different components in a single namespace

Solution:

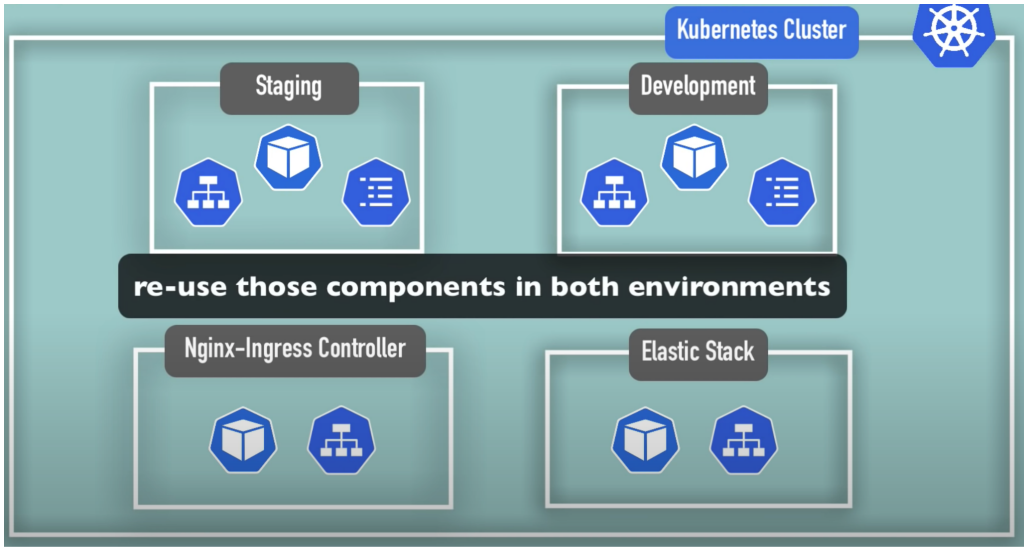
Use Case 1: Grouping resources in namespaces



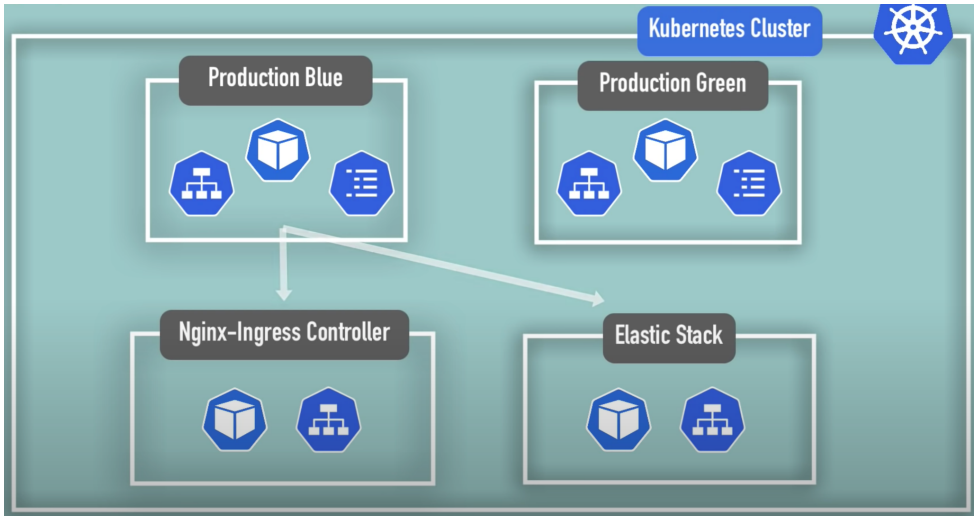
Conflicts: Many teams, same application
Single namespace:



Resource Sharing :
Use case 2: Staging and Development

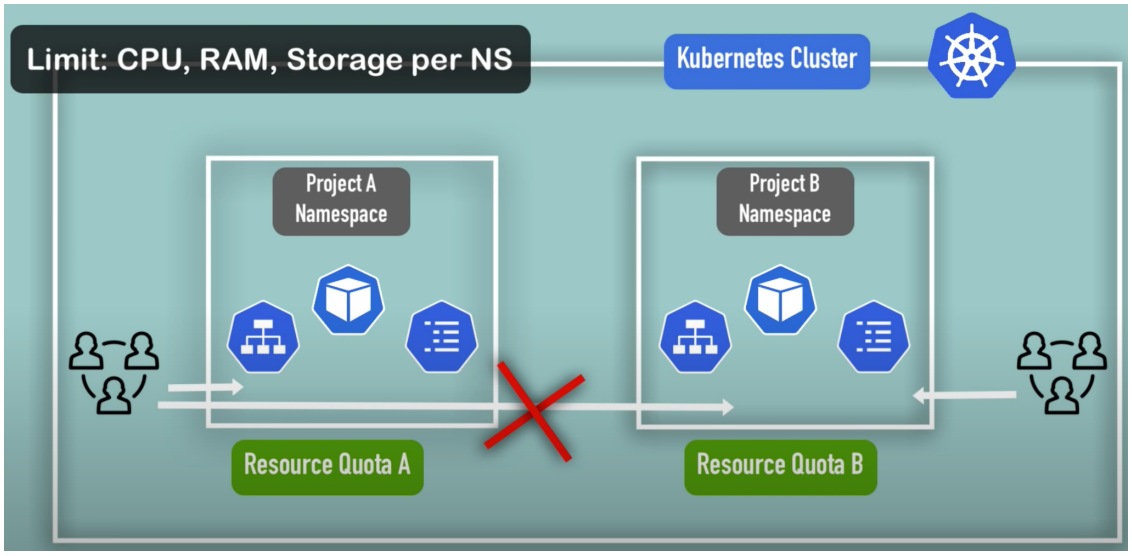


Use case 3: Blue/Green Deployment

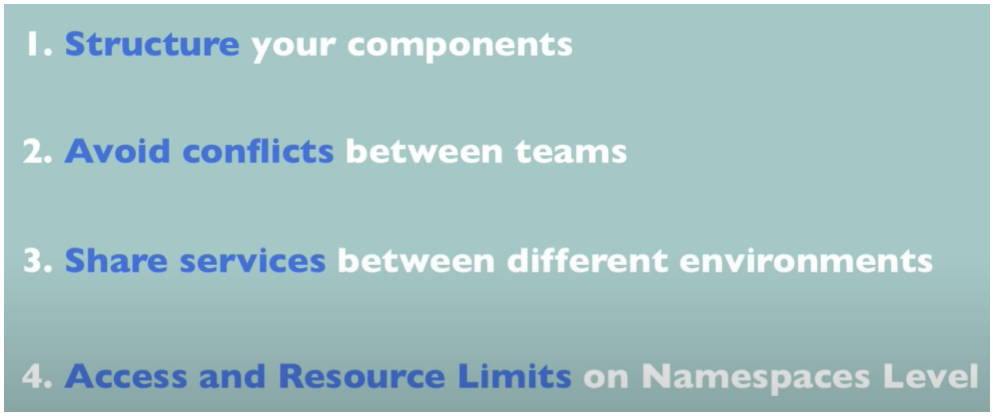


Use Case 4: Access and Resource limit on Namespaces

- We can give different team access to only their namespaces
- It will avoid one team accidentally interfering another teams work
- Each team can have own secure isolated environment
- We can limit the resources(CPU, RAM, Storage) per namespace

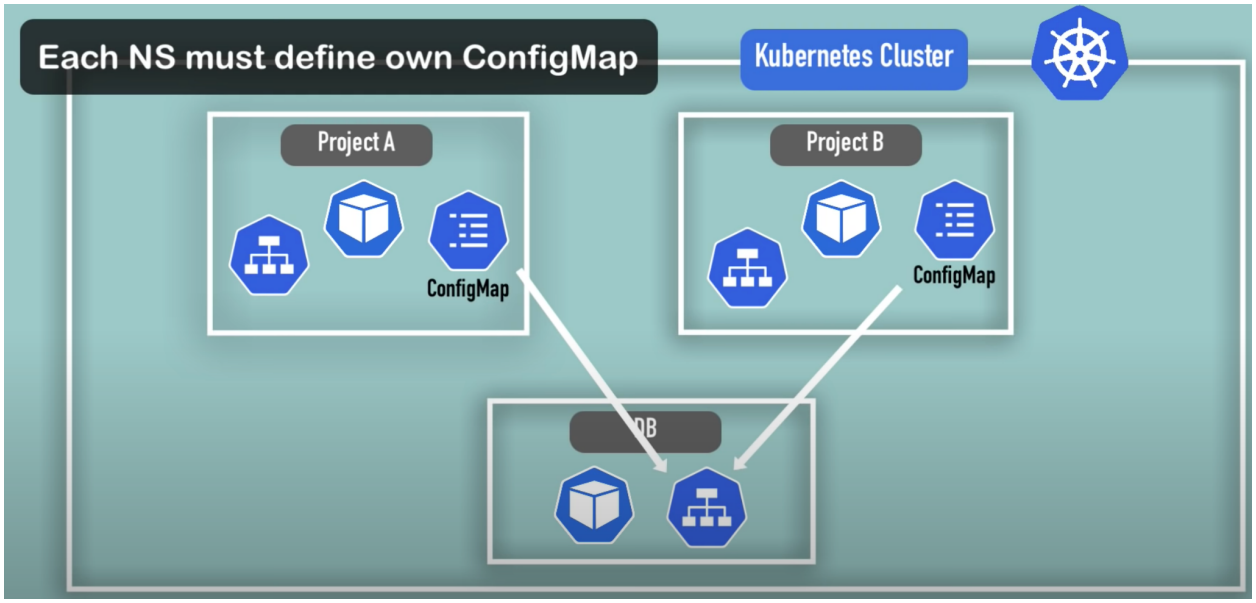


Use Cases when to use Namespaces



Constraints in namespace:

- Each namespace must define own ConfigMap
- ConfigMap in one namespace cannot be accessed by another namespace even if they refer to common resources



- Same applies for Secret as well. Cannot be shared across namespaces.

Resources that can be shared across namespace

- Services can be shared across namespaces.
- Services like nginx, elastic search can be used in different namespaces

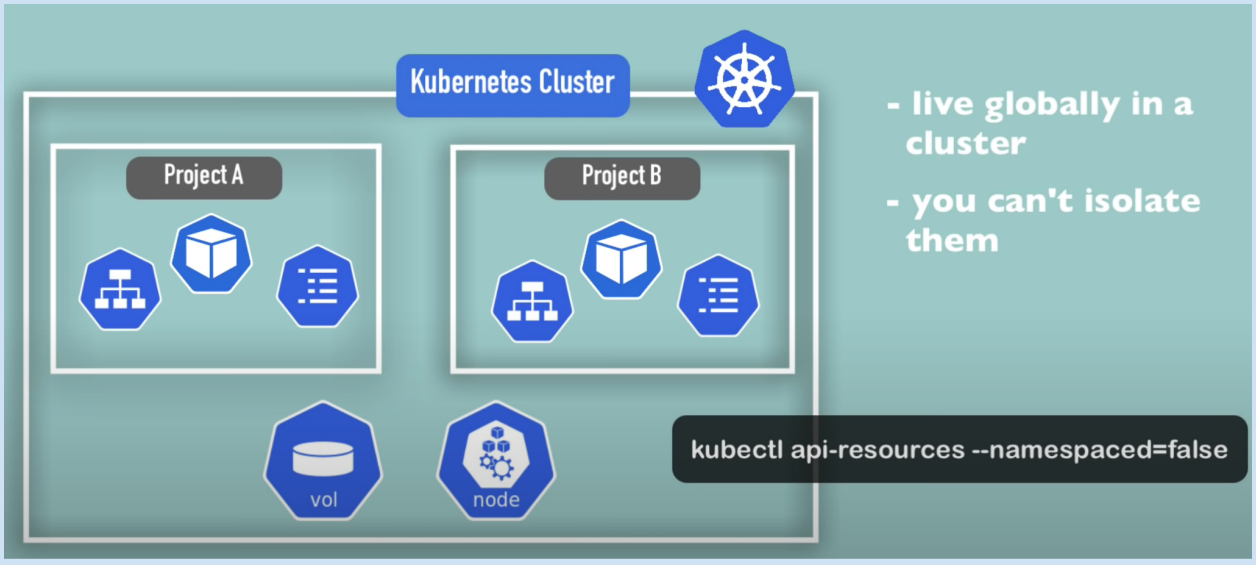
Components which can't be created within a namespace:

- Volume
- Node

Command to list down resources that are not bound to namespace:

Kubectl api-resources --namespaced=false

Command to list down resources that are bound to namespace:
Kubectl api-resources --namespaced=false



Creating a component in a Namespace:

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

Inside Config:

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

External Service and Ingress

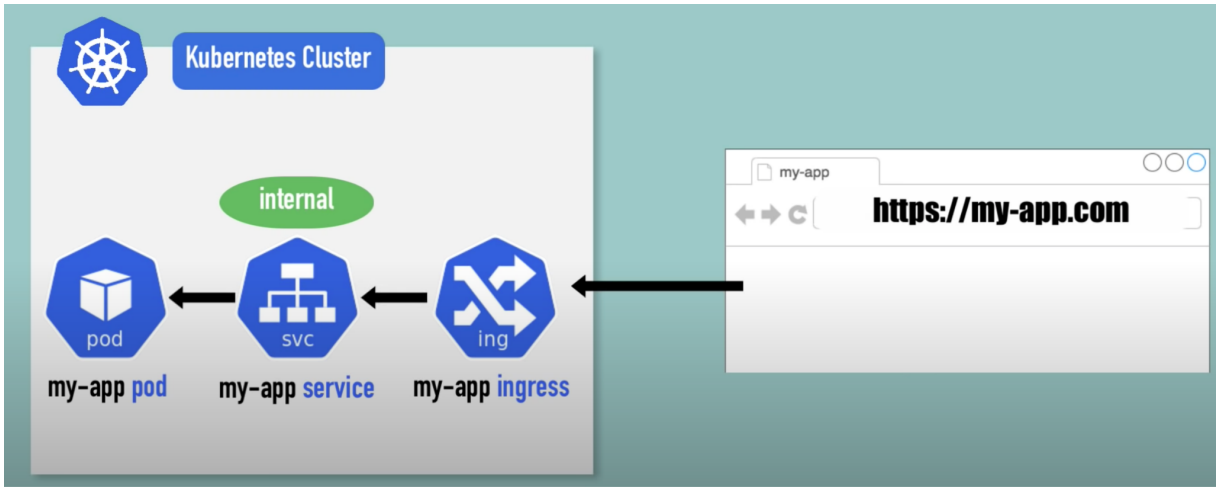
Two ways to serve public request via k8s services:

1. External Service
2. Ingress

Ingress:
We can access the external service from public internet via:
http://{node_ip}:{external_service_port}

But it should be:
https://my-app.com

The way to do this is via the Kubernetes component called Ingress.



Example YAML file: External service

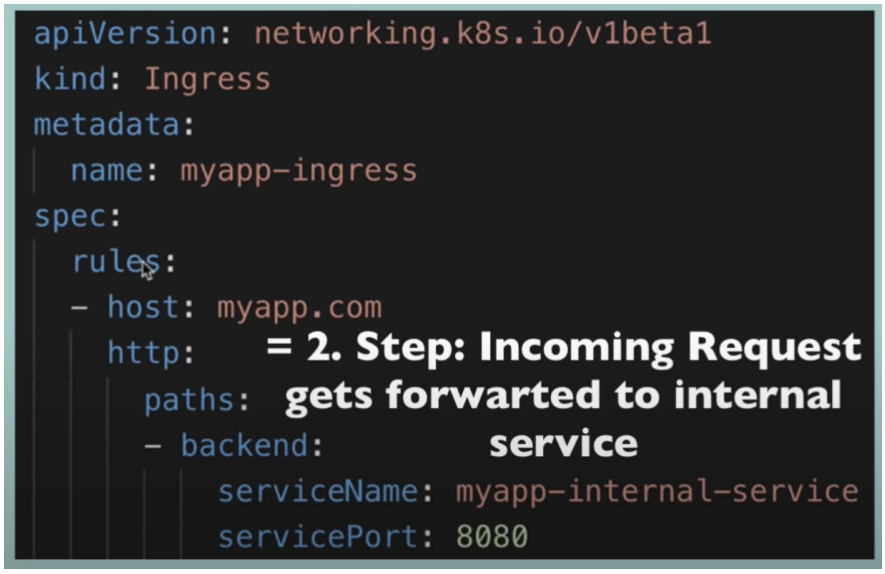
```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-external-service
spec:
  selector:
    app: myapp
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 35010
```

External YAML: Ingress

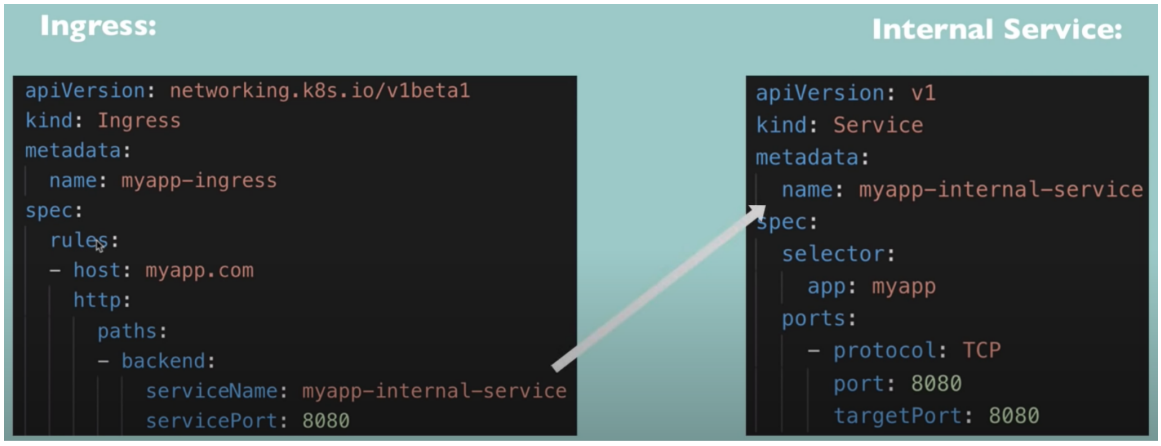
Routing Rules:
Forward request to the internal service

```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
```

- All the request reaching the above mentioned host will forward it to provided backend Service with given name

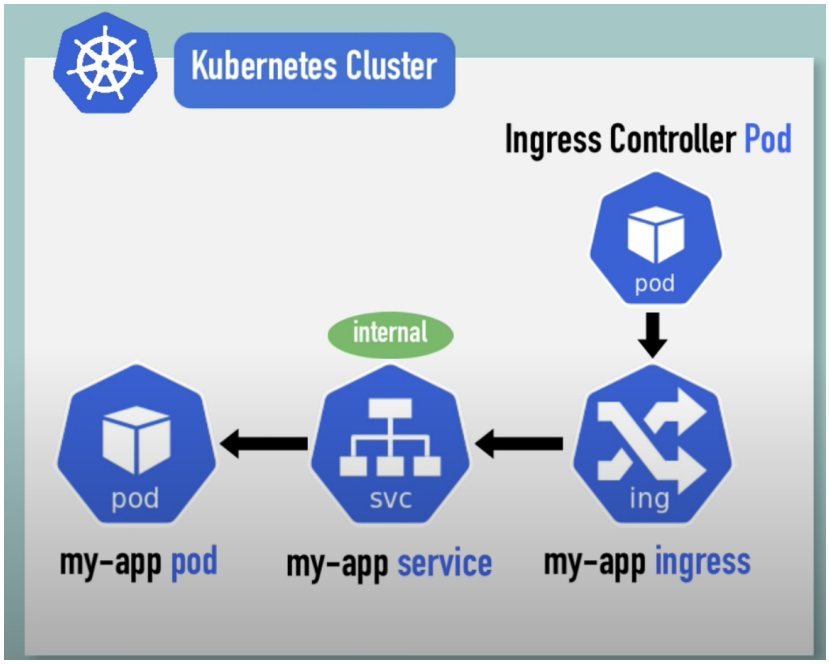


Ingress and Internal Service configuration:



Configuring the Ingress in the K8s Cluster

Ingress doesn't work in itself, it needs the implementation i.e Ingress controller to process the ingress rules:



What is an Ingress Controller?

- The function of the ingress controller is to evaluate all the rules defined in the cluster
- manage all the redirections
- Entrypoint to the cluster for all the request to the given domain or subdomain rules
- It can evaluates all the different ingress files with different ingress rules

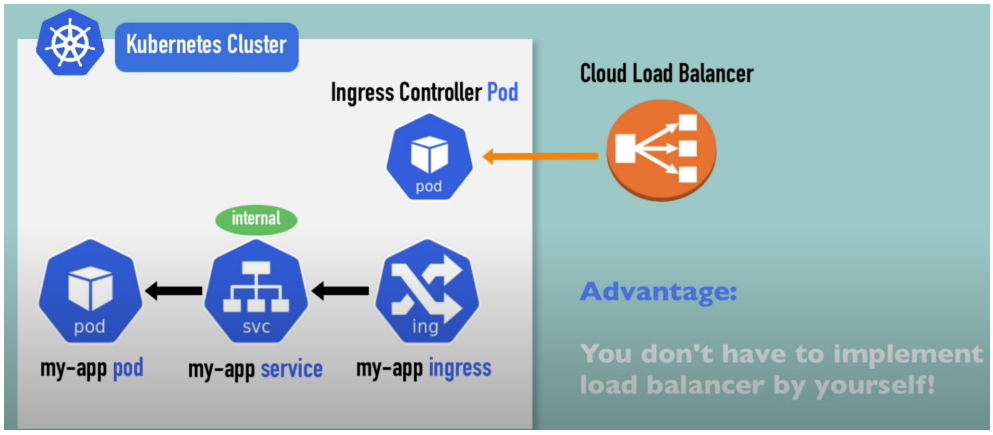
To install the implementation of Ingress in our cluster, we can choose from many different third party implementations

Notes:
Kubernetes provides K8s Nginx Ingress Controller

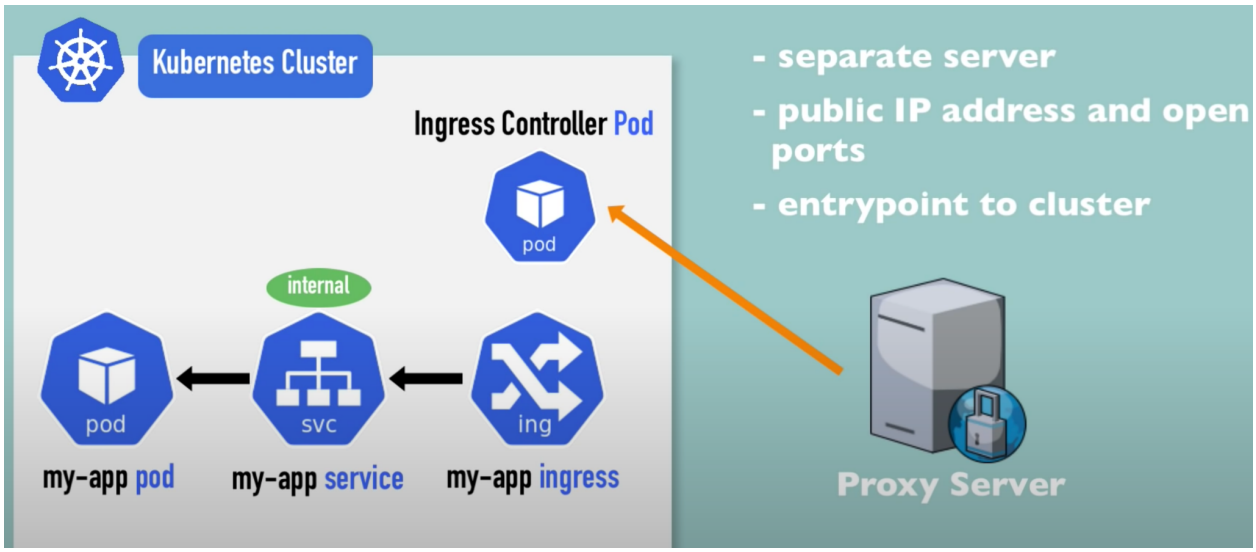
Cloud service providers:

AWS, google cloud, Azure

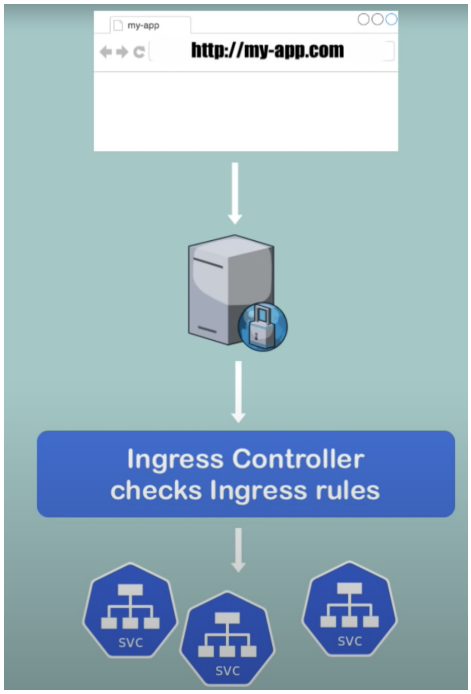
- Provides out of the box K8s solutions
- Provides own virtualized load balancers



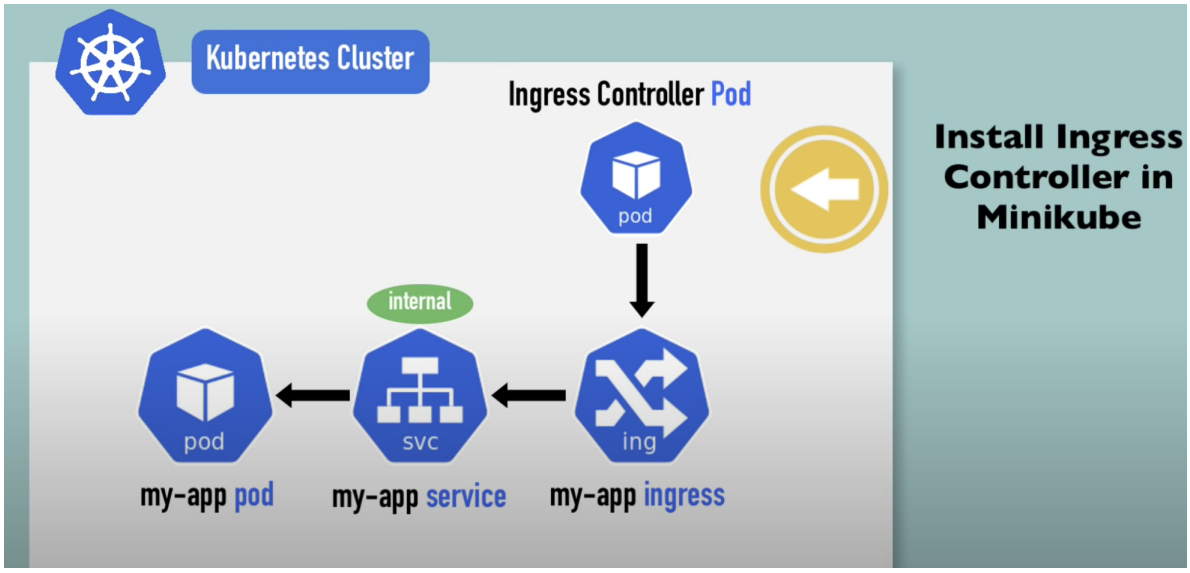
In case of in house Infrastructure, we would need to configure some kind of entrypoint.
Either inside of a cluster or outside as a separate server.

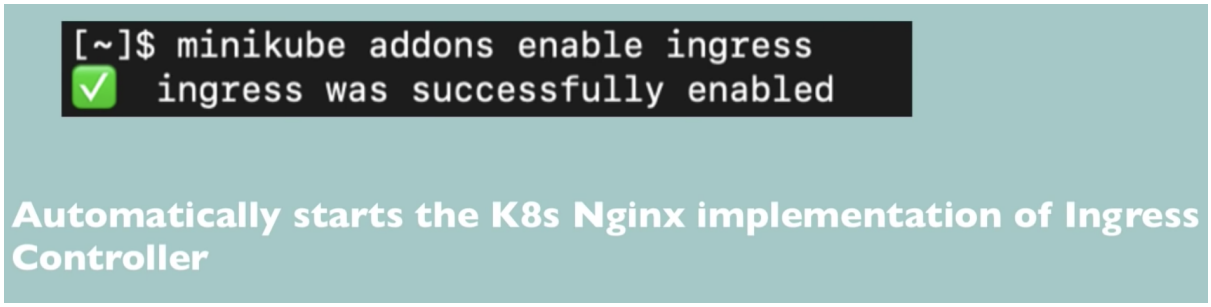


Note: No server in K9s cluster is accessible from outside



Ingress controller in Minikube:





Creating a sample Dashboard Ingress:

dashboard-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
spec:
  rules:
  - host: dashboard.com
    http:
      paths:
      - backend:
          serviceName: kubernetes-dashboard
          servicePort: 80
```

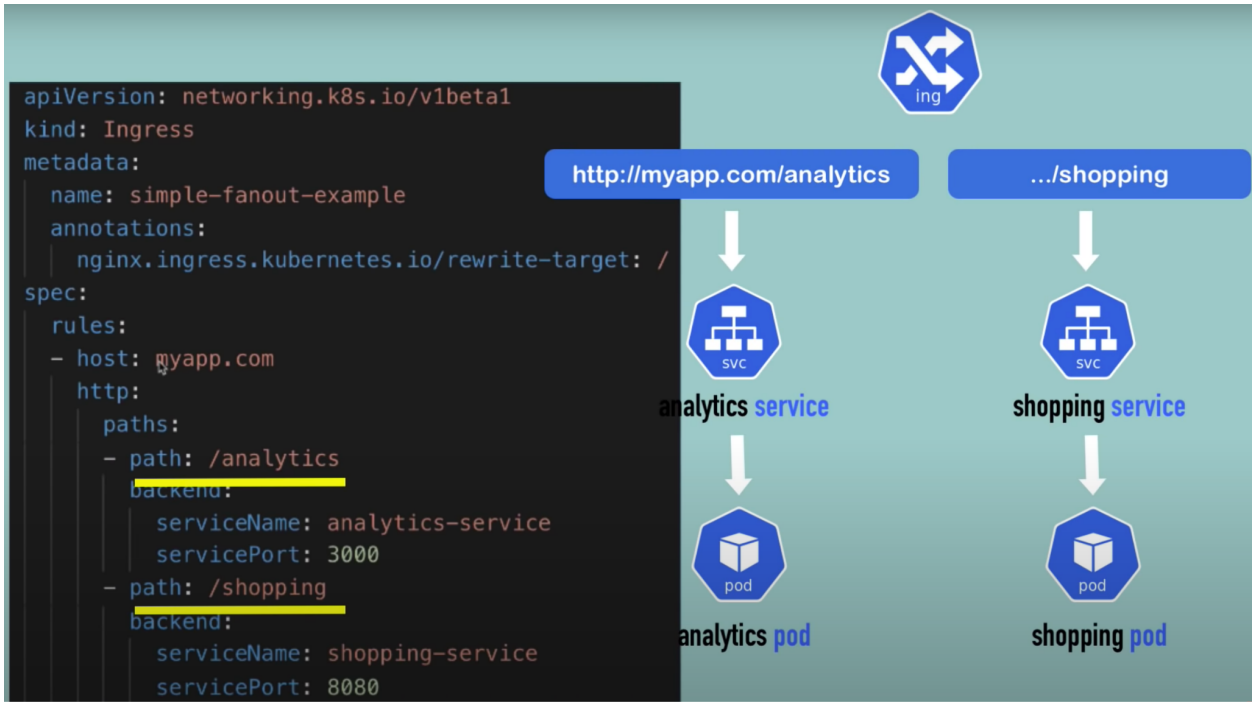
```
[~]$ kubectl apply -f dashboard-ingress.yaml
ingress.networking.k8s.io/dashboard-ingress created
[~]$ kubectl get ingress -n kubernetes-dashboard
NAME                HOSTS           ADDRESS   PORTS   AGE
dashboard-ingress   dashboard.com             80      14s
```

Note:

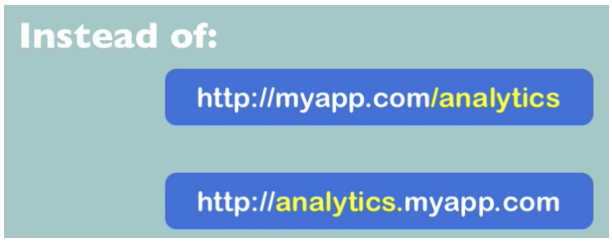
- The address is empty as it takes some time to assign ip address to it
- If there is no backend rules for ingress, it will use the default backend

Configuring default backend in Ingress:

Multiple Paths for the same host:

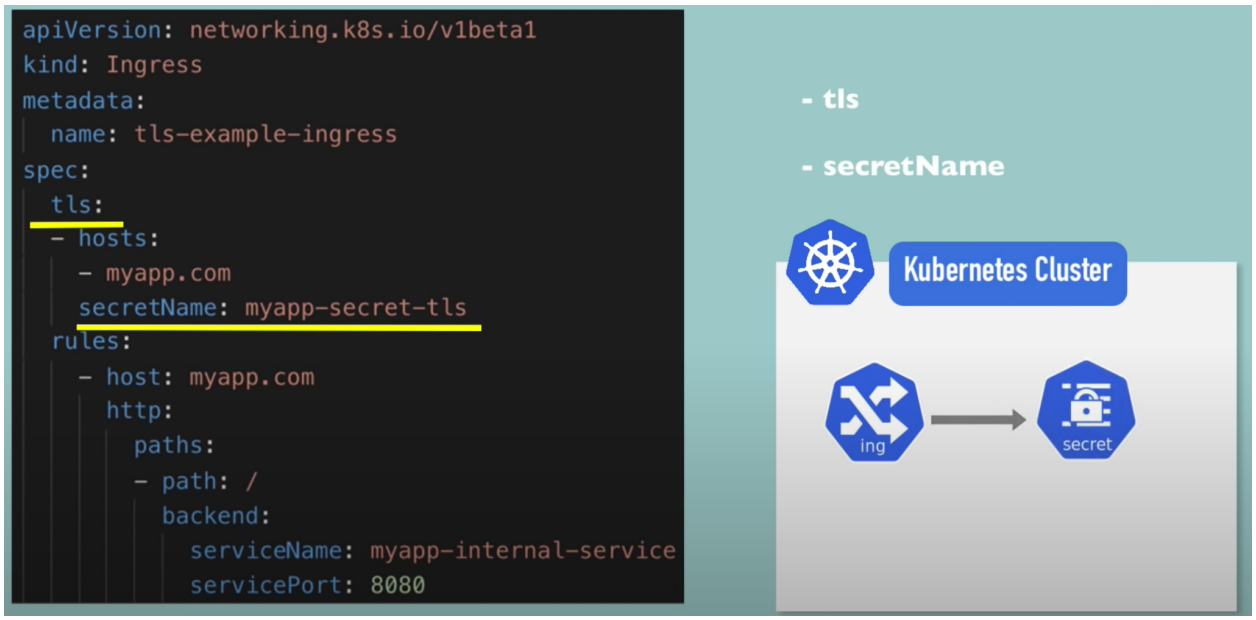


Different Subdomains

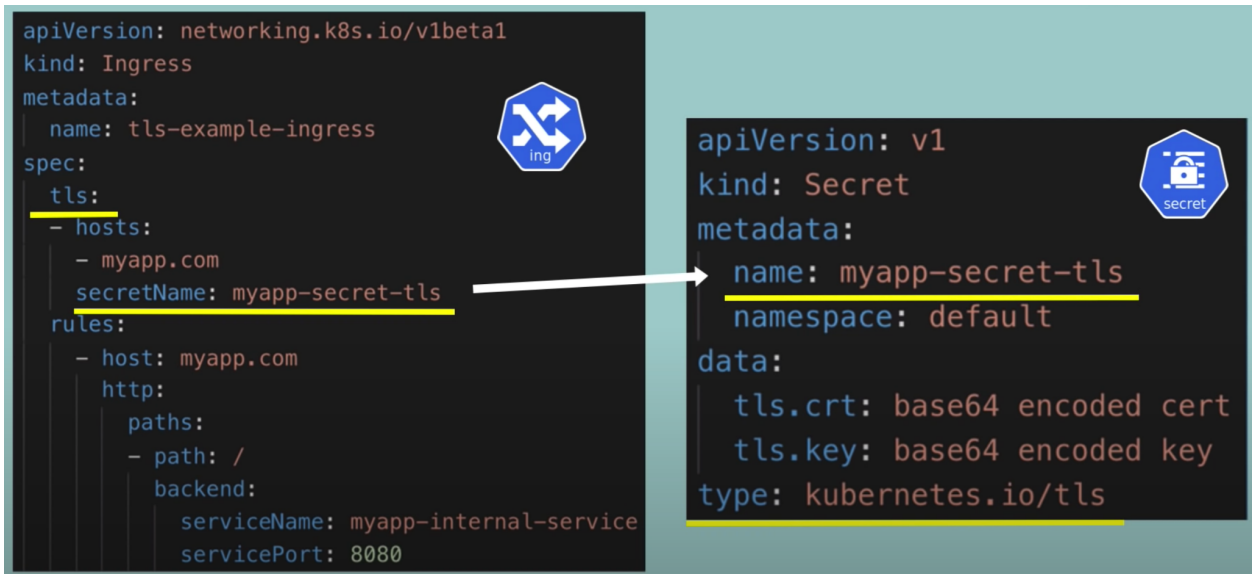


```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: analytics.myapp.com
    http:
      paths:
        backend:
          serviceName: analytics-service
          servicePort: 3000
  - host: shopping.myapp.com
    http:
      paths:
        backend:
          serviceName: shopping-service
          servicePort: 8080
```

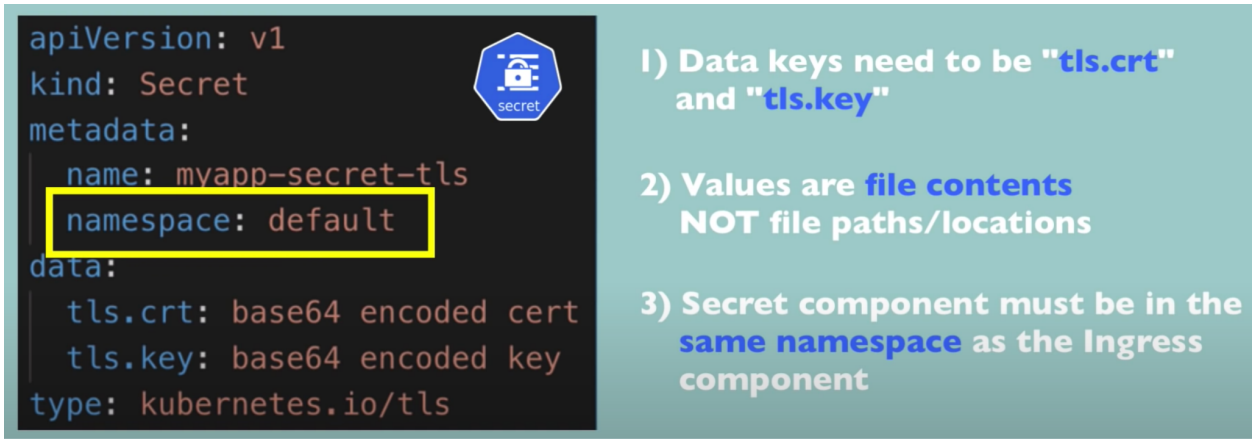
Configuring TLS Certificates:



To add TLS, we would need to create a secret component that will hold the certificate details



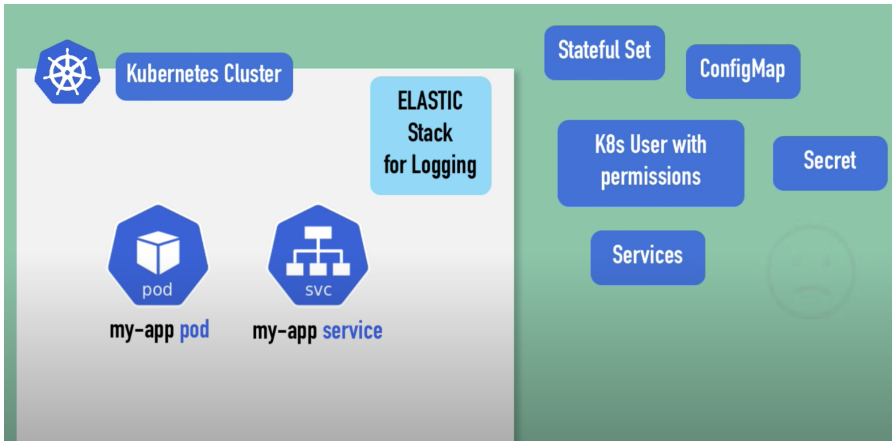
Key Notes:



Helm:

What is helm?

- It is a package manager of Kubernetes similar to apt, yum, homebrew



Creating all the above components in K8s cluster is a tedious task.
Elastic stack deployment is a standard process across all clusters, other folks can face same issue of deploying it

Uses:

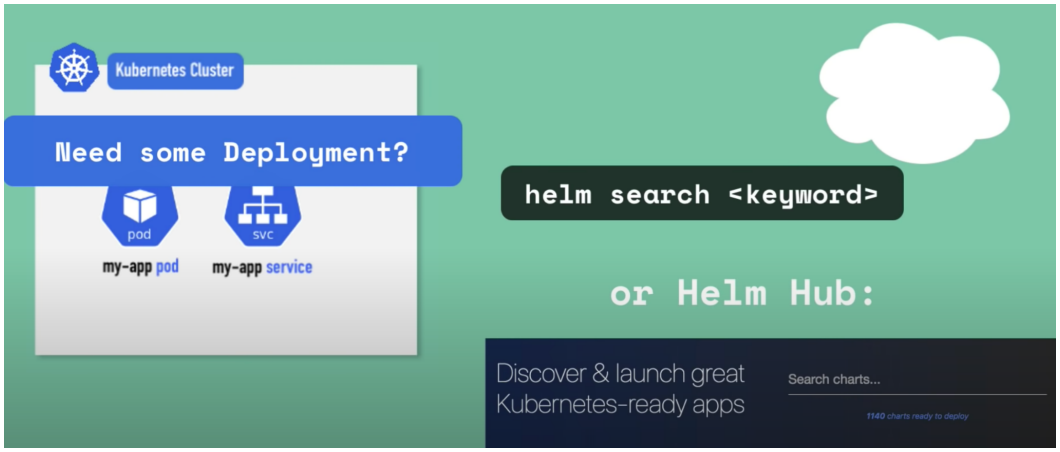
- It is a bundle of YAML files
- Using Helm we can create YAML files to bundle all the configuration for creating different k8s components.
- Create our own helm chart with helm
- Push the helm charts to Helm repository
- We can download and use existing helm charts
- So commonly used applications like database apps, Elastic Search, Mongo db, MySQL, monitoring applications like prometheus. For all these applications, charts are available in helm repository

We can use helm command to Search, pull helm charts from Helm repository.

Common actions for Helm:

- helm search: search for charts
- helm pull: download a chart to your local directory to view
- helm install: upload the chart to Kubernetes
- helm list: list releases of charts

Feature 1: Sharing Helm Charts



Feature 2: Templating engine