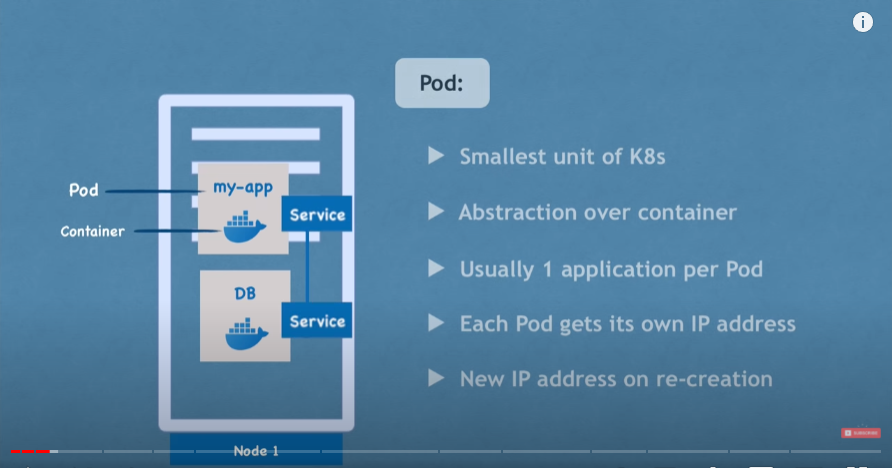
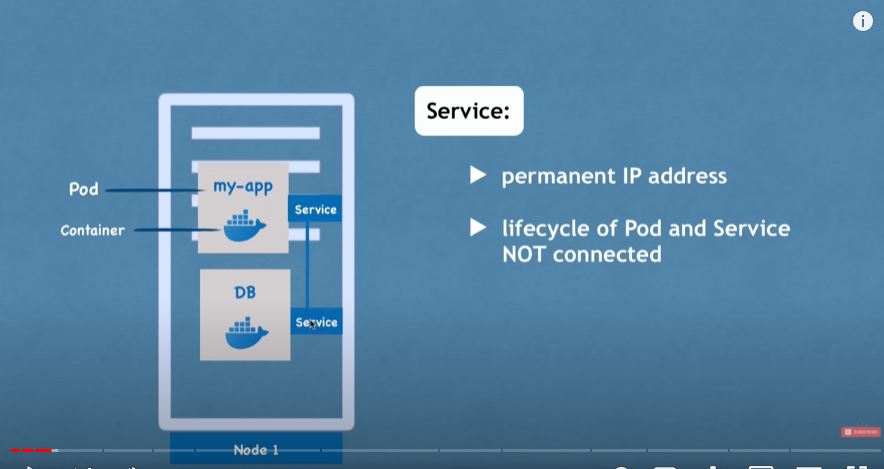
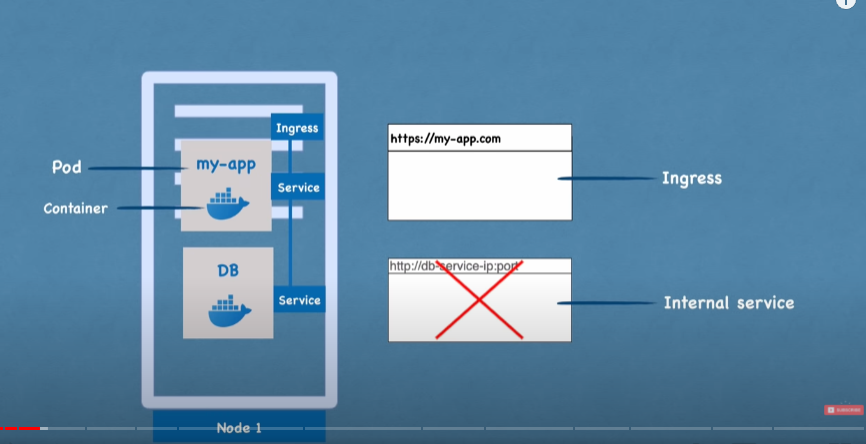
**# Pod**



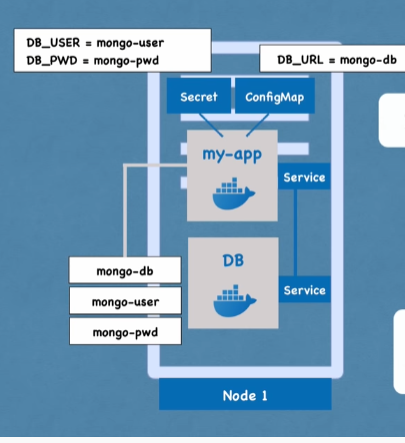
**# Service & ingress**



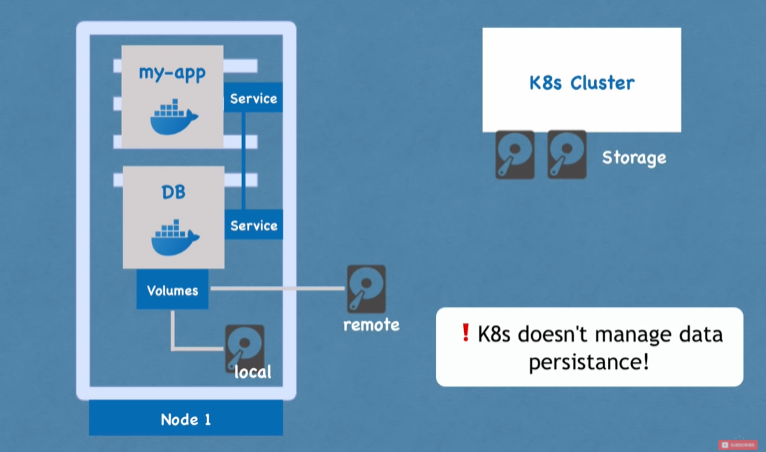


Ingress route external traffic into the cluster

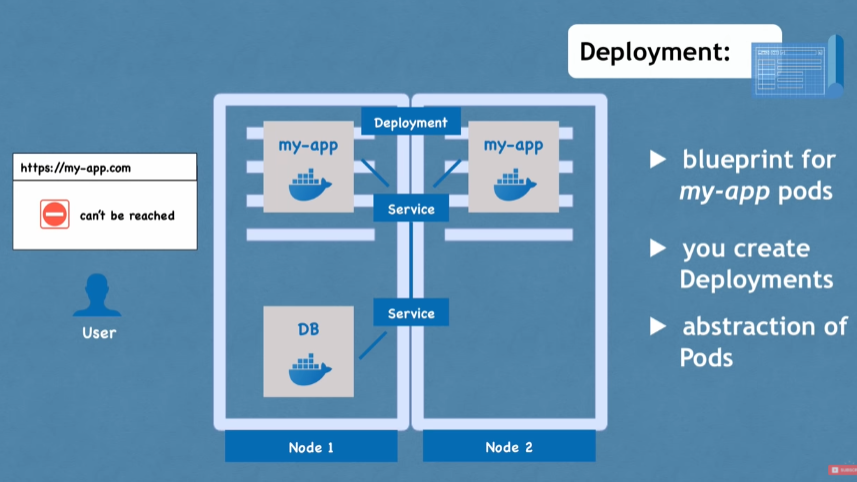
**# configMap and secret (external configurations)**

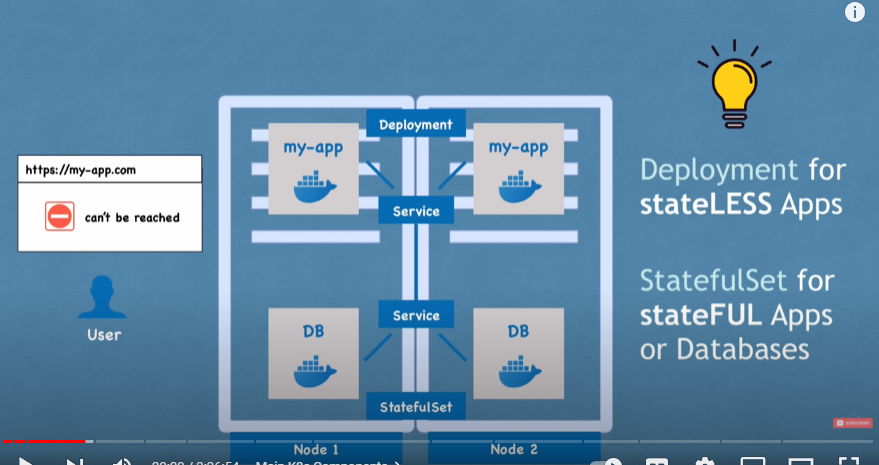


**# volume (Data persistence)**



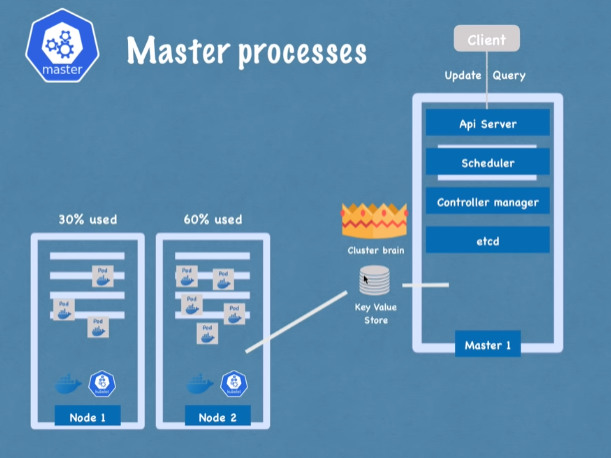
**# Deployment & StatefulSet**

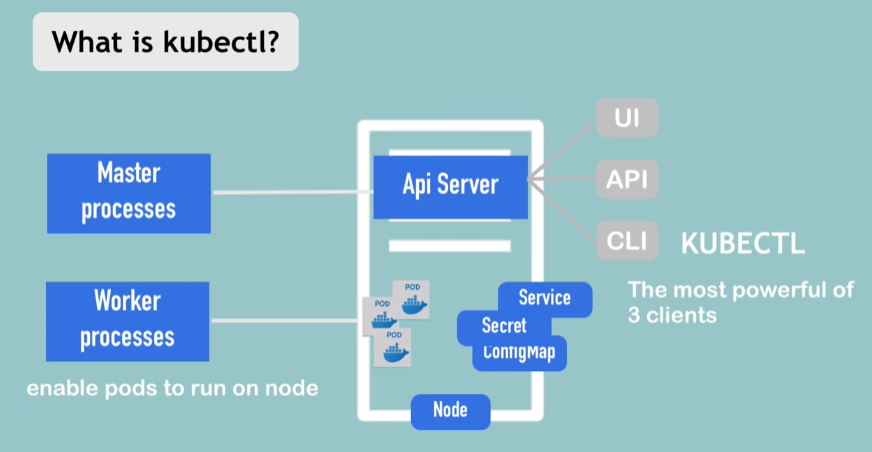




Common practice: DBs are often hosted outside of K8s cluster. And stateless apps are connected to external DBs

**# K8s architecture**





**# kubectl**

**CRUD**

**## Query**

## kubectl get all

## kubectl get node

## kubectl get service

## kubectl get pod

-o wide (for mor information)

-o yaml

## kubectl describe pod <pod-name>

**## Edit**

## kubectl edit deployment <deployment-name>

**## Create**

## kubectl create -h

## kubectl create deployment name image [options]

Not good, then yaml configuration comes to aid

## kubectl create namespace my-namespace

**## Delete**

## kubectl delete deployment <deployment-name>

**## Use config-file for CRUD**

## kubectl apply -f <config-file.yaml>

This will apply to all create and update operations as the config-file.yaml is declarative rather than imperative.

And to all components alongside deployment.

## kubectl delete -f <config-file.yaml>

**## Debug**

## kubectl logs [-f] <pod -name>

## kubectl exec -it <pod-name> -- bin/bash

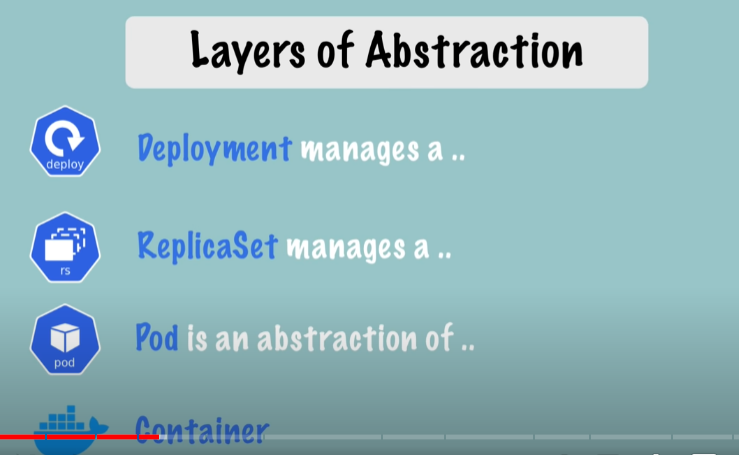
exec commands in the container of the pod

## pod-name:

<deployment-name>-<replicaset-id>-<pod-id>

## deployment manages replicaset that manages pod, which is an abstraction of containers

users only interact with deployment directly, anything below deployment is managed by kubernetes



**# configuration.yaml**

## kind: Deployment

spec (first) : deployment specification

template: pod

spec (second): pod specification

## 3 parts:

1. Kind & Metadata
2. Specification (specific to the kind of the component)
3. Status (automatically generated for status consistency detection)

* Etcd holds the current status of any K8s component

Pod has its own configuration under template in a deployment configuration

## labels & selector:

selector under spec pick components from specific labels

deployment -> pod

service -> deployment

## Ports in Service and Pod

### In service:

ports:

* protocol: TCP

port: 80 ( the port to visit from this service ip)

targetPort: 8080 ( the port to mapped to the pods belonging to this service)

### In Pod:

containers:

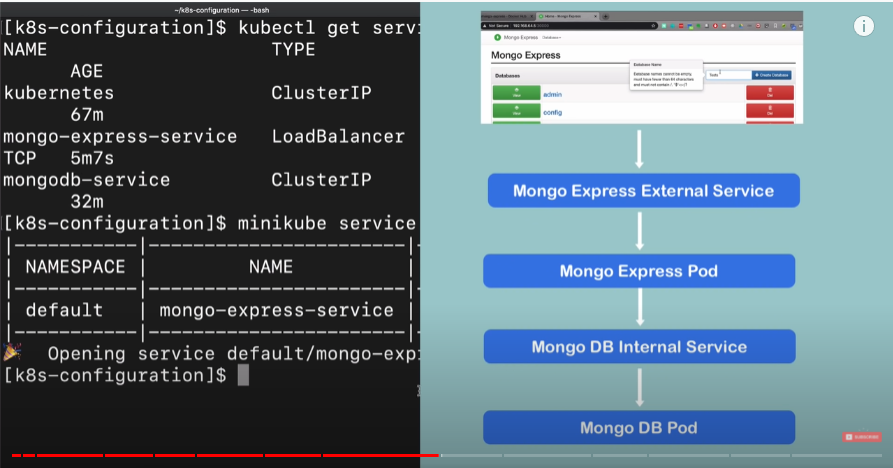
- name: nginx

Image: nginx:1.16

ports:

- containerPort: 8080 (port exposed by the container)

## Example:



### mongo-configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: mongodb-configmap

data:

database\_url: mongodb-service (difference is that this is in plain text)

### secret.yaml:

apiVersion: v1

kind: Secret

metadata:

name: mongodb-secret

type: Opaque

data:

mongo-root-username: dXNlcm5hbWU=

mongo-root-password: cGFzc3dvcmQ=

### mongodb.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongodb-deployment

labels:

app: mongodb

spec:

replicas: 1

selector:

matchLabels:

app: mongodb

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 (secret needs to be created ahead)

key: mongo-root-username

- name: MONGO\_INITDB\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mongodb-secret

key: mongo-root-password

--- (yaml document separator)

apiVersion: v1

kind: Service

metadata:

name: mongodb-service

spec:

selector:

app: mongodb

ports:

- protocol: TCP

port: 27017

targetPort: 27017

### 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

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

valueFrom:

configMapKeyRef:

name: mongodb-configmap (same as secret, config map needs to be created in k8s ahead)

key: database\_url

---

apiVersion: v1

kind: Service

metadata:

name: mongo-express-service

spec:

selector:

app: mongo-express

type: LoadBalancer

ports:

- protocol: TCP

port: 8081 (internal port)

targetPort: 8081 (port mapped to belonging containers)

nodePort: 30000 (Port for external IP address, has to be between 30000-32767)

### service type:

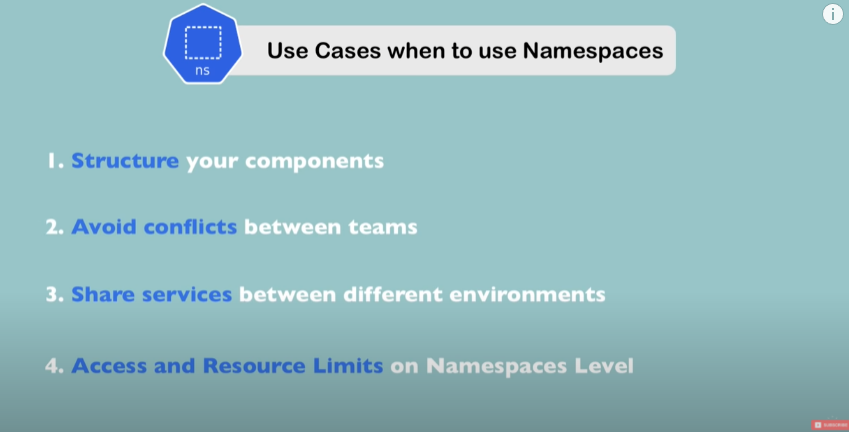
type:

ClusterIP (internal service, default type)

LoadBalancer (external service, this will assign service an external IP address and so accepts external requests)

# Namespace

Group resources into catagories



Service defined in one namespace can be referenced across namespaces

Config and Secret are defined in each namespace

Volume and node not defined in namespace, live globally in a cluster

apiVersion: v1

kind: ConfigMap

metadata:

name: mongodb-configmap

namespace: my-namespace

data:

database\_url: mongodb-service (difference is that this is in plain

text)

kubectl get configmap -n my-namespace

## change active namespace:

kubens my-namespace

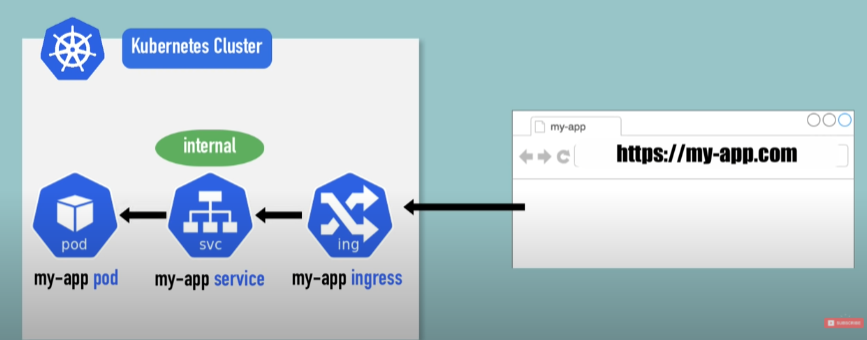
# Kubernetes Cluster DNS for services and pods

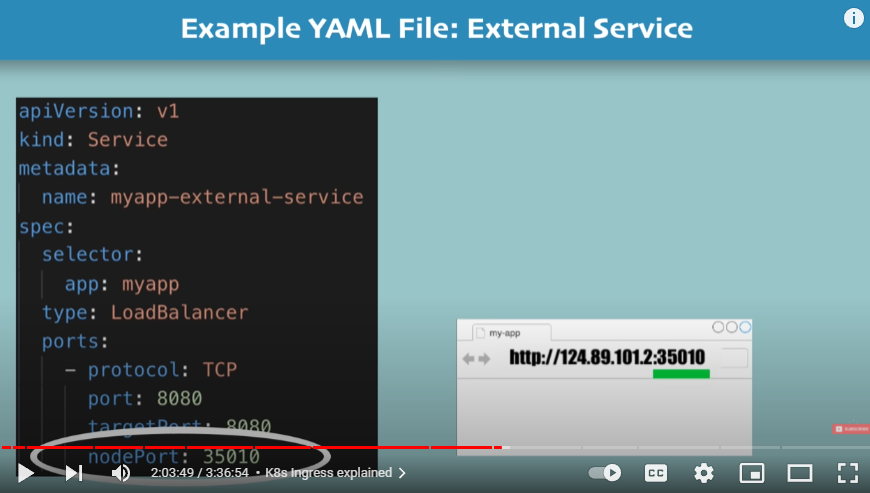
Kubernetes creates DNS records for services and pods. You can contact services with consistent DNS names instead of IP addresses.

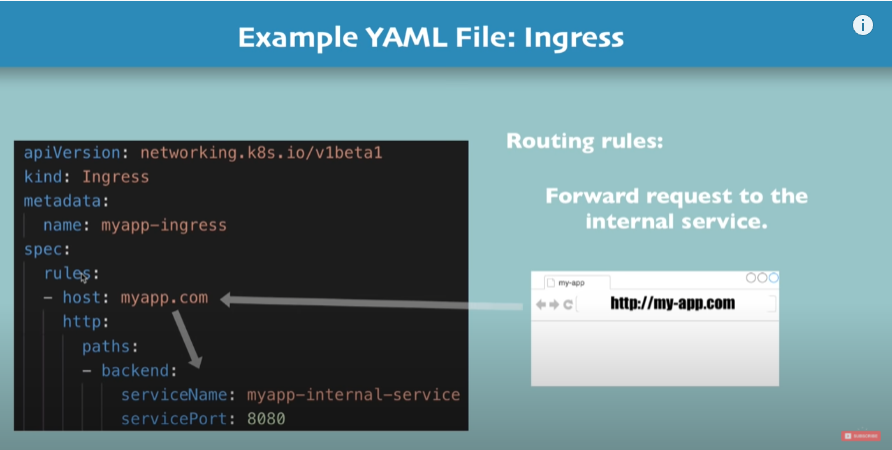
<service-name>.<namespace>.svc.cluster.local

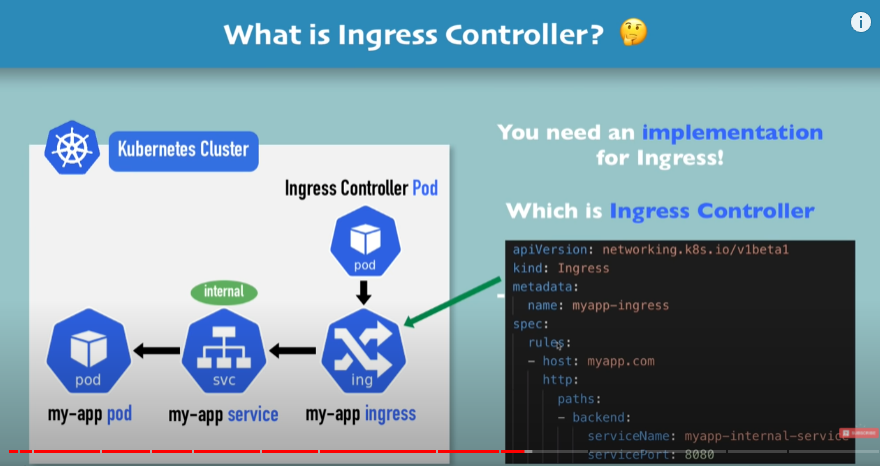
# K8s ingress

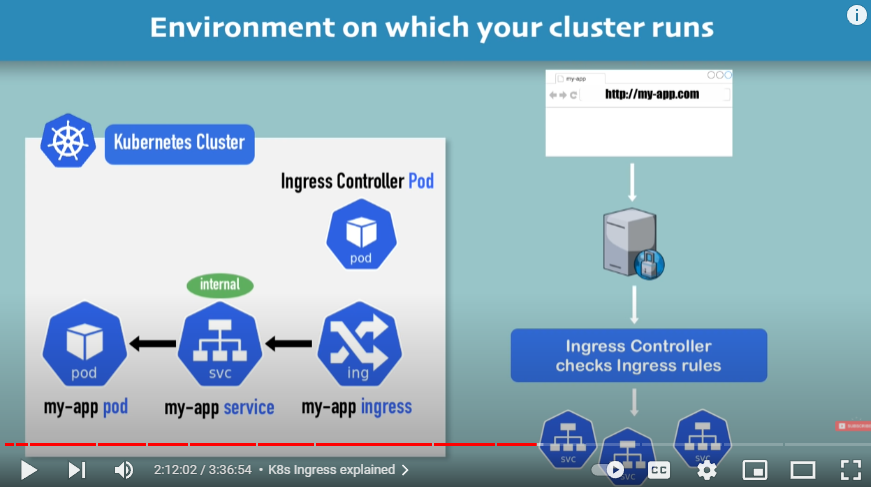
External service is assigned an IP address, when used externally, it’s not really for production





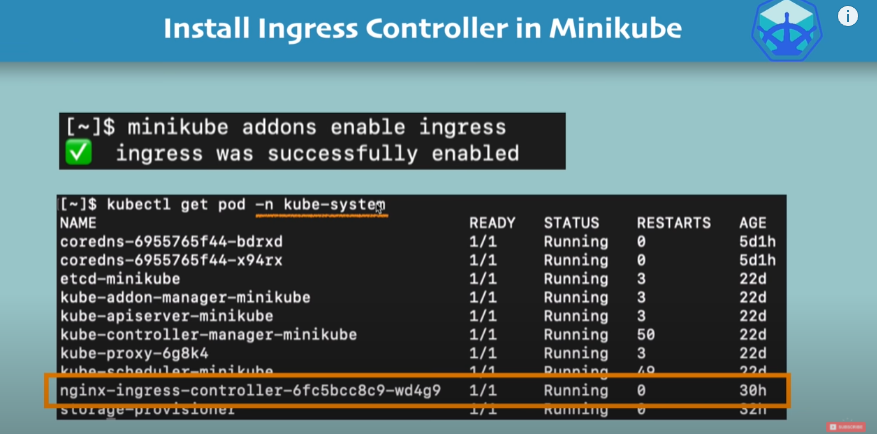




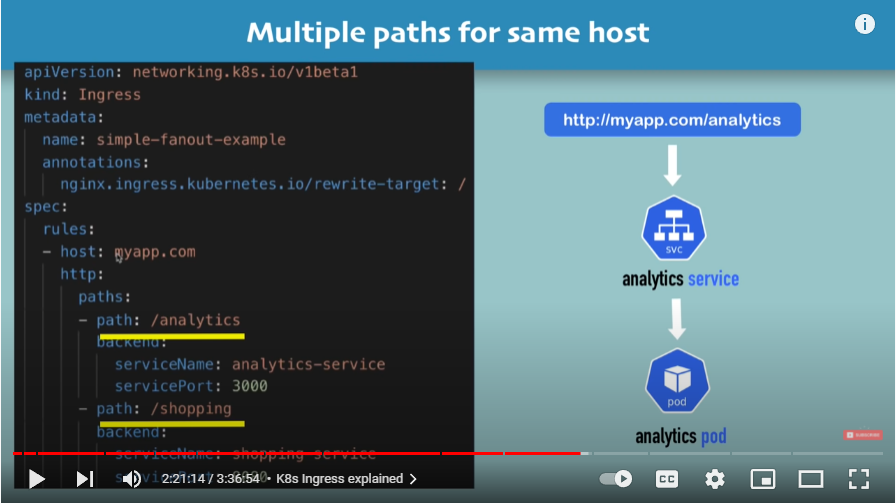


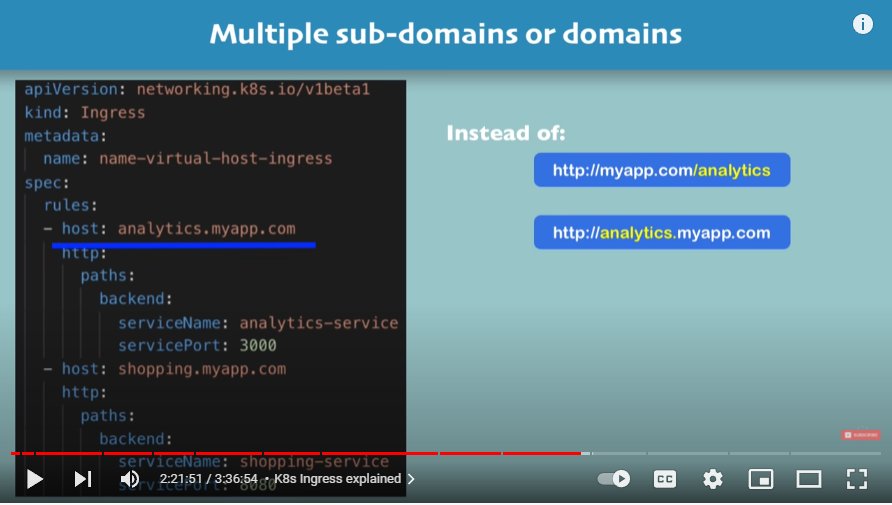
After implementation of Ingress Controller is installed, you can check that under kube-system:

kubectl get pod -n kube-system



## example:





**# TLS:**

