



Module Checklist

Container Orchestration with Kubernetes

By Techworld with Nana

Video Overview



- ★ Introduction to Kubernetes
- ★ Basic Concepts and Main K8s Components
- ★ Kubernetes Architecture
- ★ Minikube and kubectl - Local Setup
- ★ Kubernetes CLI - Main kubectl commands
- ★ Introduction to YAML Configuration File
- ★ Demo project: Deploying MongoDB and Mongo Express
- ★ Organizing components with Namespaces
- ★ Kubernetes Services
- ★ Kubernetes Ingress
- ★ Persisting Data with Volumes
- ★ ConfigMap & Secret Volume Types
- ★ Deploying stateful Apps with StatefulSet
- ★ Introduction to Managed Kubernetes Services
- ★ Helm - Package Manager of Kubernetes
- ★ Helm Demo: Install a Stateful Application on Kubernetes using Helm
- ★ Demo: Deploy App from Private Docker Registry
- ★ Extending the K8s API with Operators
- ★ Prometheus Operator Demo with Helm: Setup Prometheus Monitoring in K8s
- ★ Secure your cluster - Authorization with RBAC
- ★ Microservices in Kubernetes
- ★ Demo project: Deploy Microservices Application
- ★ Production & Security Best Practices
- ★ Demo project: Create Helm Chart for Microservices
- ★ Demo project: Deploy Microservices with Helmfile

Video Overview



Demo Projects	
Kubernetes Demo	https://gitlab.com/nanuchi/bootcamp-kubernetes
Demo Project for Private Docker Registry	https://gitlab.com/nanuchi/developing-with-docker
Online Shop Microservices	https://github.com/nanuchi/microservices-demo
Configuration Files & Helm Chart for Microservices Application	https://gitlab.com/nanuchi/online-shop-microservices-deployment

Check your progress... 1/11

Introduction to Kubernetes

- Watched video



Basic Concepts and Main K8s Components

- Watched videos

Useful Links:

- Managing K8s Secrets: <https://blog.aquasec.com/managing-kubernetes-secrets>

Kubernetes Architecture

- Watched videos

Minikube and Kubectl - Local Setup

- Watched videos
- Demo executed:**
 - Installed and setup Minikube
 - Installed Kubectl

Useful Links:

- Installation guide for Minikube (Mac, Linux and Windows):
<https://minikube.sigs.k8s.io/docs/start/>
- Installation guide for Kubectl:
<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

Check your progress... 2/11

Kubernetes CLI - Main kubectl commands

- Watched video
- Demo executed:**
 - Created nginx Deployment
 - Edited Deployment
 - Created mongodb Deployment
 - Inspected logs of a Pod
 - Got shell of a running container - kubectl exec
 - Deleted deployment
 - Applied configuration file



Useful Links:

- Example commands repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/basic-kubectl-commands>

YAML Configuration File

- Watched videos

Useful Links:

- Configuration File:
<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/>
- Example files repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/kubernetes-configuration-file-explained>

Best practices:

- Store Configuration Files with your application code or own Git Repository just for the configuration files

Check your progress... 3/11

Demo project: Deploying MongoDB and Mongo Express

- Watched videos
- Prerequisite:
 - Minikube cluster running
- Demo executed - Deploying MongoDB and MongoExpress:
 - Created MongoDB Deployment
 - Created Secret for Mongo Credentials
 - Created MongoDB Internal Service
 - Created MongoExpress Deployment
 - Created ConfigMap for DB Server URL
 - Created Mongo Express External Service



Useful Links:

- MongoDB Docker Image: https://hub.docker.com/_/mongo
- Mongo Express Docker Image: https://hub.docker.com/_/mongo-express
- Project Repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/demo-kubernetes-components>

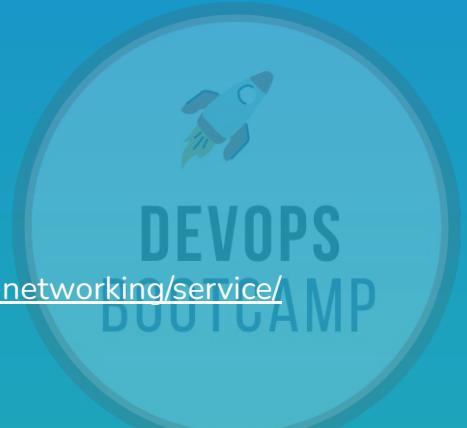
Organizing components with Namespaces

- Watched videos

Useful Links:

- Kubectx: <https://github.com/ahmetb/kubectx#installation>

Check your progress... 4/11



Kubernetes Services

- Watched video

Useful Links:

- Service: <https://kubernetes.io/docs/concepts/services-networking/service/>

Best Practice:

- Do NOT use NodePort Service Type for external connections. Use Ingress or Load Balancer instead.

Ingress

- Watched video

Useful Links:

- Project repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/kubernetes-ingress>
- List of Ingress Controllers you can choose from:
<https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>
- Ingress Controller Bare Metal:
<https://kubernetes.github.io/ingress-nginx/deploy/baremetal/>

Persisting Data with Volumes

- Watched video

Useful Links:

- Volume Types:
<https://kubernetes.io/docs/concepts/storage/volumes/#volume-types>
- Project demo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/kubernetes-volumes>

Check your progress... 5/11

ConfigMap & Secret Volume Types

- Watched video
- Demo executed:**
 - Created Mosquitto Deployment without any volumes
 - Created ConfigMap component to overwrite mosquito.conf file
 - Created Secret component to add passwords file
 - Adjusted Mosquitto Deployment to include volumes



Useful Links:

- Project demo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/configmap-and-secret-volumes>
- ConfigMap Volume Type:
<https://kubernetes.io/docs/concepts/storage/volumes/#configmap>
- Secret Volume Type: <https://kubernetes.io/docs/concepts/storage/volumes/#secret>
- Mosquitto Public Docker Image: https://hub.docker.com/_/eclipse-mosquitto

Deploying Stateful Apps with StatefulSet

- Watched videos

Introduction to Managed Kubernetes Services

- Watched videos

Helm - Package Manager of Kubernetes

- Watched videos

Useful Links:

- Install Helm: <https://helm.sh/docs/intro/install/>
- Helm Hub: <https://artifacthub.io/>

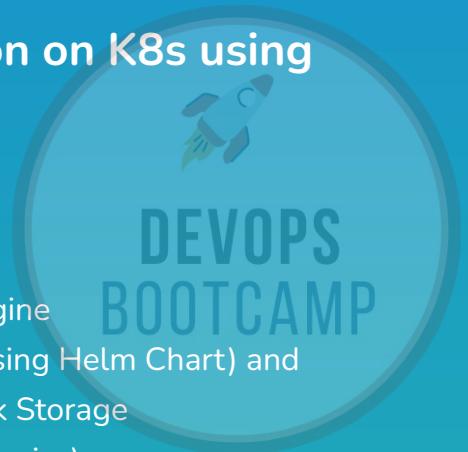
Check your progress... 6/11

Helm Demo: Install a Stateful Application on K8s using Helm

- Watched video
- Demo executed:**
 - Created K8s cluster on Linode Kubernetes Engine
 - Deployed replicated MongoDB (StatefulSet using Helm Chart) and configured Data Persistence with Linode Block Storage
 - Deployed MongoExpress (Deployment and Service)
 - Deployed NGINX Ingress Controller as Loadbalancer (using Helm Chart)
 - Configured Ingress rule

Useful Links:

- Project Repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/linode-kubernetes-engine-demo>
- Create a Linode account (\$100 - 60day credit with this link:
<https://bit.ly/31p4GW2>)
- Mongo Express Docker Image: https://hub.docker.com/_/mongo-express



Check your progress... 7/11

Demo: Deploy App from Private Docker Registry

- Watched videos
- Demo executed:**
 - Logged in to AWS Container Repository | docker login and create docker config.json file
 - Created Secret component
 - Configured Deployment for demo app
- Pre-Requisites:**
 - Setup a Private Docker Repository (e.g. AWS Elastic Container Registry)
 - Have a demo application (see provided one)



Useful Links:

- K8s Project Repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/pull-images-from-private-repository-in-k8s>
- Sample NodeJs application Repo:
<https://gitlab.com/nanuchi/developing-with-docker>

Extending the K8s API with Operators

- Watched videos

Useful Links:

- Find Operators: <https://operatorhub.io/>

Check your progress... 8/11

Prometheus Operator Demo with Helm: Setup Prometheus Monitoring on Kubernetes

- Watched videos
- Demo executed:**
 - Installed Prometheus Operator Helm Chart
 - Accessed Grafana UI (configured port-forward)
 - Accessed Prometheus UI (configured port-forward)



Useful Links:

- Prometheus Monitoring - What it is and how it works:
<https://youtu.be/h4Sl21AKiDq>
- Project Repo:
<https://gitlab.com/nanuchi/bootcamp-kubernetes/-/tree/master/setup-prometheus-operator>

Secure your Cluster - Authorization with RBAC

- Watched videos

Useful Links:

- Site dedicated to good practices and tooling around Kubernetes RBAC:
<https://rbac.dev/>
- Authenticating:
<https://kubernetes.io/docs/reference/access-authn-authz/authentication/>
- 3 Realistic Approaches to K8s RBAC:
<https://thenewstack.io/three-realistic-approaches-to-kubernetes-rbac/>

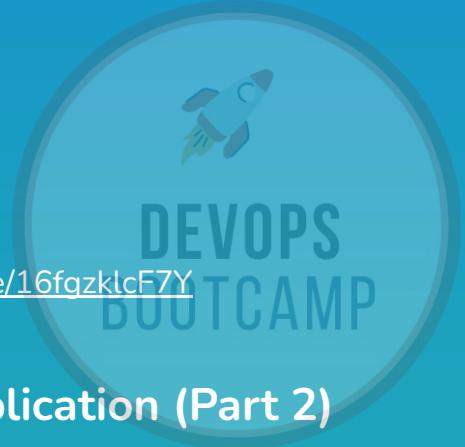
Check your progress... 9/11

Microservices in Kubernetes (Part 1)

- Watched videos

Useful Links:

- Learn more about Istio Service Mesh: <https://youtu.be/16fqzklcF7Y>



Demo Project: Deploy Microservices Application (Part 2)

- Watched videos

- Demo executed:**

- Created YAML file with 11 Deployment and corresponding Service manifests
- Note: All Services' Components are Internal Services, except the Frontend Service, which needs to be accessed from browser
- Created a K8s cluster with 3 Worker Nodes on Linode (or any other cloud platform)
- Connected to the cluster
- Created a Namespace and deployed all the micro services into it
- Accessed Online Shop with Browser

Useful Links:

- Microservices Git Repo: <https://github.com/nanuchi/microservices-demo>
- Git Repo for Configuration Files for the Microservices App: <https://gitlab.com/nanuchi/online-shop-microservices-deployment>
- Redis Docker Image: https://hub.docker.com/_/redis
- Volume Type emptyDir: <https://kubernetes.io/docs/concepts/storage/volumes/#emptydir>
- Ephemeral Volumes: <https://kubernetes.io/docs/concepts/storage/ephemeral-volumes/>

Check your progress... 10/11

Production & Security Best Practices (Part 3)

- Watched videos
- Demo executed - Improved Microservices Config Files
 - BP 1: Added version to each container image
 - BP 2: Configured Liveness Probe for each container
 - BP 3: Configured Readiness Probe for each container
 - BP 4: Configured Resource Requests
 - BP 5: Configured Resource Limits
 - BP 6: Don't use NodePort Service Type
 - BP 7: Configure more than 1 Replica for each Deployment



Useful Links:

- Git Repo for Best Practices Configuration Files for the Microservices App:
<https://gitlab.com/nanuchi/online-shop-microservices-deployment>
- Configure Liveness, Readiness Probes:
<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>
- Resource Requests & Limits:
<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits>

Check your progress... 11/11

Create Helm Chart for Microservices (Part 4)

- Watched videos
- Demo executed
 - Created “microservices” Helm Chart
 - Created values.yaml files for each microservice
 - Created “redis” Helm Chart and values file for it



Useful Links:

- Git Repo - Helm Charts:
<https://gitlab.com/nanuchi/online-shop-microservices-deployment>
- Helm Chart Developer Guide: https://helm.sh/docs/chart_template_guide/
- Built-In Objects: https://helm.sh/docs/chart_template_guide/builtin_objects/
- Best practices for creating Charts: https://helm.sh/docs/chart_best_practices/

Deploy Microservices with Helmfile (Part 5)

- Watched videos
- Demo executed
 - Deployed Microservices Application with “helm install”
 - Created Helmfile
 - Installed Helmfile
 - Deployed Helm Charts with Helmfile

Useful Links:

- Git Repo - Helmfile:
<https://gitlab.com/nanuchi/online-shop-microservices-deployment>
- Official Helmfile Repo: <https://github.com/roboll/helmfile>

More Resources...

More Best practices

- Configuration Best Practices:
<https://kubernetes.io/docs/concepts/configuration/overview/>
- 9 Security Best Practices:
<https://www.cncf.io/blog/2019/01/14/9-kubernetes-security-best-practices-everyone-must-follow/>

Cheatsheet

- K8s CLI Cheat Sheet: <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>





Container Orchestration with Kubernetes

Key Takeaways

Introduction to Kubernetes

Official Definition

- Open source **container orchestration tool**
- Developed by **Google**
- **Automates** many processes involved in deploying, managing and scaling containerized applications



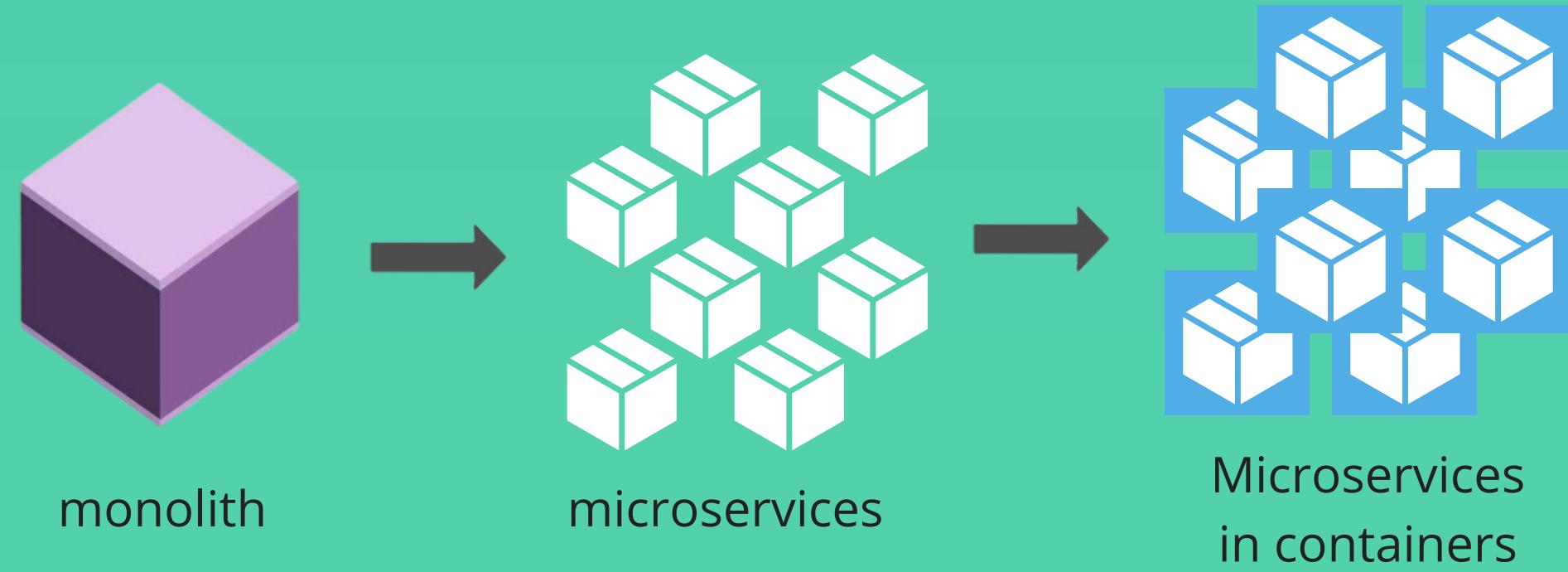
- **Most used** container orchestration platform
- Also known as "K8s" or "Kube"



k8s helps you manage containerized application in different deployment environments (physical / virtual / cloud)

The need for Kubernetes

- Trend from **Monolith** to **Microservices**
- Containers are the perfect host for microservice applications
- Resulted in an **increased usage of containers**



- Manually **managing hundreds or hundreds of 1000s containers** is a lot of effort
- Kubernetes automates many of those manual tasks and provides
 - **High Availability** or no downtime
 - Automatic **Scaling**
 - **Disaster Recovery** - Backup and Restore
 - **Self-Healing**



Core Kubernetes Components - 1

Kubernetes has many components, but these are the **main ones you need to know:**



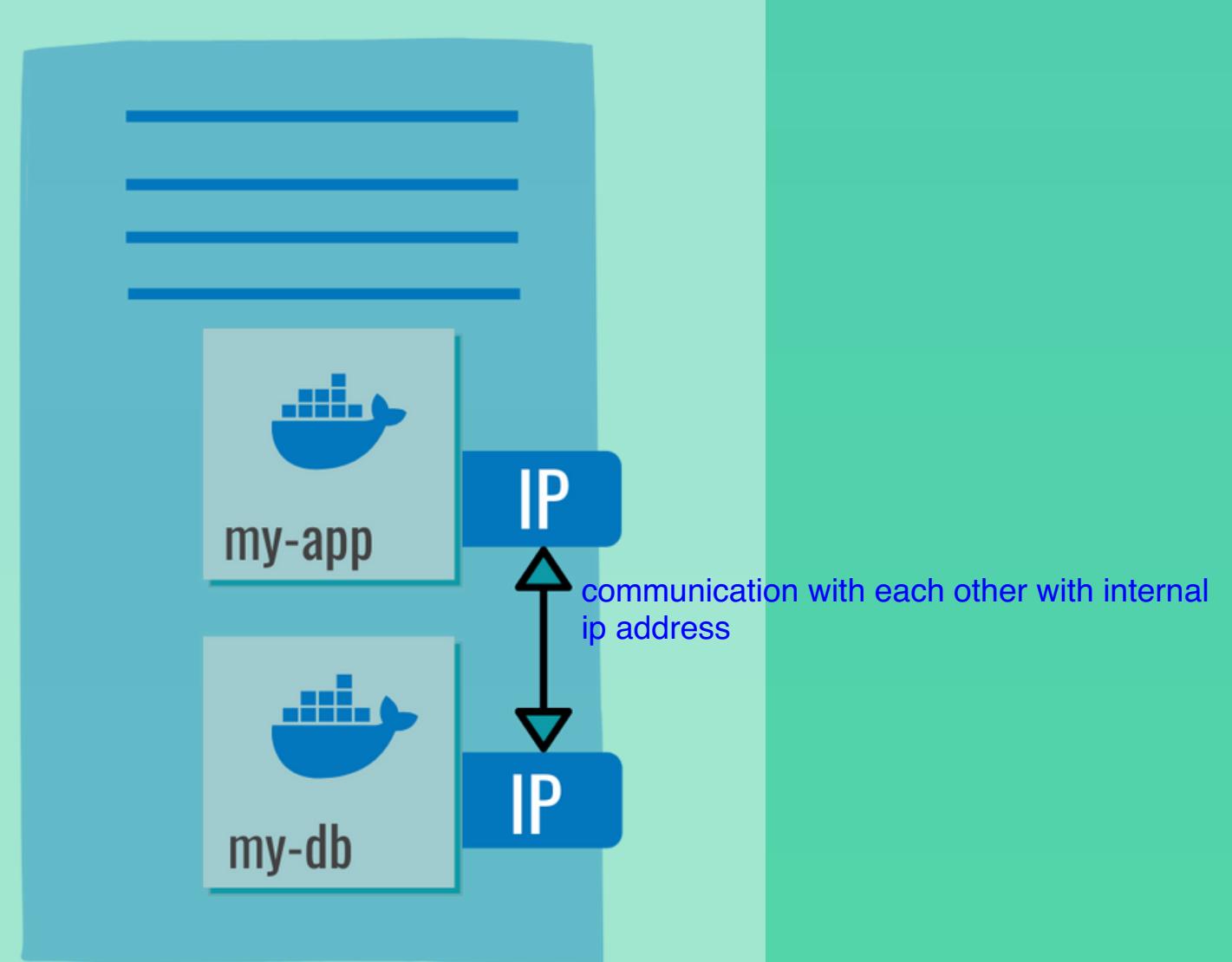
Core Kubernetes Components - 2

POD

- Group of 1 or more containers
- Smallest unit of K8s
- An abstraction over container
- Usually 1 application/container per Pod
- Pods are ephemeral



New IP address assigned on re-creation
if db container is crashed , new container is created with new ip -> using service



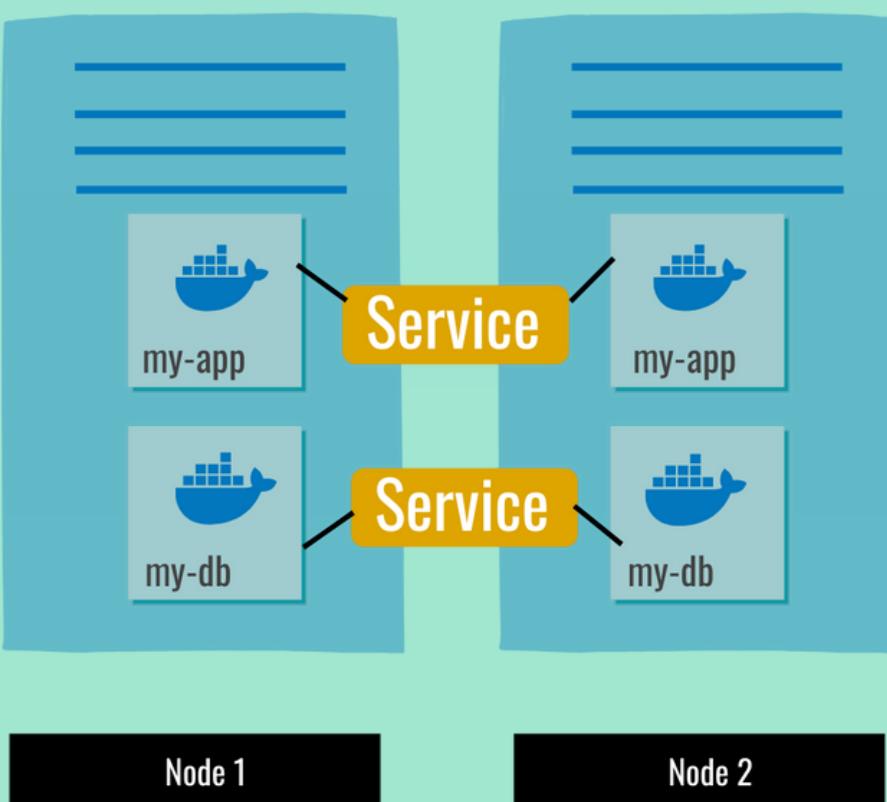
Core Kubernetes Components - 3

SERVICE

- Basically a static or **permanent IP address** that can be attached to each Pod
- Also serves as a **loadbalancer**
- **Lifecycles** of Service and Pod are **not connected**



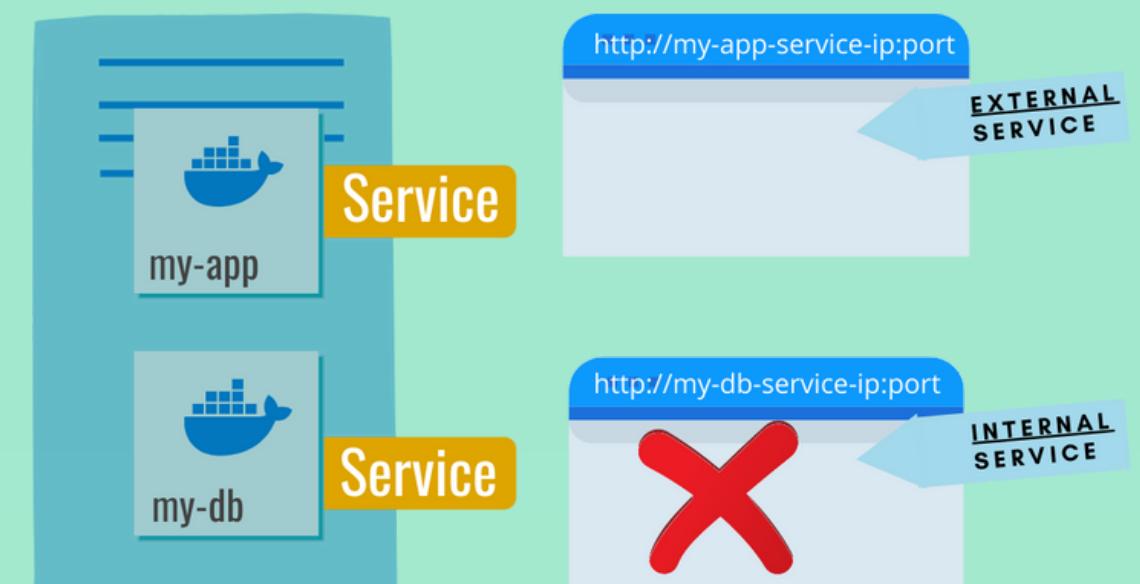
If Pod crashes, the Service and its IP address will be the same



Internal vs External Service

When creating a service you can specify its **type**:

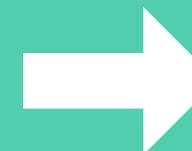
- **Internal Service:** By default, for example a database, which should not be accessible from outside
- **External Service:** Application accessible through browser



Core Kubernetes Components - 4



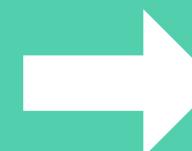
URL of external Service:



http://124.89.101.2:8080



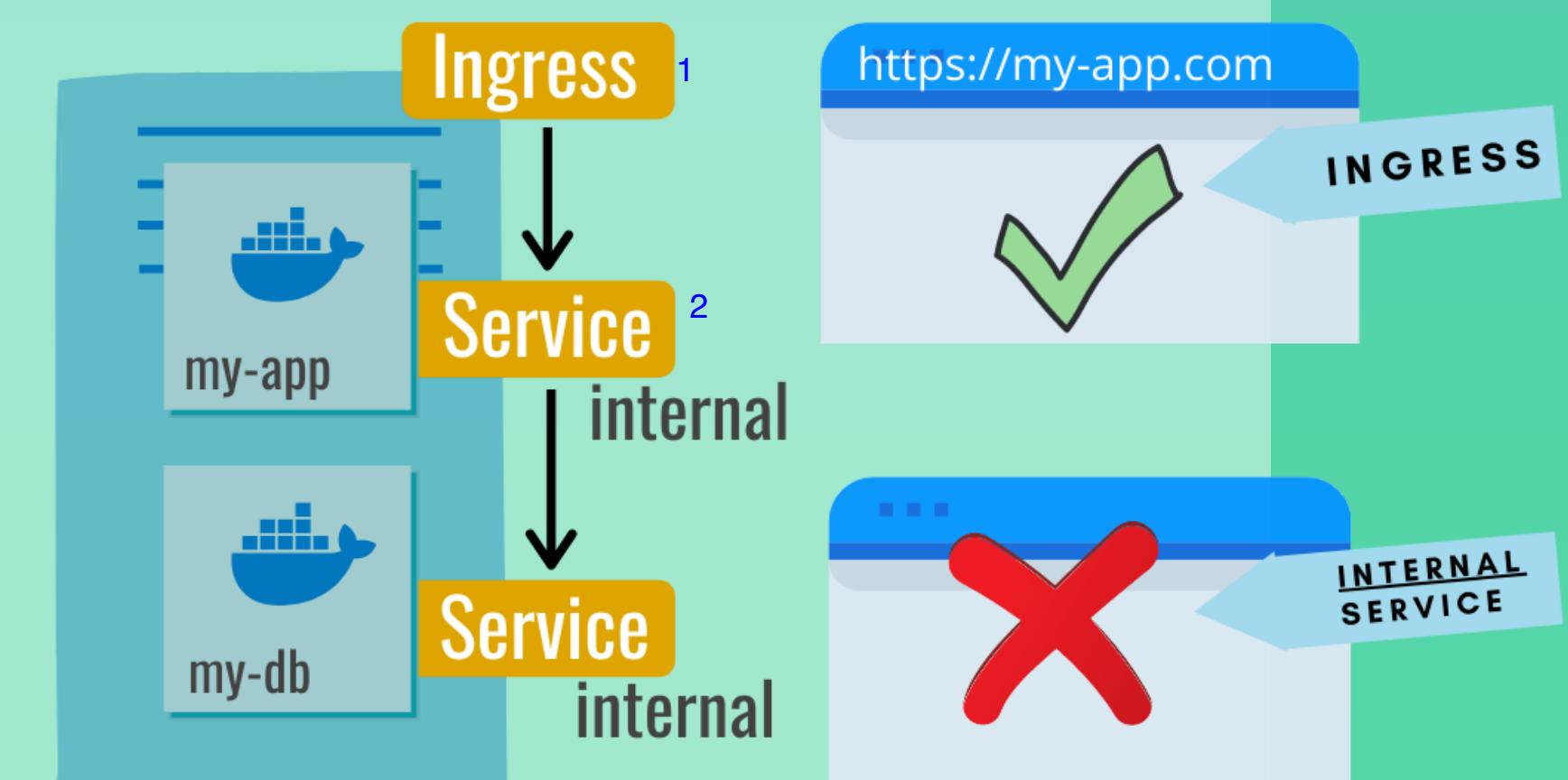
URL of Ingress Service:



https://my-app.com

INGRESS

- Ingress is the entrypoint to your K8s cluster
- Request goes to Ingress first, which does the forwarding to the Service

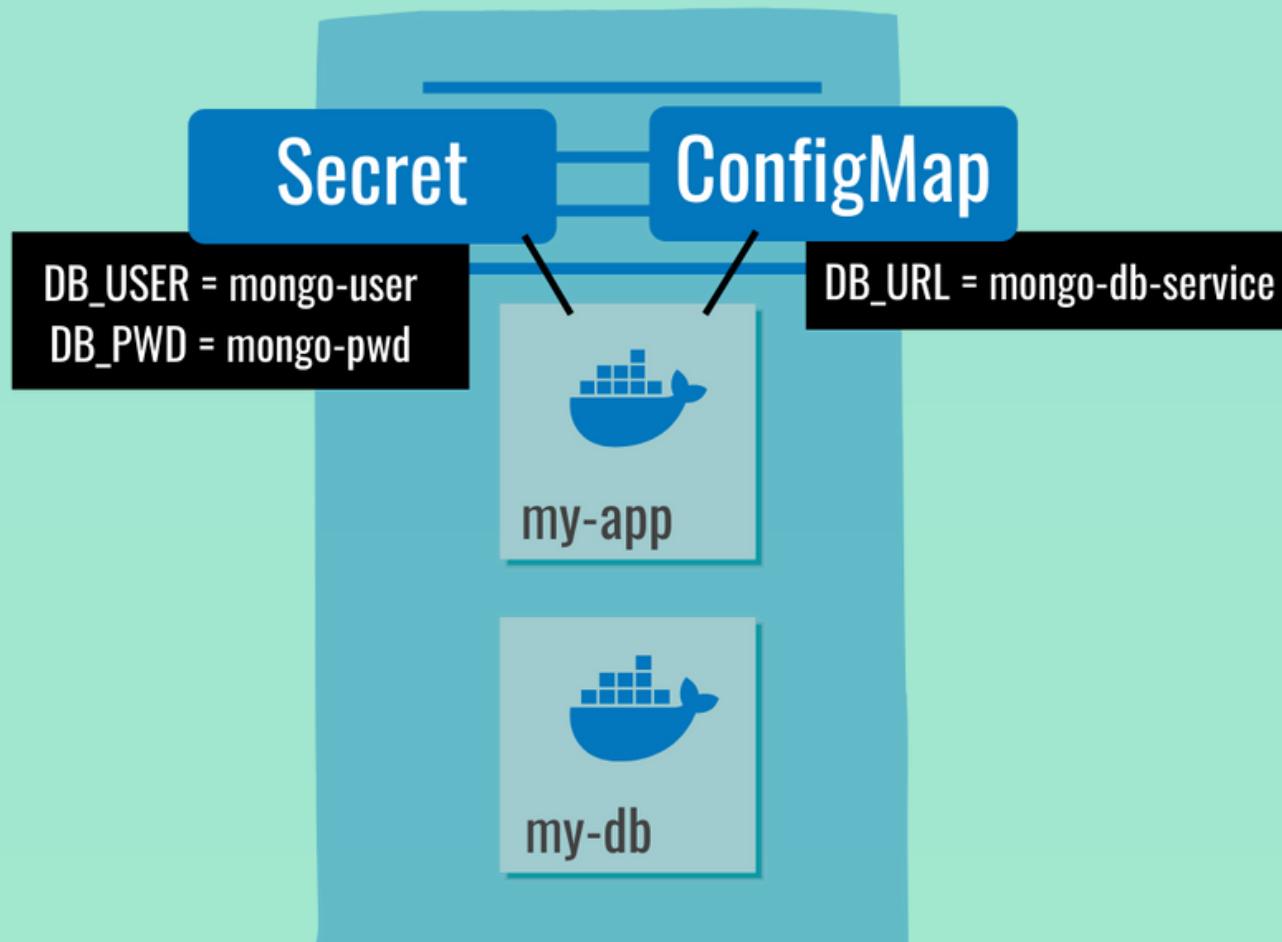


Core Kubernetes Components - 5

For **external configuration**, Kubernetes has these 2 components:

CONFIGMAP

- To **store non-confidential data** in key-value pairs



SECRET

- Similar to ConfigMap, but to **store sensitive data** such as passwords or tokens

- **Pods can consume** ConfigMaps and Secrets as environment variables, CLI arguments or as config files in a Volume



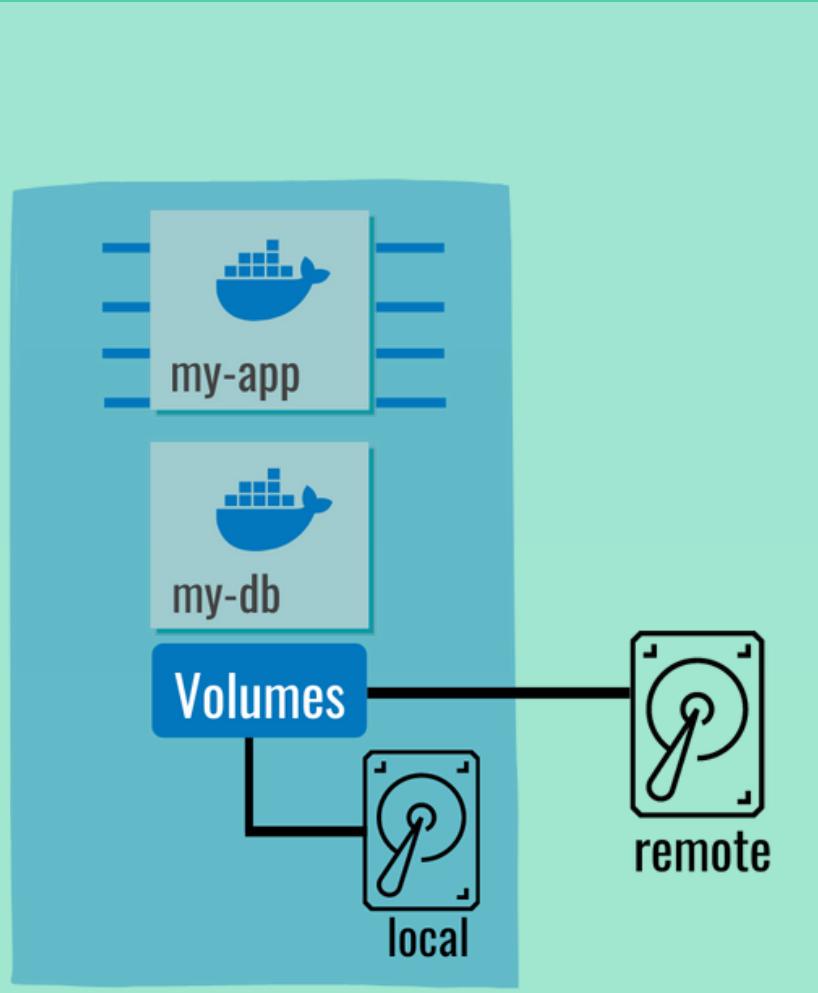
Storing the data in a Secret component doesn't automatically make it secure. There are built-in mechanisms (like encryption, defining authorization policies) for basic security, which are not enabled by default! Recommended to use third-party secret management tools, because the provided capabilities by K8s are not enough for most companies

Core Kubernetes Components - 6

When a container crashes, K8s restarts the container but with a clean state. Meaning your **data is lost!**

VOLUME

- Volume component basically attaches a physical storage on a hard drive to your Pod
- Storage could be either on a local server or outside the K8s cluster



Kubernetes cluster 



storage

Think of storage as an **external hard drive plugged in** to your K8s cluster



K8s doesn't manage any data persistence, meaning **you are responsible for backing up, replicating the data etc.**

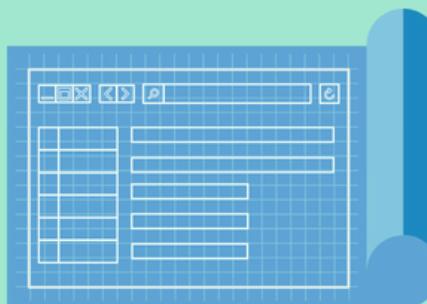


Core Kubernetes Components - 7

Deployment and StatefulSet are an abstraction of Pods

DEPLOYMENT

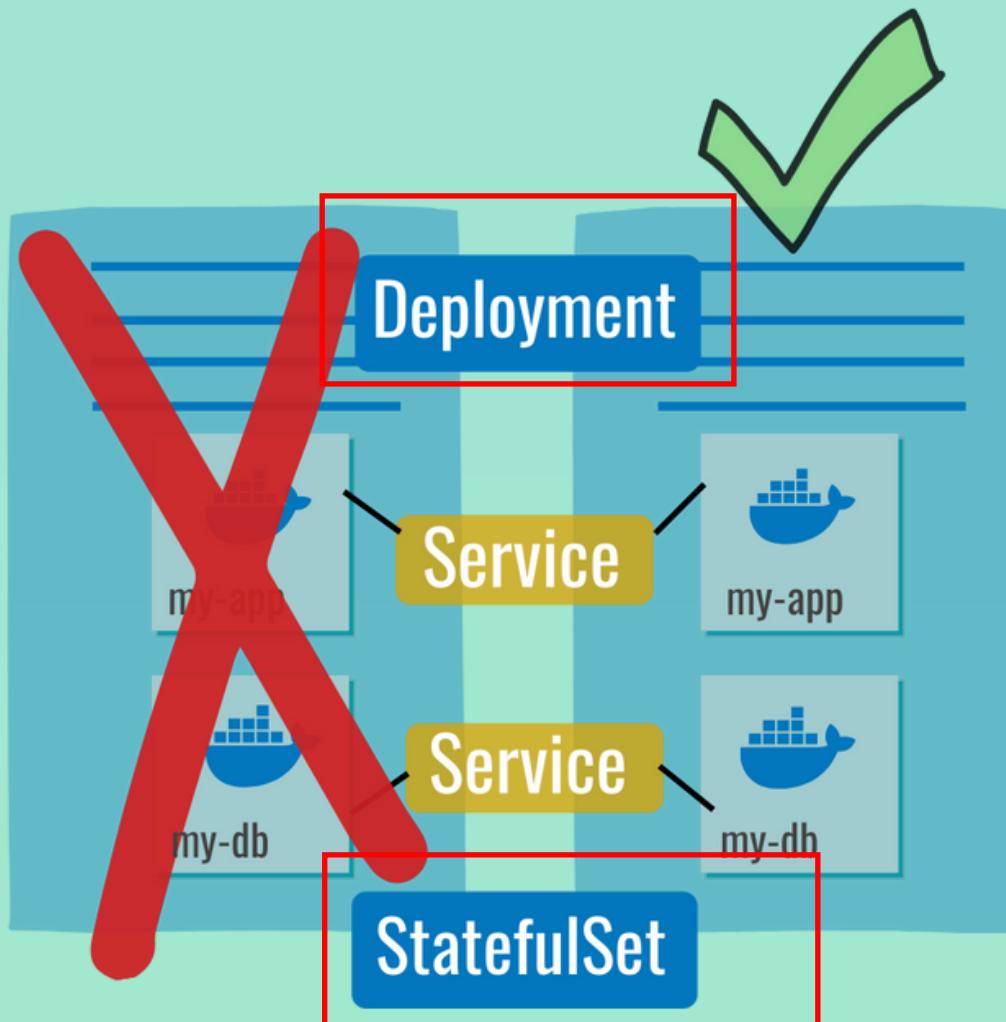
- Blueprint for Pods
- You work with Deployments and by defining the number of **replicas**, K8s creates Pods



Application

Abstraction of container is pod
abstraction of pods is deployment
Mostly works with deployment not pod

Database -> Statefulset application



STATEFULSET

- Blueprint for stateful applications
- Like databases etc
- In addition to **replicating features**, StatefulSet makes sure database reads and writes are synchronized to avoid data inconsistencies

Host database application outside of k8s cluster



Having load balanced replicas our setup is much **more robust**

Kubernetes Architecture

2 Types of Nodes

- A Kubernetes cluster consists of a set of worker machines, called "Nodes"

Worker Node

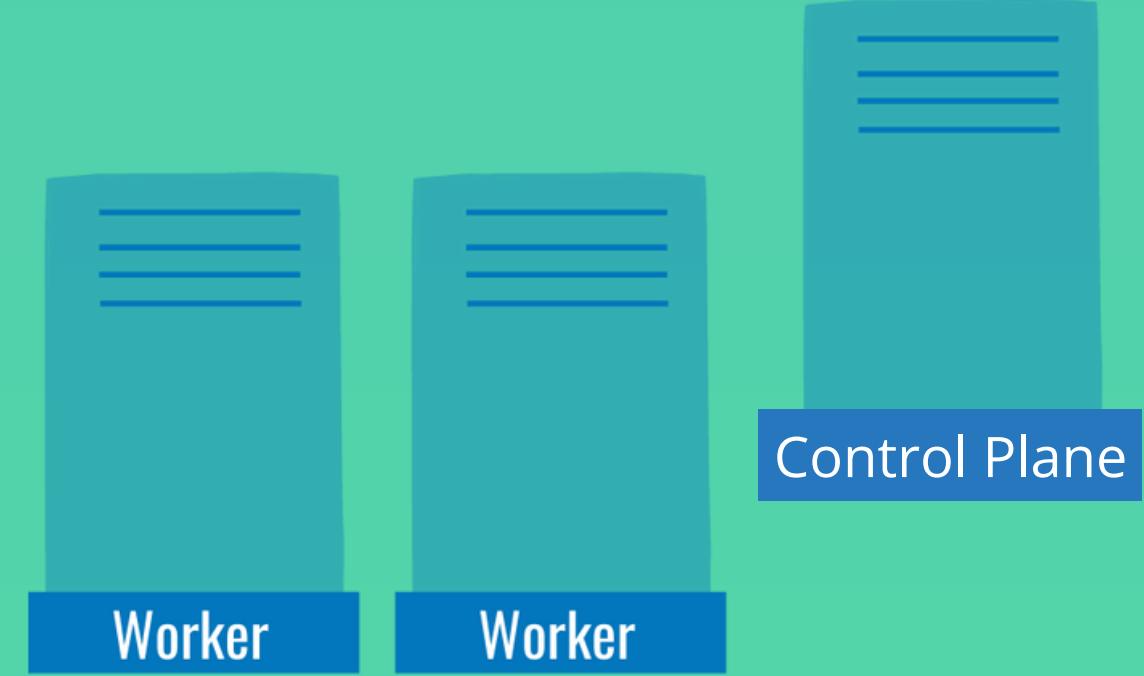
- The **containerized applications run on the Worker Nodes**
- Each Node runs multiple Pods on it
- Much more compute resources needed, because the actual workload runs on them

worker nodes do the actual work

Three processes must be installed on every node
- container runtime

Control Plane

- **Manages** the Worker Nodes and the Pods in the cluster
- Much more important and needs to be replicated
- So in production, replicas run across multiple machines



Worker Node Components

Each worker node needs to have 3 processes installed:

1) Container Runtime

- Software responsible for running containers
- For example containerd, CRI-O or Docker

2) Kubelet

- Agent that makes sure containers are running in a Pod
- Talks to underlying server (to get resources for Pod) and container runtime (to start containers in Pod)

Kubelet: interacts with container and nodes. Take the configuration and start the pod with a container inside and signing resources to the container

3) Kube-proxy

- A network proxy with intelligent forwarding of requests to the Pods

3) Kube-proxy

2) Kubelet



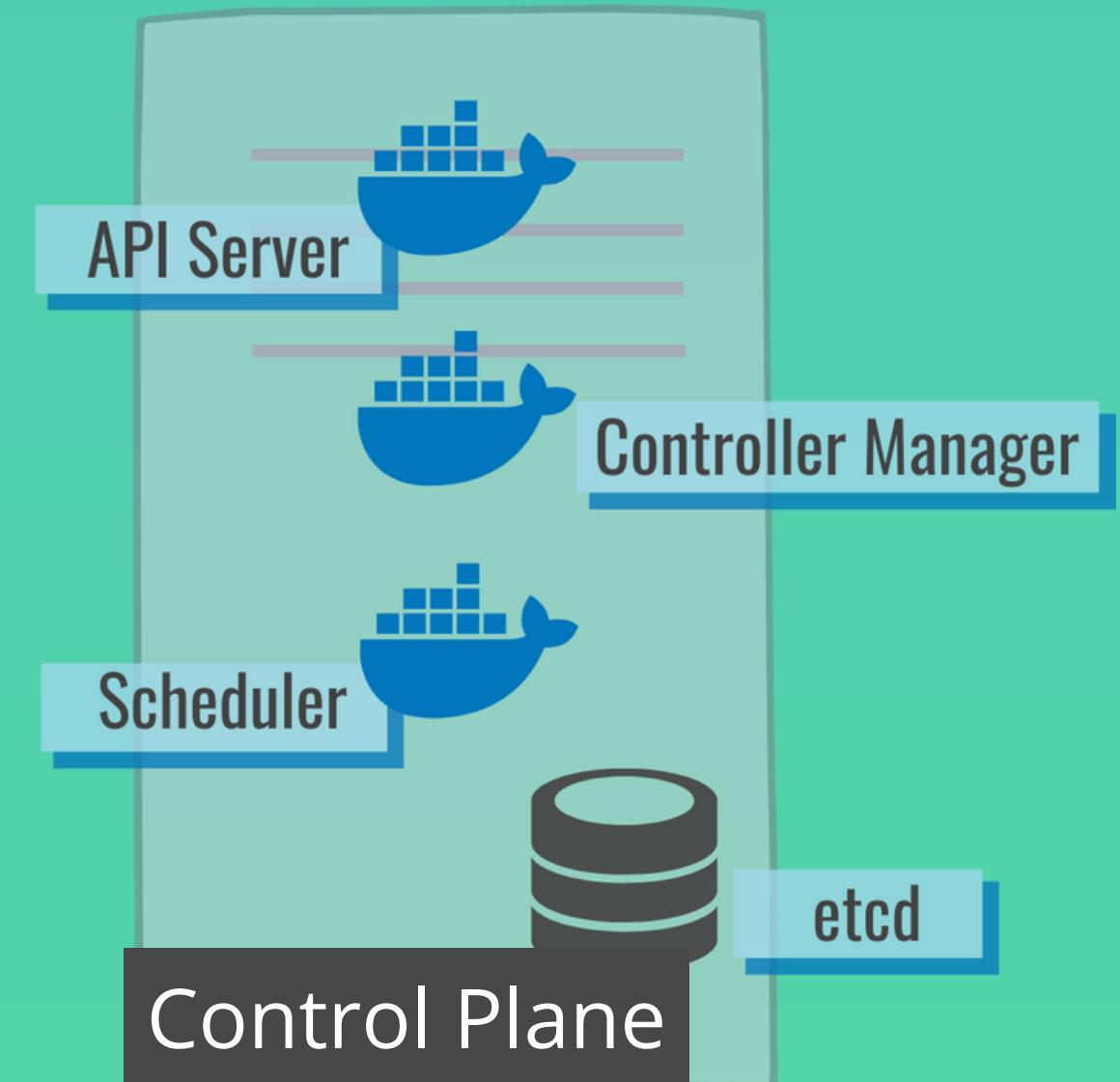
1) Worker

Control Plane Components - 1

- Each control plane needs to have **4 processes installed:**

- 1) API Server
- 2) Scheduler
- 3) Controller Manager
- 4) etcd

- Control Plane makes global decisions about the cluster
- Detects and responds to cluster events



APIservice: Client interact with APIServer (kubectl, UI) -> APIServer is like cluster gateway (update / query) / also acts as a gatekeeper for authentication / only one entry point to the cluster

Scheduler: deploy a pod -> APIServer -> Scheduler -> start a pod in node -> where to put a pod? -> kubectl(resource the pod need, cpu, go through worker nodes see where is least busy and has the resource) then determine where to locate the pod. and kubectl install the pod in the node.

Controller manager: pods died. detect pod is died and recovery the cluster asap. controller manager -> scheduler -> kubelet

etcd: key value store. cluster brain, all changes are saved and updated. All scheduler and controller manager can work because the etcd saved data. (Application data is not stored in the etcd)

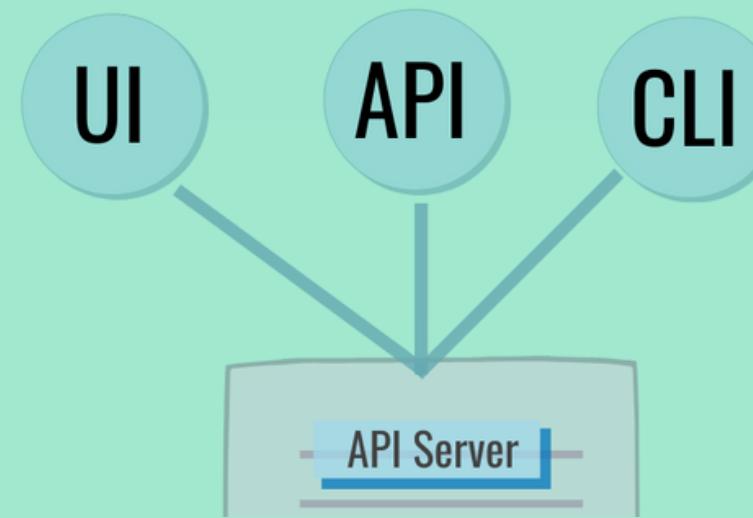
API server is load balanced

Distributed storage across all master nodes

Control Plane Components - 2

1) API Server

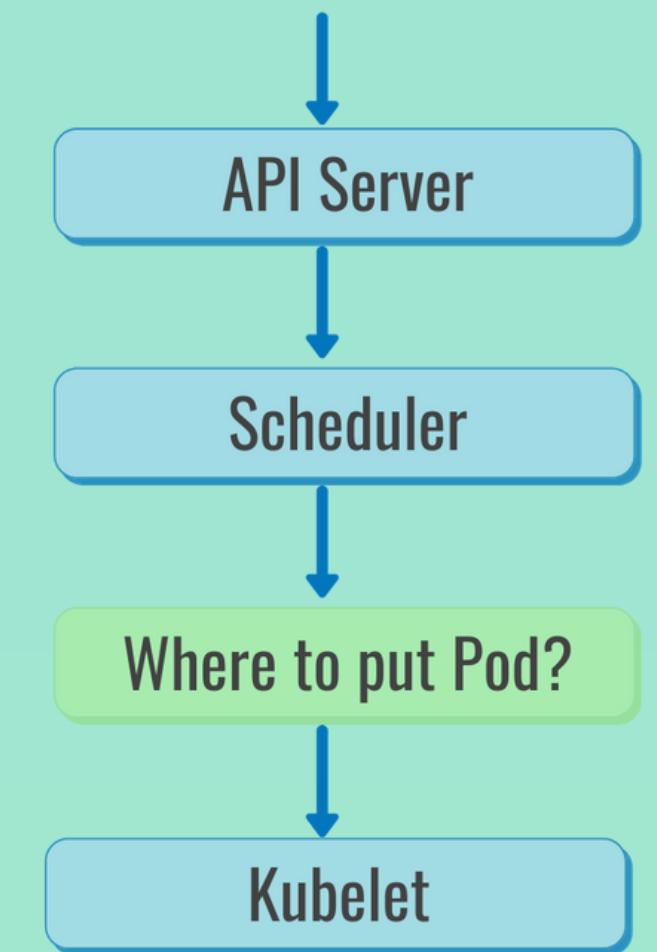
- The cluster gateway - **single entrypoint to the cluster**
- Acts as a gatekeeper for authentication, validating the request
- Clients to interact with the API server: UI, API or CLI



2) Scheduler

- Decides on which Node new Pod should be scheduled
- Factors taken into account for scheduling decisions: resource requirements, hardware/software/policy constraints, data locality, ...

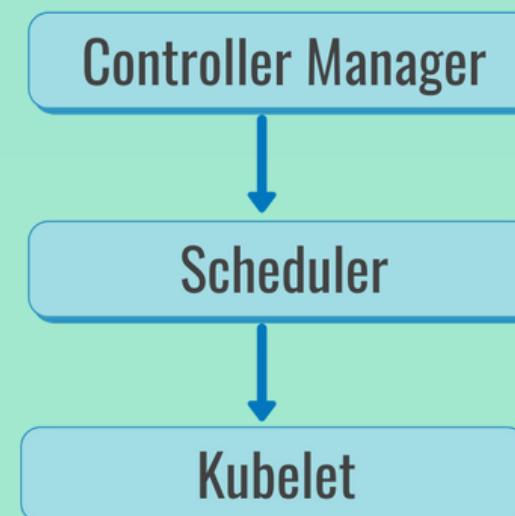
SCHEDULE NEW POD



Control Plane Components - 3

3) Controller Manager

- **Detects state changes**, like crashing of Pods and tries to recover the cluster state as soon as possible
- For that it makes **request to the Scheduler** to reschedule those Pods and the same cycle happens

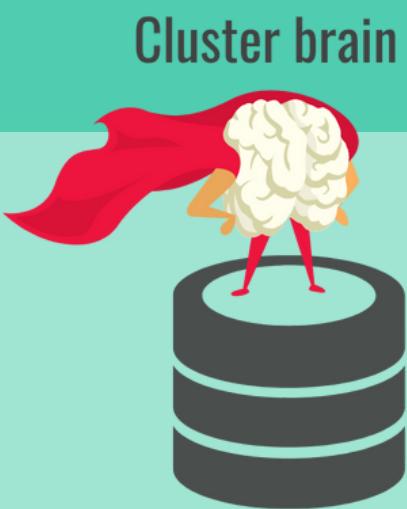


4) etcd

- K8s' backing **store for all cluster data**. A consistent, high-available key-value store
- Think of it as a cluster brain, every change in the cluster gets saved or updated into it
- **All other processes** like Scheduler, Controller Manager etc **work based on the data in etcd** as well as communicate with each other through etcd store



The **actual application data** is NOT stored in the etcd store



Increase Kubernetes Cluster Capacity

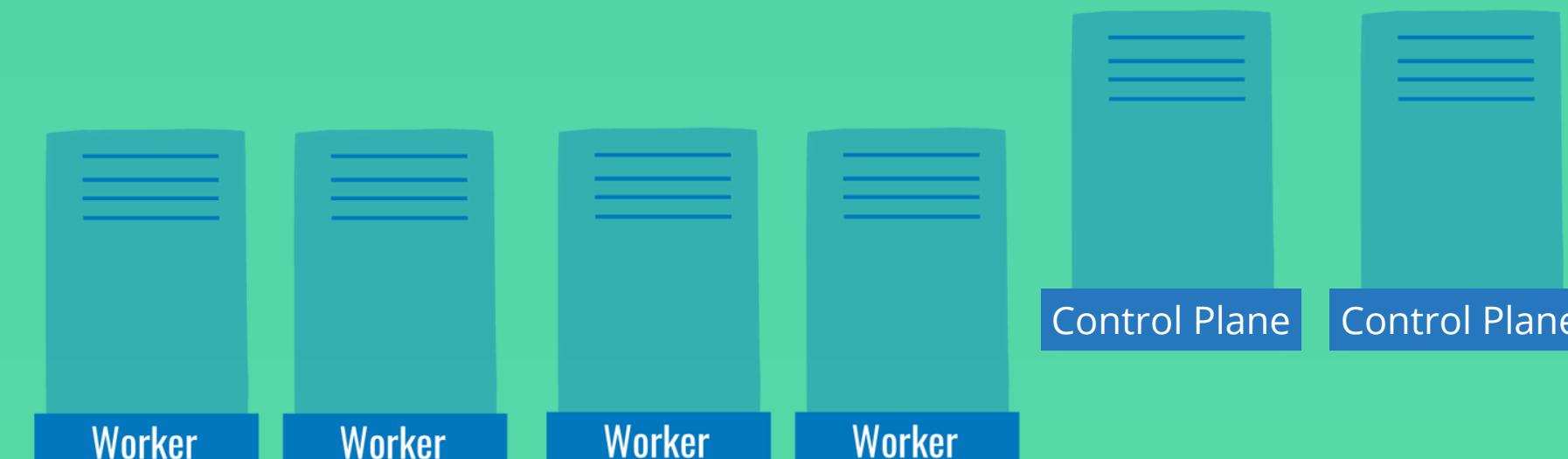
As your application grows and its demand for resources increases, you may actually **add more Nodes** to your cluster, thus forming a more powerful and robust cluster to meet your application resource requirements

Add a Control Plane Node

1. Get a fresh new server
2. Install all control plane processes on it
3. Join it to the K8s cluster using a K8s command

Add a Worker Node

1. Get a fresh new server
2. Install all the Worker Node processes, like container runtime, Kubelet and KubeProxy on it
3. Join it to the K8s cluster using a K8s command



Deep Dive into Kubernetes

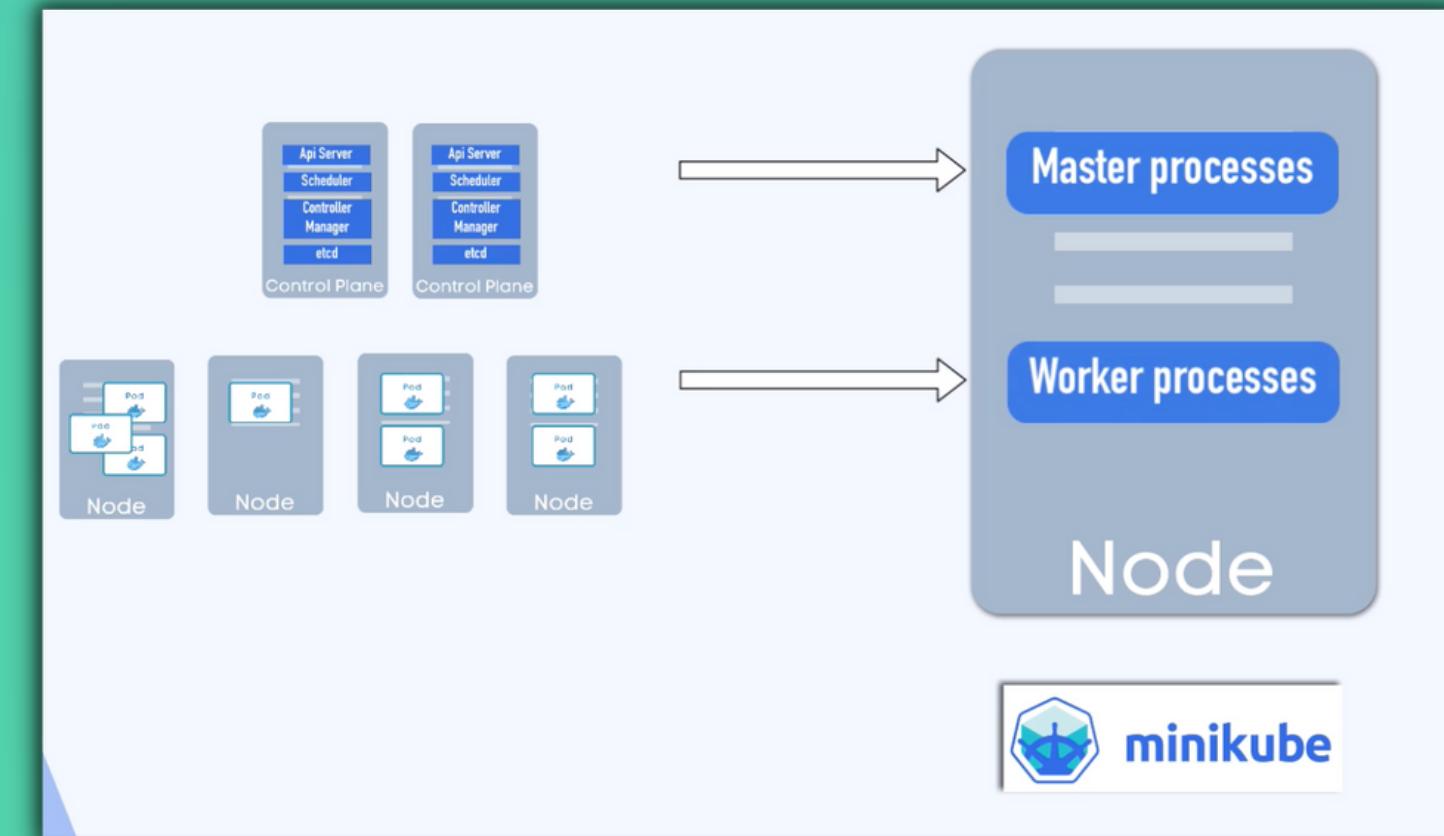
Minikube - Local Cluster Setup

- Minikube implements a **local** K8s cluster
- Useful for local K8s application development,
because running a test cluster would be complex

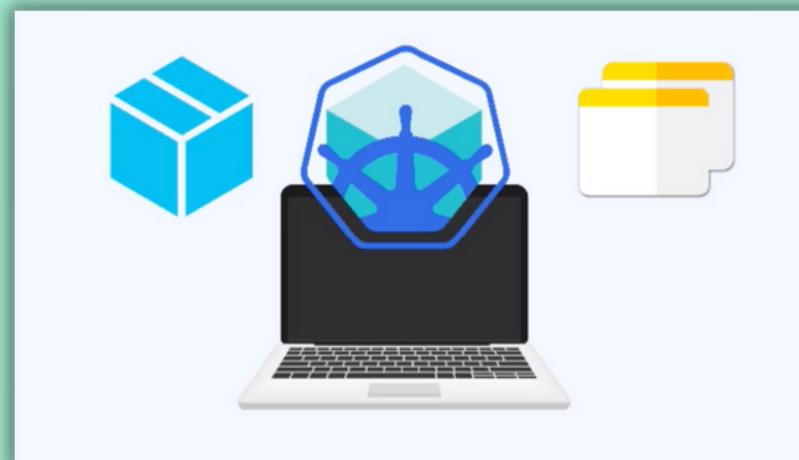


Control Plane and Worker

processes run on **ONE machine**



Run Minikube either as a **container** or **virtual machine** on your laptop



- So you need a container runtime or virtual
machine manager on your laptop

Kubectl - Command Line Tool

- CLI Tool to **interact with your K8s cluster** (minikube or cloud cluster...)
- In order for kubectl to access a K8s cluster, it needs a **kubeconfig file**, which is created automatically when deploying your minikube cluster
- By default, config file is located at **~/.kube/config**



kubectl

API server

Basic kubectl commands:

```
kubectl get {k8s-component}  
kubectl get nodes  
kubectl get pods  
kubectl get services  
kubectl get deployment
```

Get status of different components

```
kubectl create {k8s-component} {name} {options}  
kubectl create deployment my-nginx-depl --image=nginx
```

CRUD

```
kubectl edit {k8s-component} {name}  
kubectl delete {k8s-component} {name}
```

```
kubectl logs {pod-name}  
kubectl describe {pod-name}
```

Debugging

```
kubectl exec -it {pod-name} -- bash  
kubectl apply -f config-file.yaml
```

K8s YAML Configuration File - 1

- Also called "Kubernetes manifest"
- Declarative: A manifest **specifies the desired state** of a K8s component
- Config files are in **YAML format**, which is user-friendly, but **strict indentation!** [Use yaml online validator](#)
- Config files should be stored in version control

Each configuration file has **3 parts**:

1) **metadata**

```
! nginx-deployment.yaml ✘
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  labels:...
6
7  spec:
8    replicas: 2
9    selector:...
10   template:...
```

```
! nginx-service.yaml ✘
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5
6  spec:
7    selector:...
8    ports:...
```

2) **specification**

- Attributes of "spec" are specific to the kind

```
! nginx-deployment.yaml ✘
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  labels:...
6
7  spec:
8    replicas: 2
9    selector:...
10   template:...
```

```
! nginx-service.yaml ✘
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5
6  spec:
7    selector:...
8    ports:...
```

Attributes of "spec" are specific to the kind!
store configuration file in the application code
infracsture as code concpet

K8s YAML Configuration File - 2

Current status: 1 replica

```
status:  
  availableReplicas: 1  
  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-7d64f4b"  
      reason: NewReplicaSetAvailable  
      status: "True"  
      type: Progressing  
  observedGeneration: 1  
  readyReplicas: 1  
  replicas: 1  
  updatedReplicas: 1
```



3) status

- **Automatically generated** and added by Kubernetes
- K8s gets this information from **etcd**, which holds the current status of any K8s component

Desired status: 2 replica

```
! nginx-deployment.yaml ✘  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: nginx-deployment  
5  +   labels: ...  
7  spec:  
8    replicas: 2  
9  +   selector: ...  
12 +   template: ...  
22
```

self healing: when the statuses are not identiy

Deployment Configuration File

- Deployment Configuration is a bit special

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  >  labels: ...
7  spec:
8    replicas: 2
9  >  selector: ...
12 template:
13   metadata: [red box]
14     labels:
15       app: nginx
16   spec: [red box]
17     containers:
18       - name: nginx
19         image: nginx:1.16
20         ports:
21           - containerPort: 8080
```

- Since it's an abstraction over Pod, we have
 - the **Pod configuration inside Deployment configuration**
- Own "metadata" and "spec" section
- Blueprint for Pod

Labels & Selectors

Labels

- Labels are **key/value pairs** that are attached to resources, such as Pods.
- Used to **specify identifying attributes** that are meaningful and relevant to users

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment
5   labels:
6     app: nginx
7 spec:
8   replicas: 2
9   selector:
10    matchLabels:
11      app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
```

Label Selectors

- Labels do not provide uniqueness
- Via **selector** the user can identify a set of resources

Connecting Services to Deployments

```
service
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nginx-service
5 spec:
6   selector:
7     app: nginx
8   ports: ...
```

where service is mit welchem Pod verbinden

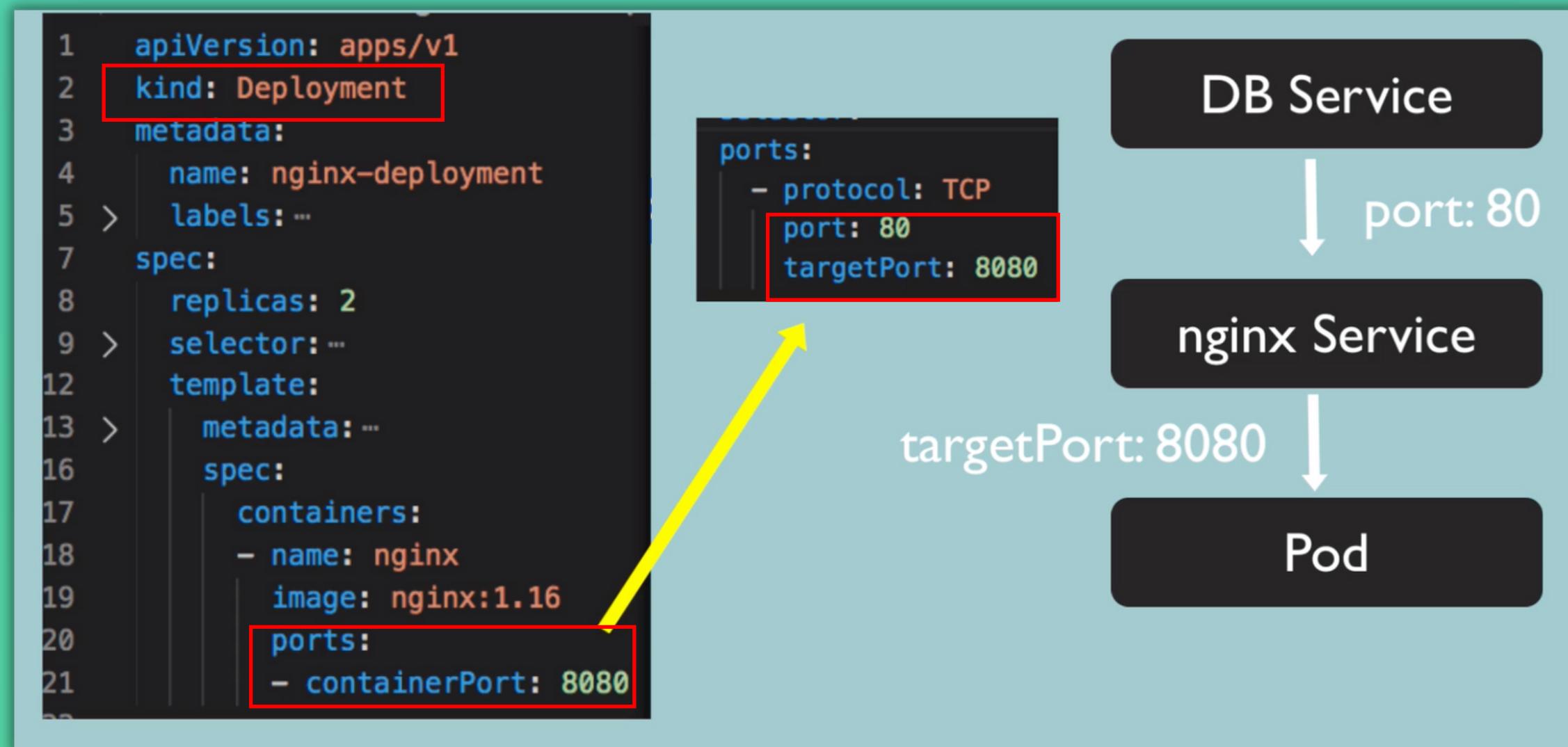
Ports in Service and Pod

- In Service component you need to specify:

- **port** = the port where the service itself is accessible
- **targetPort** = port, the container accepts traffic on

- In Deployment component:

- **containerPort** = port, the container accepts traffic on



Browser Request Flow through the K8s components

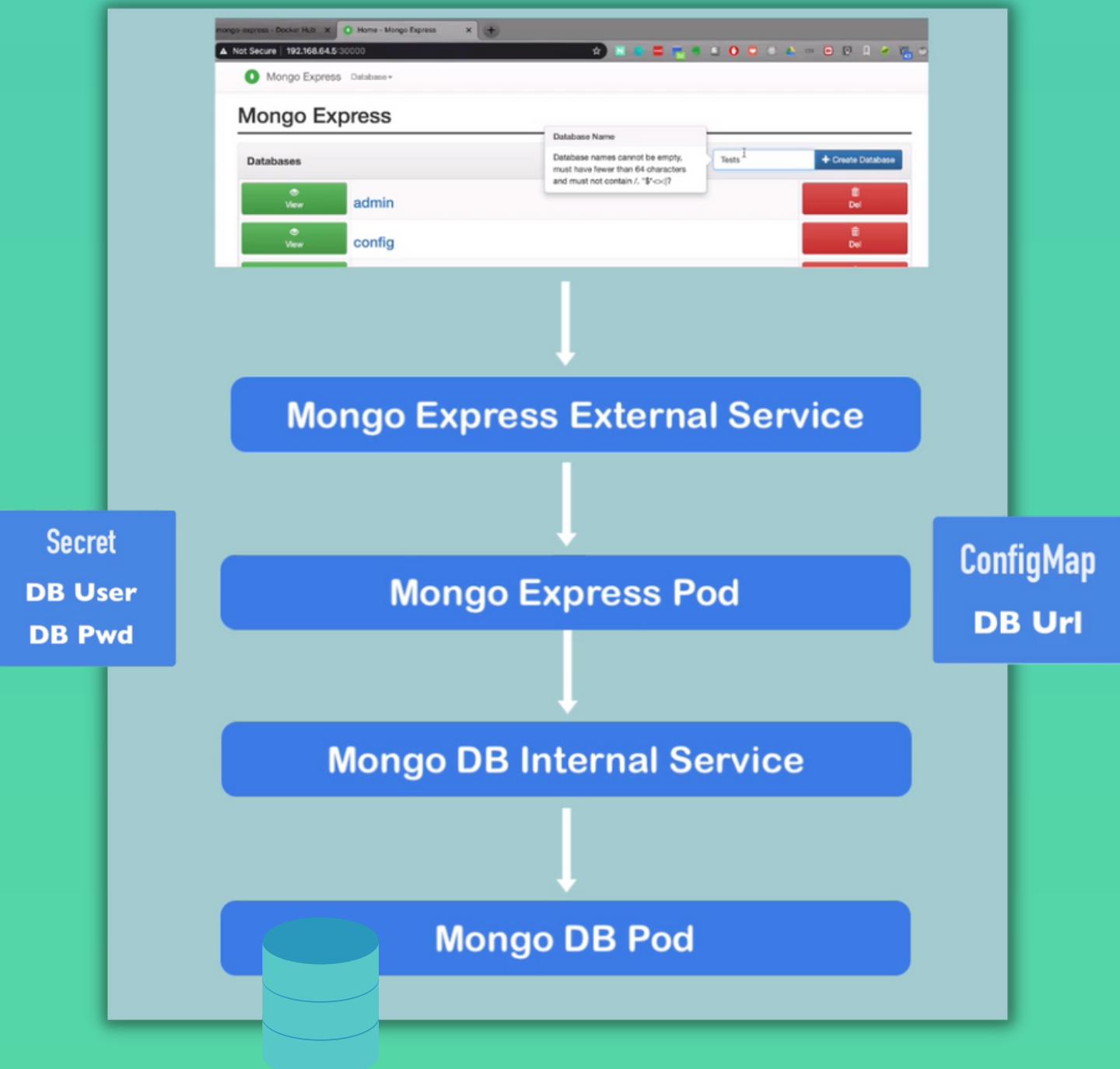
Example Setup:

- Mongo Express as UI
- MongoDB as database
- User updates entries in database via browser
- ConfigMap and Secret holds the MongoDB's endpoint (Service name of MongoDB) and credentials (user, pwd), which gets injected to MongoExpress Pod, so MongoExpress can connect to the DB

K8s Components needed in this setup

- 2 Deployment / Pod
- 2 Services
- 1 ConfigMap
- 1 Secret

URL: IP address of Node + Port of external Service



Namespaces - 1

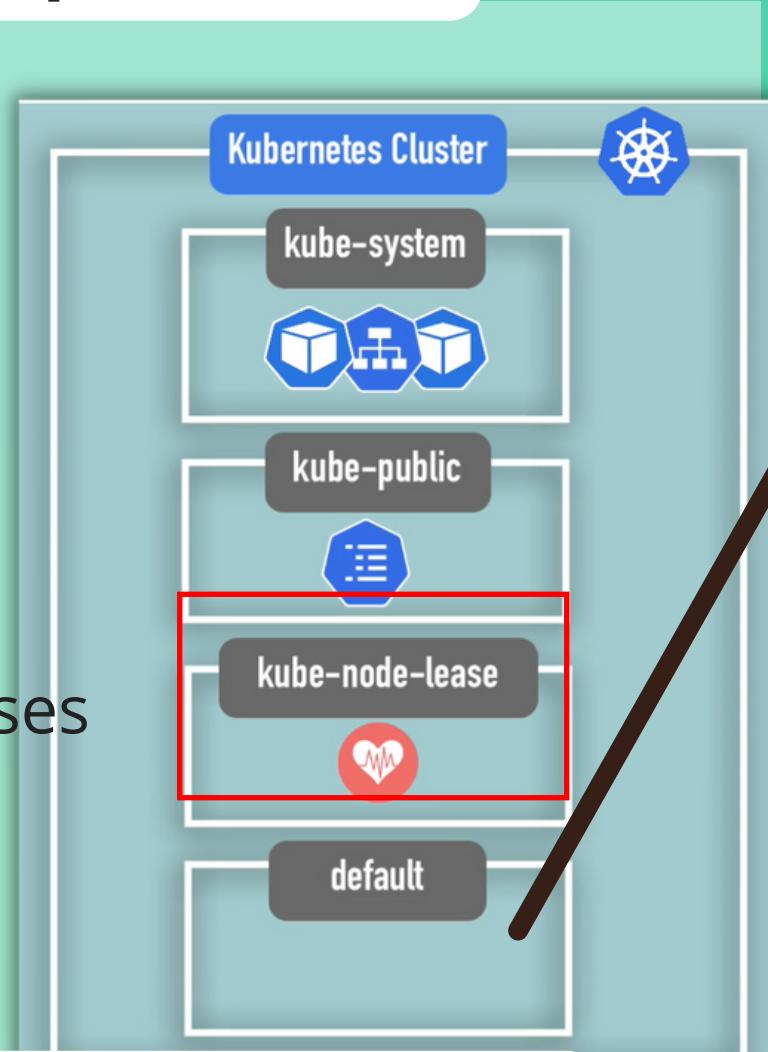
- Namespaces provide a **mechanism for isolating groups of resources** within a single cluster
- Names of resources need to be unique within a namespace, but not across namespaces
- Like a virtual cluster inside a cluster

Namespaces per default⁴

- Namespaces per default, when you install K8s
- "kube-system" has control plane processes running



DON'T modify kube-system!!



"Default" namespace

- Start deploying your application in the default namespace called "default"

Create a new namespace

- Via **kubectl** command:
- Via **configuration file**:

kubectl create namespace my-ns



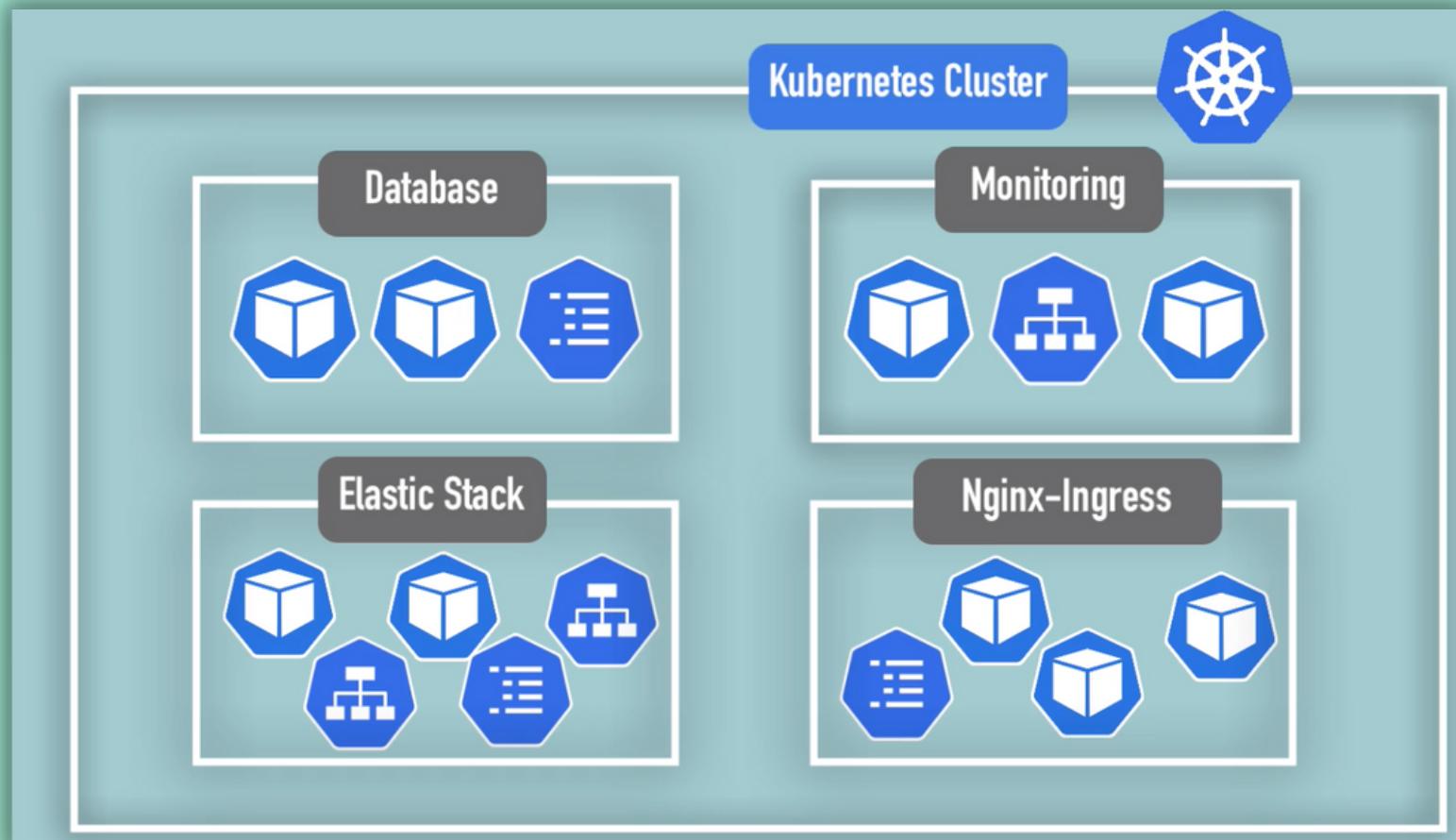
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-configmap
  namespace: my-namespace
data:
  db_url: mysql-service.database
```

Namespaces - 2

Use Cases for when to use namespaces:

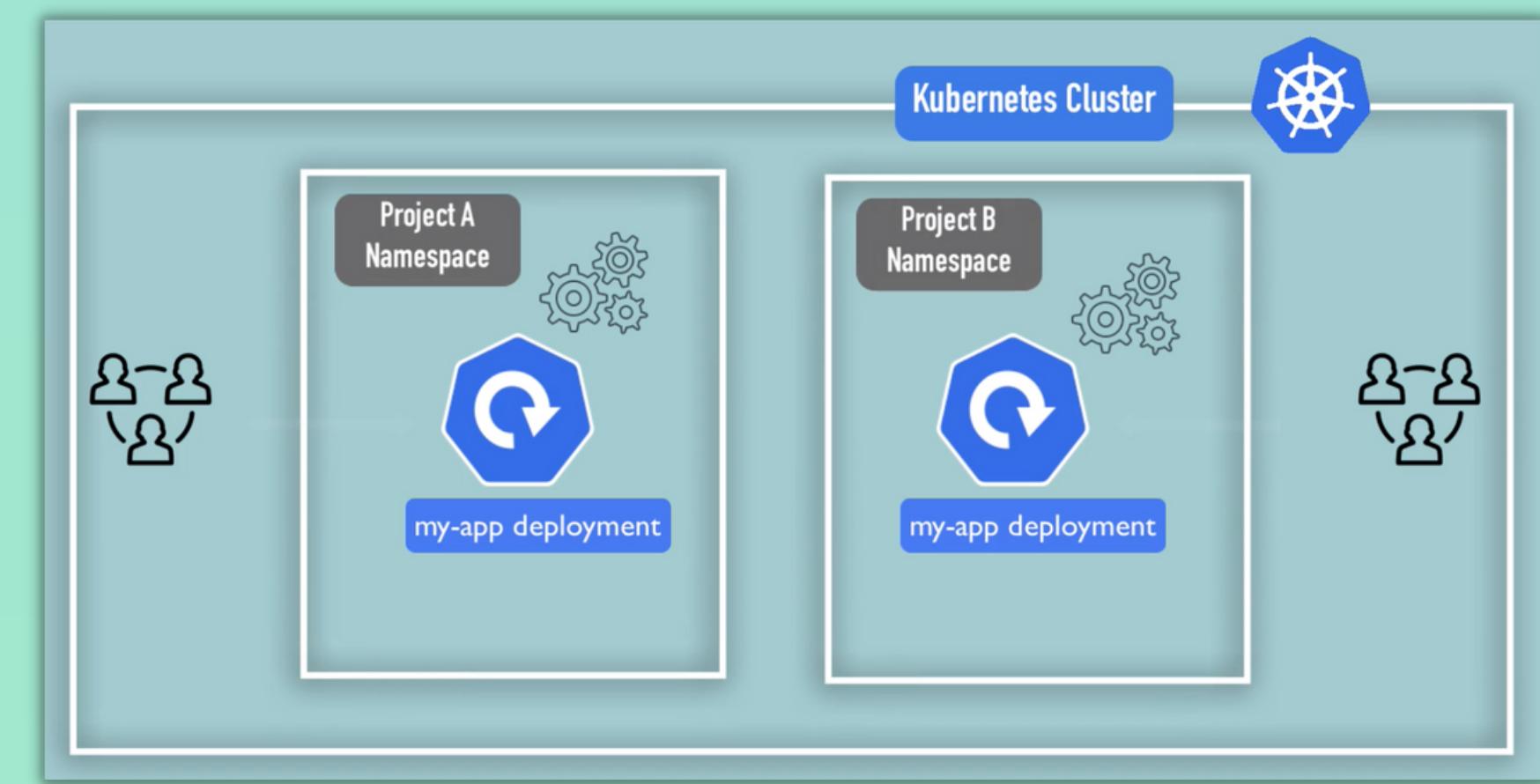
1 - Group resources logically

- Instead of having all in the "default" namespace



2 - Isolate team resources

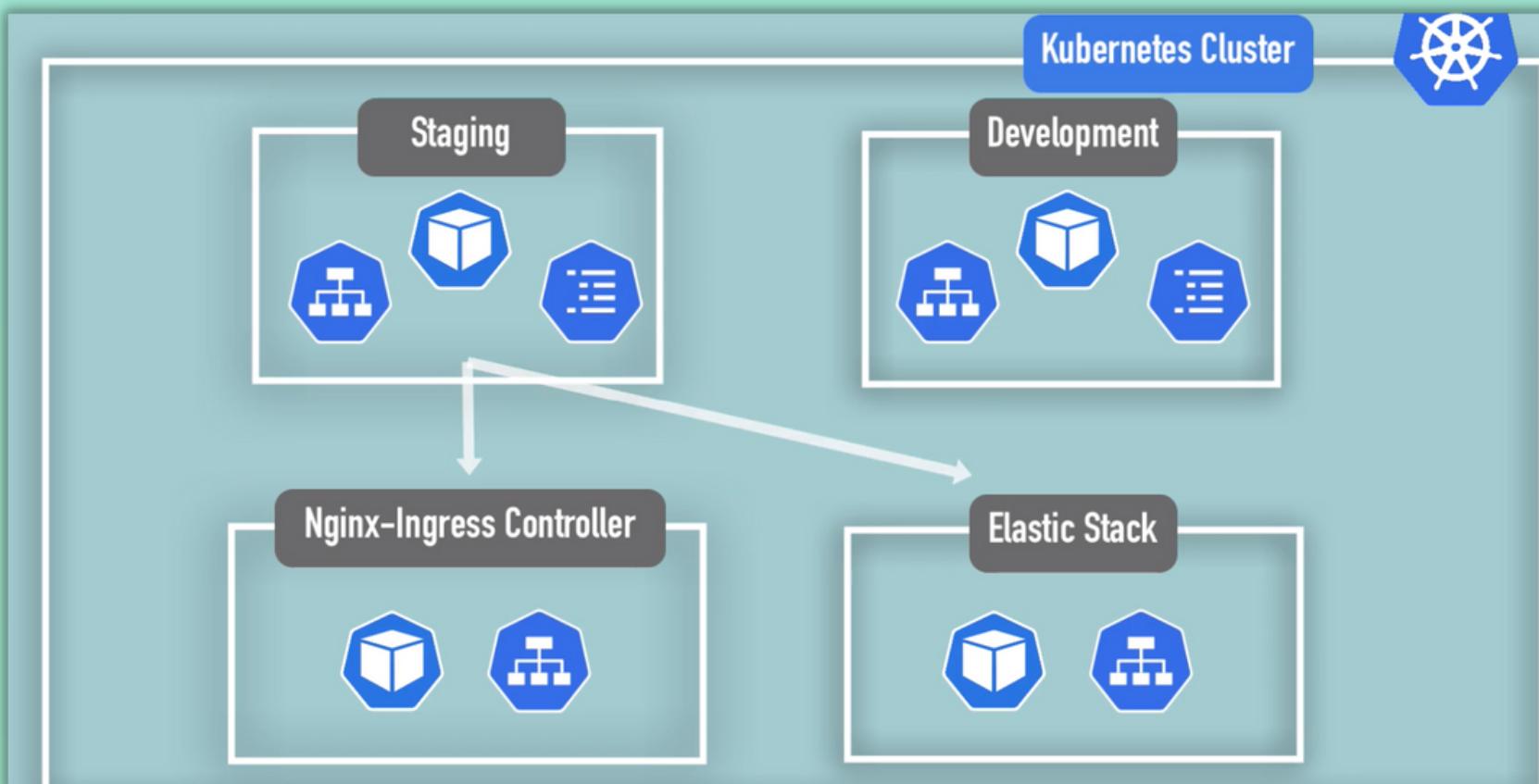
- To avoid conflicts



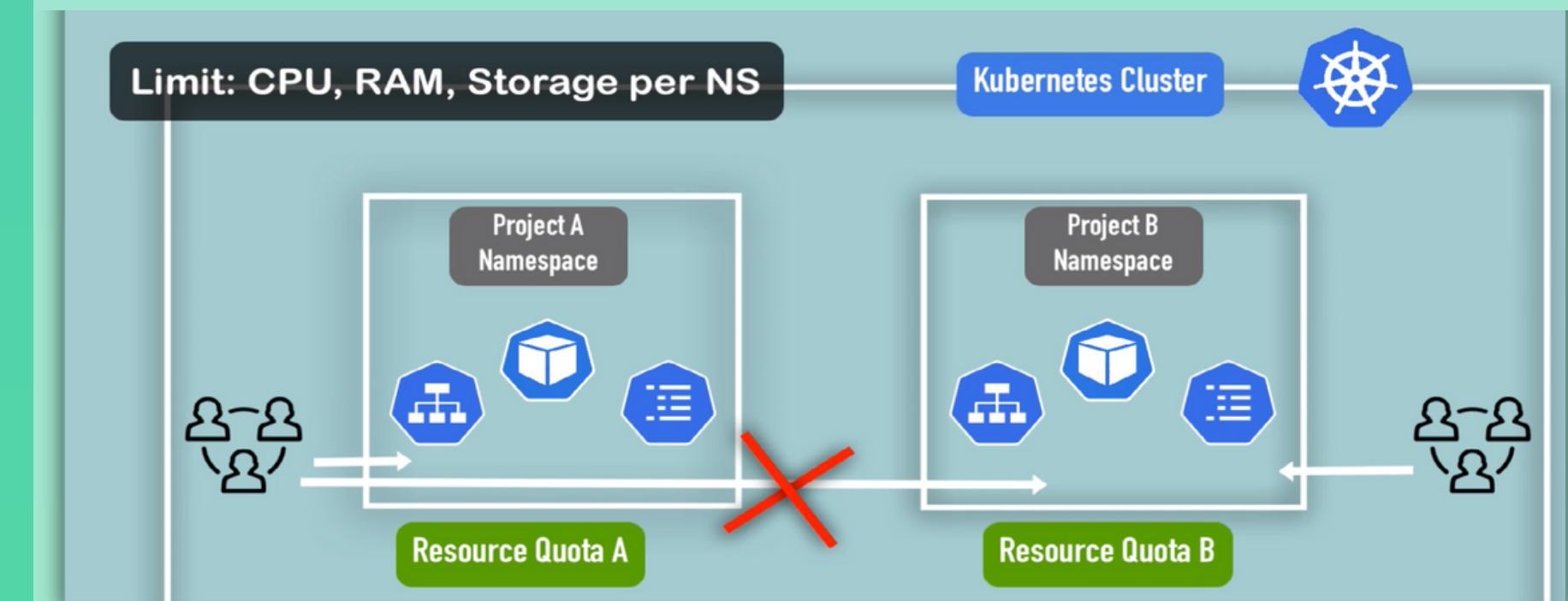
Namespaces - 3

Use Cases for when to use namespaces:

3 - Share resources between different environments



4 - Limit permissions and compute resources for teams

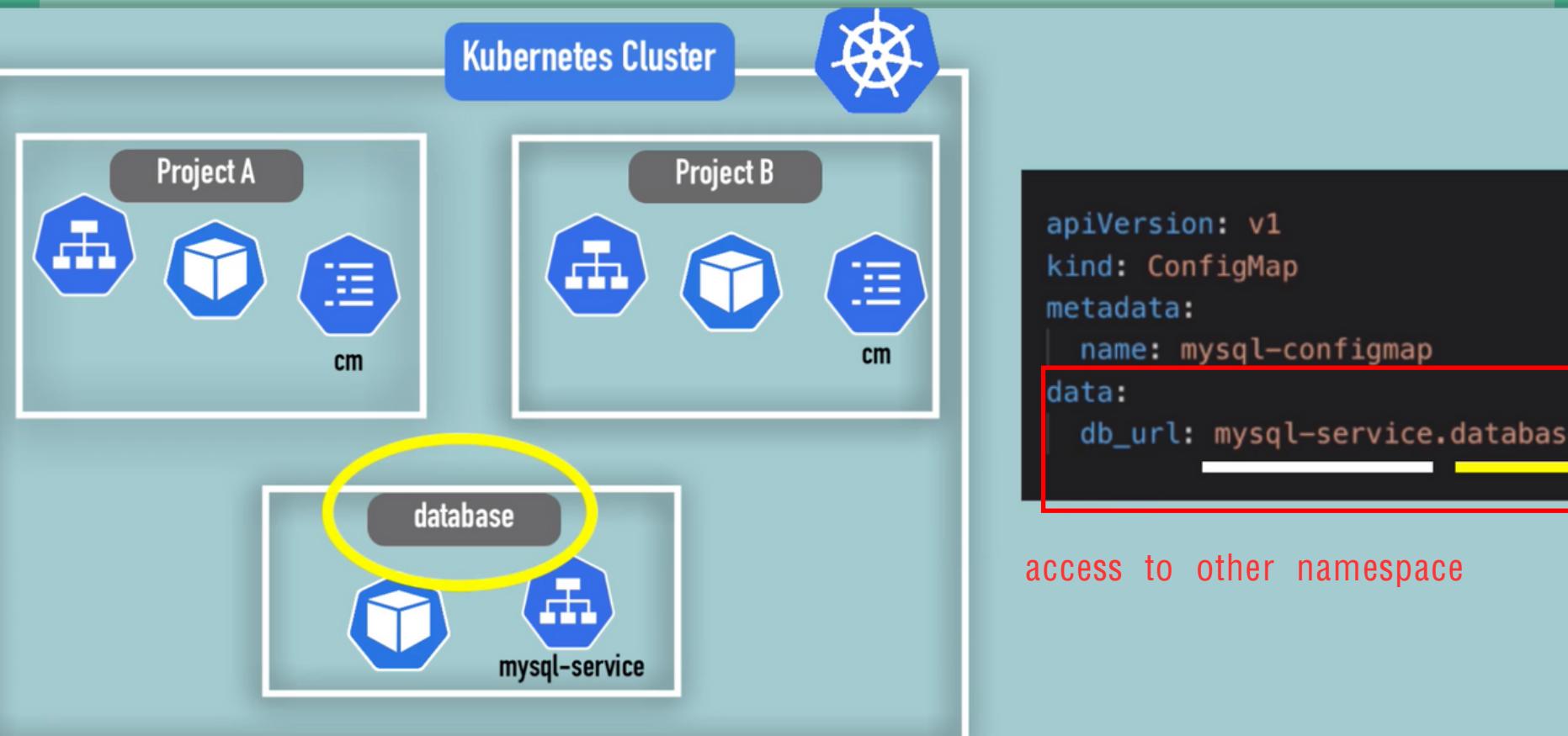


Namespaces - 4

- There are resources, which can't be created within a namespace, called cluster-wide resources:

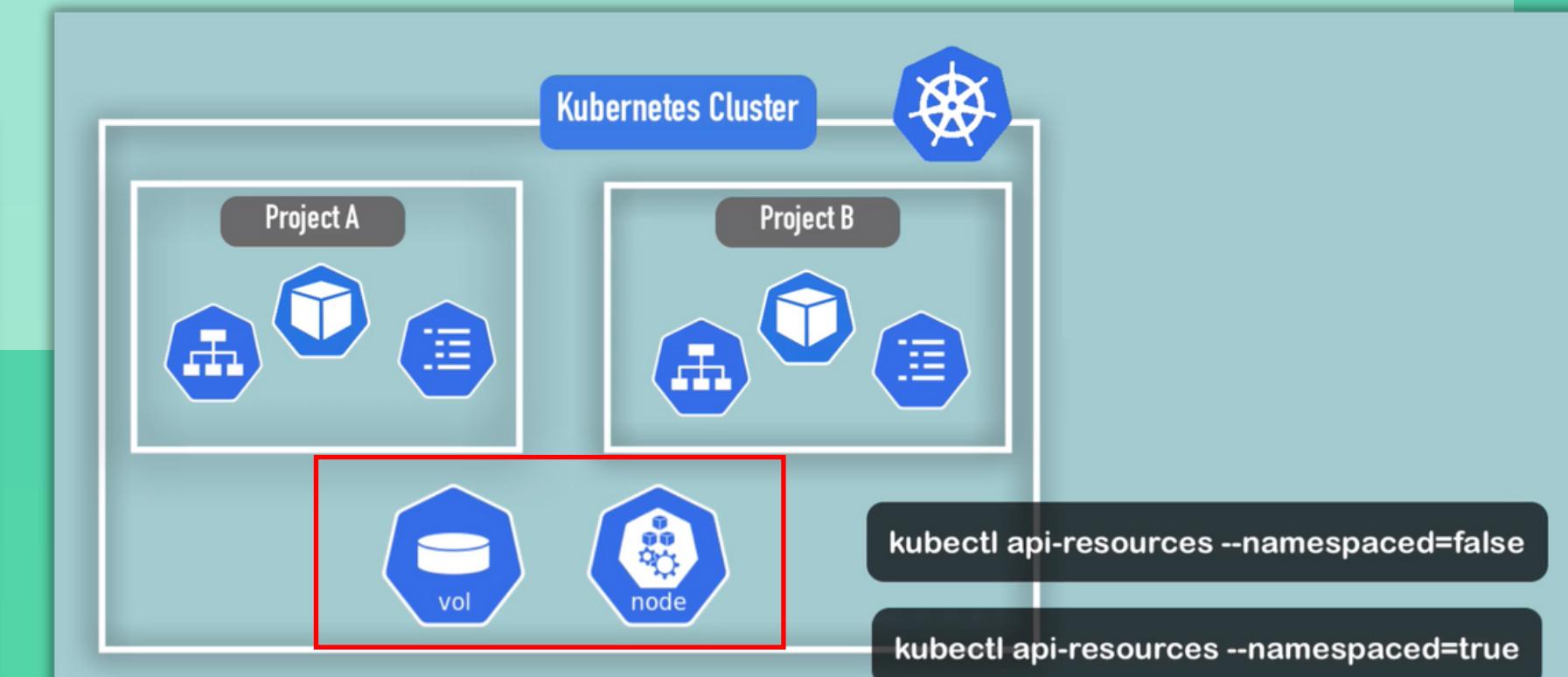
Namespaced resources

- Most K8s resources (e.g. pods, services, etc.) are in some namespaces
- Access service in another namespace:



Cluster-wide resources

- Live globally in a cluster, you can't isolate them
- Low-level resources, like Volumes, Nodes



deploy and same time create a namespace

kubectl apply -f xx.yaml --namespace=my-namespace

Inside the configuration file in the metadata -> namespace:

!!! before we use: kubectl get configmap

!!!! now we get a configmap from a namespace: kubectl get configmap -n my-namespace

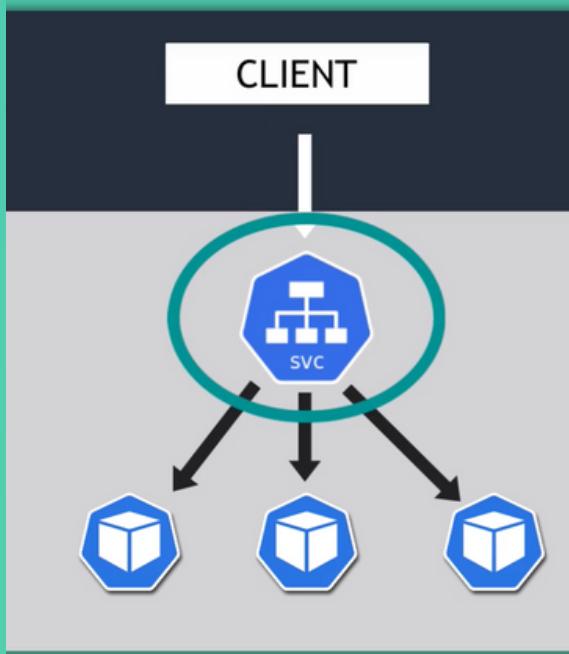
CHANGE ACTIVE NAMESPACE using kubens! i

Deep Dive into Kubernetes Services

Kubernetes Services - 1

- Abstract way to expose an application running on a set of Pods
- Different types of services:

3 Service type attributes



Why Service?

- ✓ Stable IP address
- ✓ Loadbalancing
- ✓ Loose coupling
- ✓ Within & Outside Cluster

ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
```

NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
```

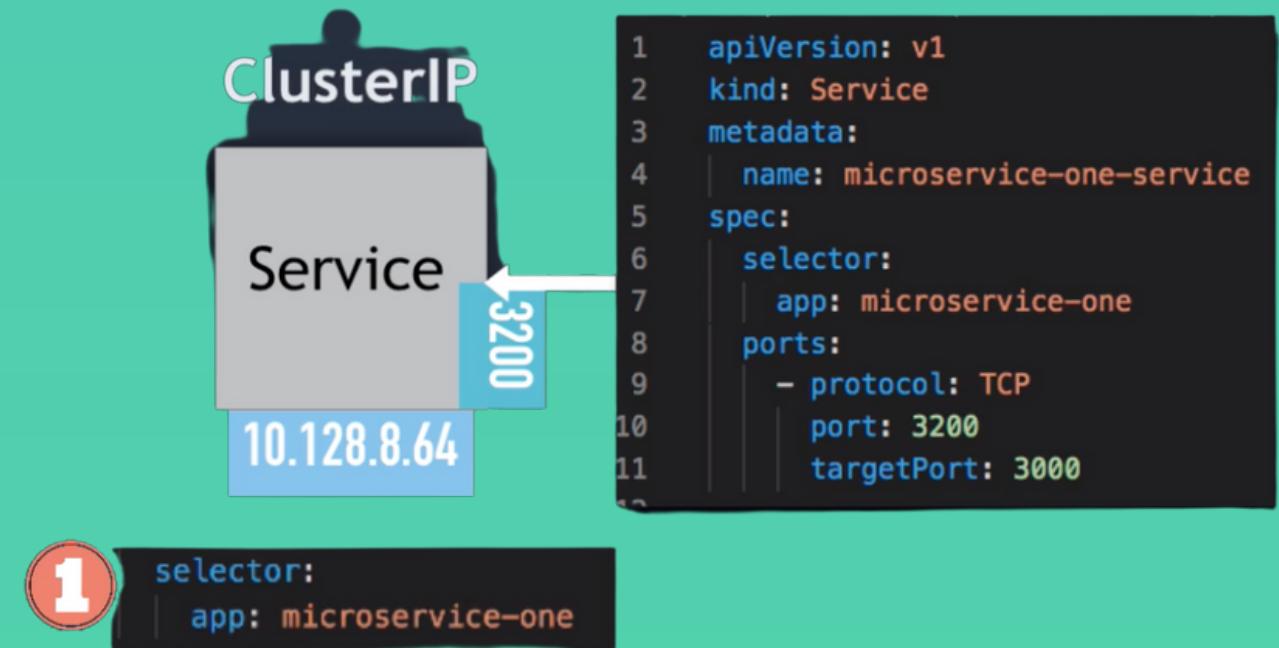
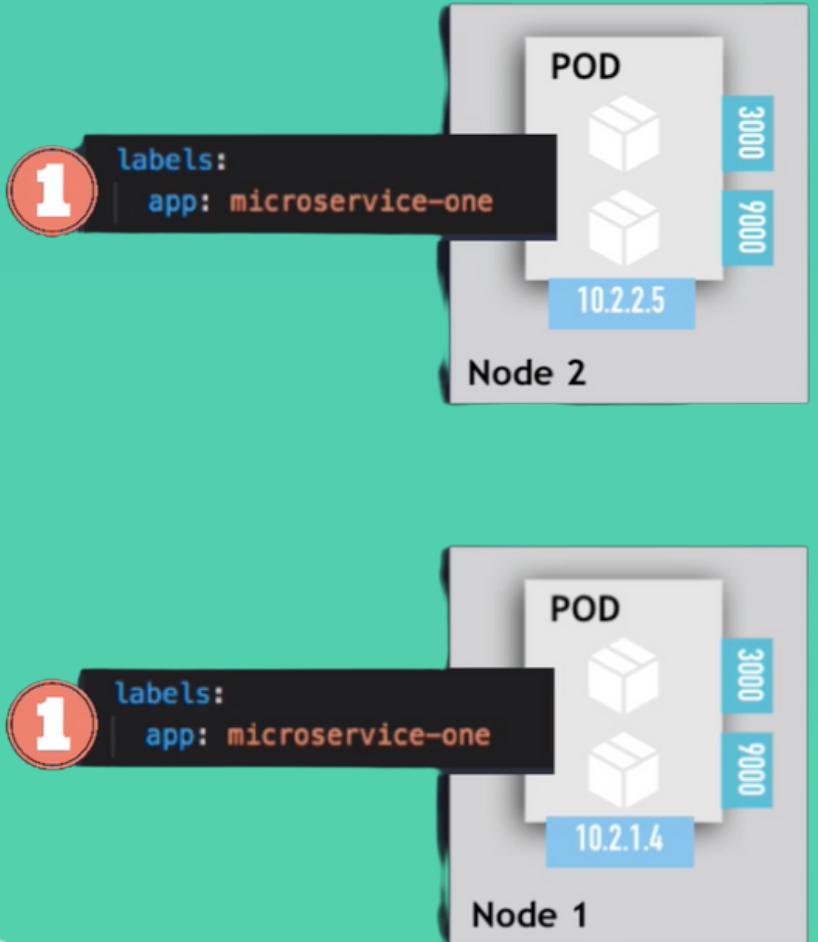
LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
```

- ClusterIP is the **default type**, when you don't specify a type

Kubernetes Services - 2

- Service defines a logical set of Pods
- The set of Pods targeted by a Service is **determined by a selector**



This creates a new Service named "miroservice-one-service", which **targets port 3000 on any Pod with the *app=microservice-one* label.**

A pod on Node 2 is shown with two containers, each listening on ports 3000 and 9000. An arrow points from the "targetPort: 3000" entry in the service's ports section to the "port: 3000" entry in the pod's containers section.

```
apiVersion: v1
kind: Service
metadata:
  name: microservice-one-service
spec:
  selector:
    app: microservice-one
  ports:
    - protocol: TCP
      port: 3200
      targetPort: 3000
```

PORTS:

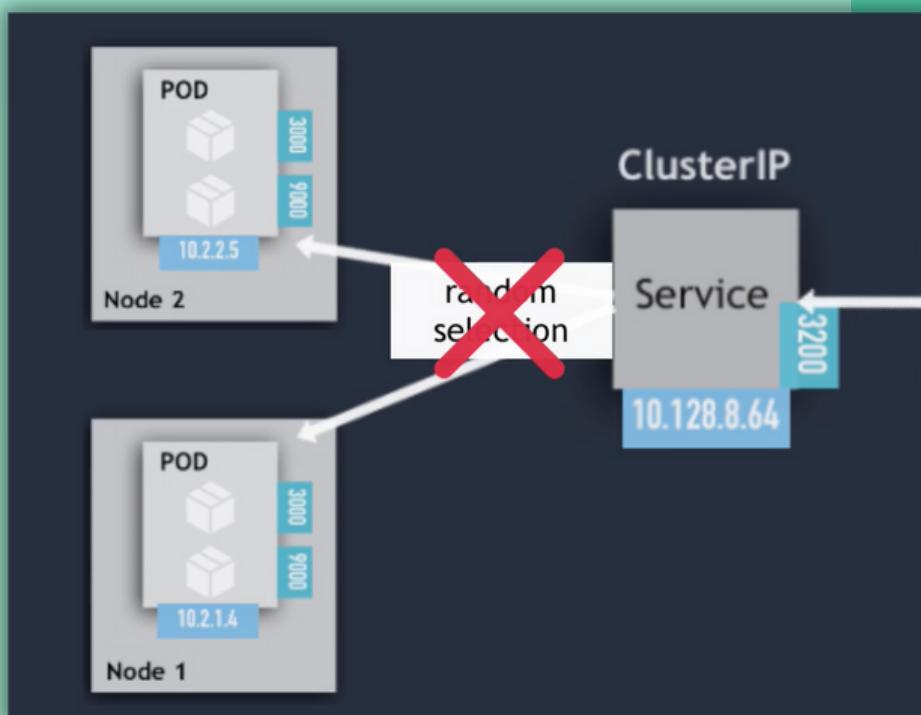
- The service port is arbitrary
- targetPort must **match the port the container is listening at**

ClusterIP Service & its subtypes

- ClusterIP is an internal service, not accessible from outside the cluster
- All Pods in the cluster can talk to this internal service

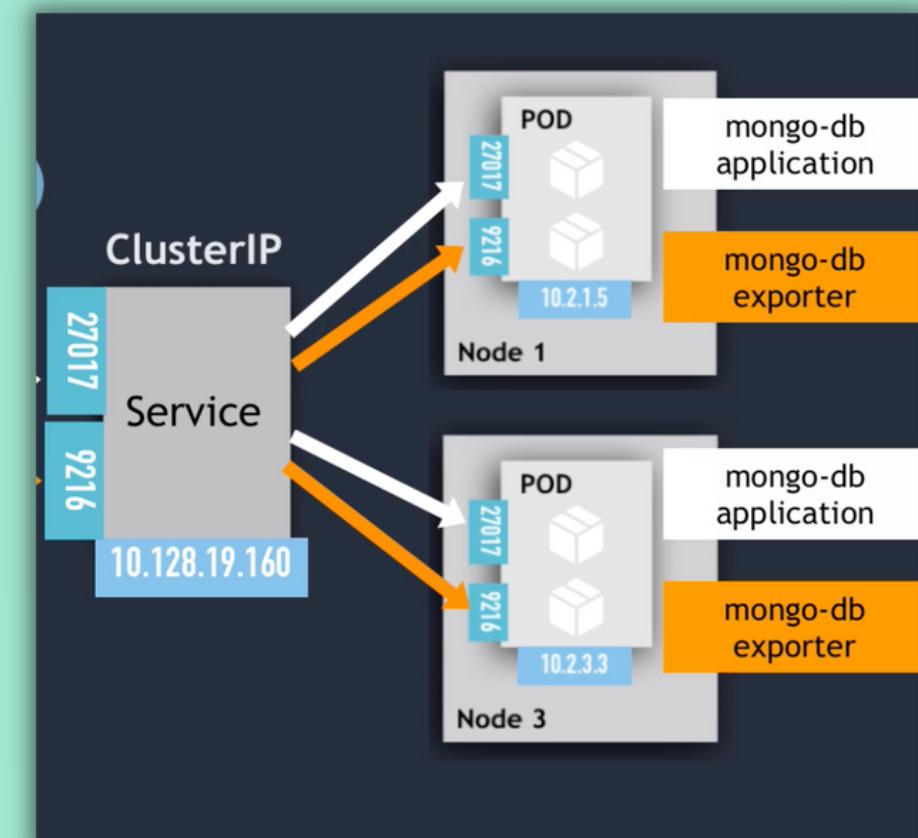
Headless Internal Service

- When client needs to **communicate with 1 specific Pod directly, instead of randomly selected**
- Use Case: When Pod **replicas are not identical**. For example stateful apps, like when only master is allowed to write to database

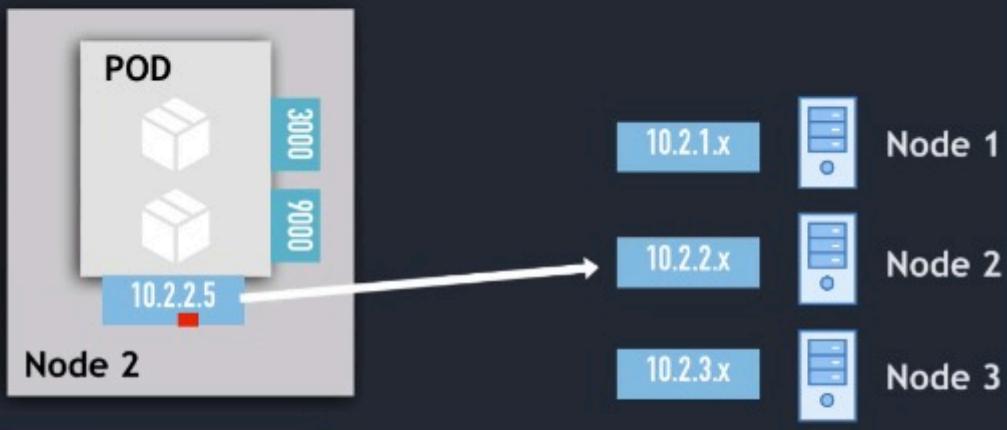


Multi-Port Internal Service

- When you need to expose more than 1 port
- K8s lets you **configure multiple port definitions on a Service**
- In that case, you must give all of your ports names so that these are unambiguous



ClusterIP Services



```
kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
mongodbs-deployment-55577c4cbc-gkz8b > <none>	1/1	Running	0	29s	10.2.2.2
mongodbs-deployment-55577c4cbc-z828l > <none>	1/1	Running	0	29s	10.2.1.4
mongodbs-deployment-55577c4cbc-zv9h8 > <none>	1/1	Running	0	29s	10.2.2.3

ess.yaml x

```
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-clientIPLoadBalancer

```
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
```

Service Communication: selector

which Pods to forward the request to?

which port to forward it to?

pod with multiple ports: -> service -> targetport
decide which pod receivevice

Service: selector attribute

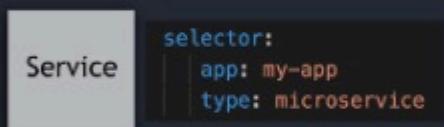
```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: microservice-one-service
5 spec:
6   selector:
7     app: microservice-one
8 ...
9
```

Pod: spec->template -> metadata-> labels:

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: microservice-one
5 ...
6 spec:
7   replicas: 2
8 ...
9 template:
0   metadata:
1     labels:
2       app: microservice-one
```

Service Communication: selector

- ▶ Svc matches all 3 replica
- ▶ registers as Endpoints
- ▶ must match ALL the selectors



Serve endpoints

Service Endpoints

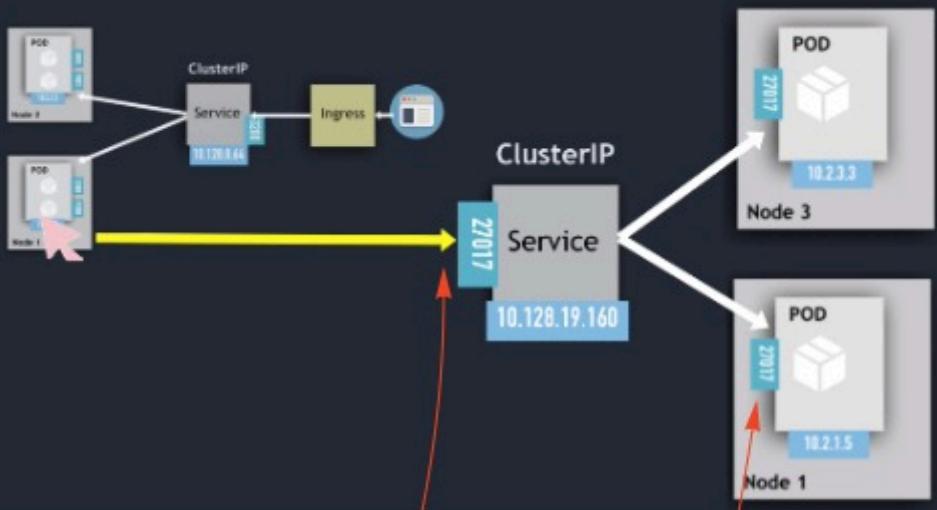
```
[k8s-services]$ kubectl get endpoints
NAME           ENDPOINTS             AGE
kubernetes     172.104.231.137:6443   15m
mongodb-service 10.2.1.4:27017,10.2.1.5:27017 5m27s
```

K8s creates Endpoint object

- same name as Service

- keeps track of, which Pods
are the members/endpoints of the Service

Service Communication: Example

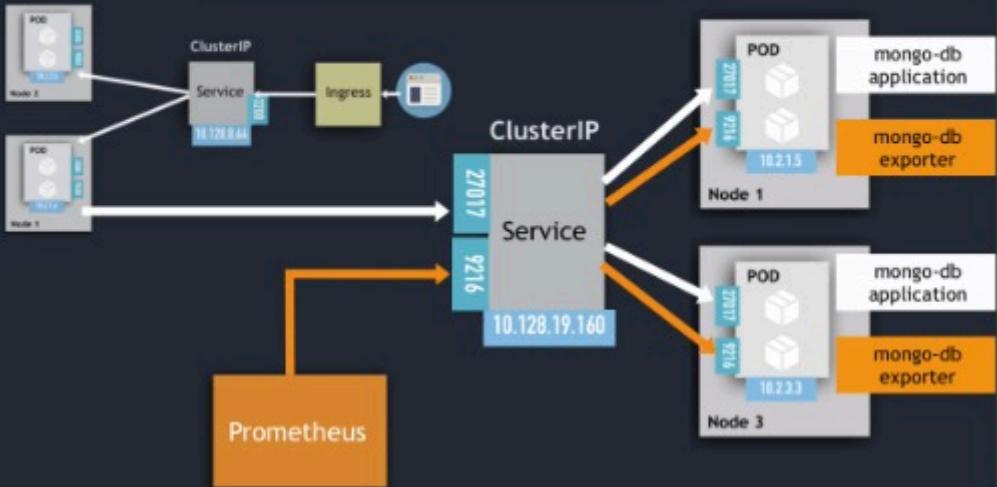


```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
...
spec:
  selector:
    app: mongodb
  ports:
    - name: mongodb
      protocol: TCP
      port: 27017
      targetPort: 27017
```

Example

10.128.19.160

Multi-Port Services



```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: mongodb-service
5 ...
6 spec:
7   selector:
8     app: mongodb
9   ports:
10  - name: mongodb
11    protocol: TCP
12    port: 27017
13    targetPort: 27017
14  - name: mongodb-exporter
15    protocol: TCP
16    port: 9216
17    targetPort: 9216
```

multiple pods

multiple pods

headless
services

Headless Services

- ▶ Client wants to communicate with 1 specific Pod directly

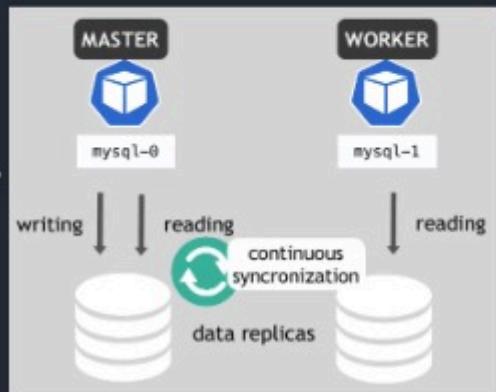
- ▶ Pods want to talk directly with specific Pod

- ▶ So, not randomly selected

- ▶ Use Case: Stateful applications, like

✗ Pod replicas are not identical

Only Master is allowed to write to DB



Headless Services

Client needs to figure out IP addresses

Option 1 - API call to K8s API Server

✗ makes app too tied to K8s API

✗ inefficient

Option 2 - DNS Lookup

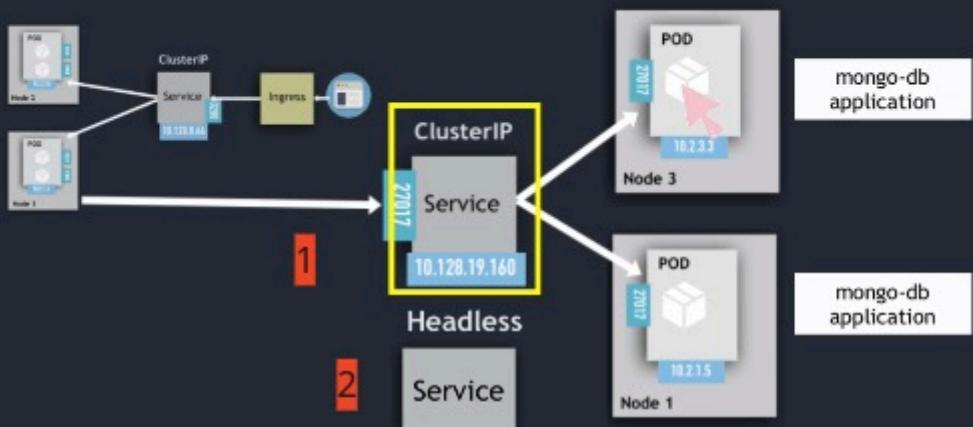
```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: mongodb-service-headless
5  spec:
6    clusterIP: None
7    selector:
8      app: mongodb
9    ports:
10   - protocol: TCP
11     port: 27017
12     targetPort: 27017
```

- ▶ DNS Lookup for Service - returns single IP address (ClusterIP)

- ▶ Set ClusterIP to "None" - returns Pod IP address instead ✓

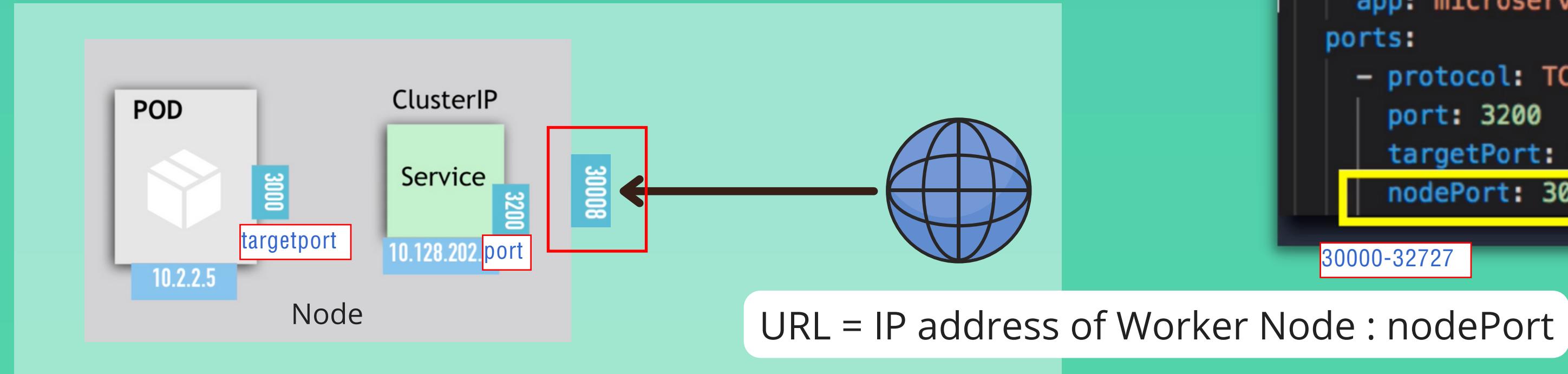
Define a headless services. -> set clusterIP none
Headless service and normal service (for loadbalance) are together there.

Headless Services



NodePort Service

- Unlike internal Service, is accessible directly from outside cluster
- **Exposes the Service on each Node's IP at a static port**



- A ClusterIP Service, to which the NodePort Service routes, is automatically created

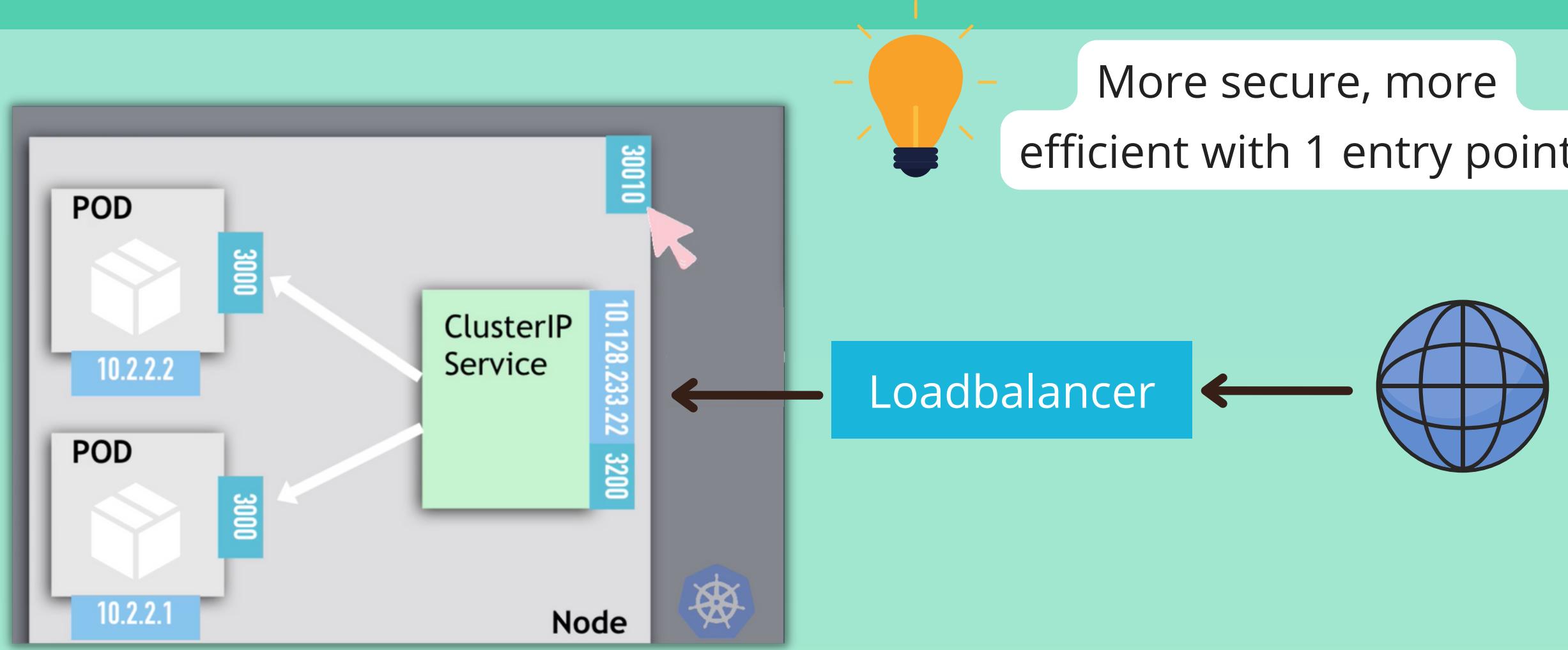
[k8s-services]\$ kubectl get svc	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
	kubernetes	ClusterIP	10.128.0.1	<none>	443/TCP	20m
	mongodb-service	ClusterIP	10.128.204.105	<none>	27017/TCP	10m
	mongodb-service-headless	ClusterIP	None	<none>	27017/TCP	2m8s
	ms-service-nodeport	NodePort	10.128.202.9	<none>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



Not Secure: External traffic has access to fixed port on each Worker Node

Loadbalancer Service

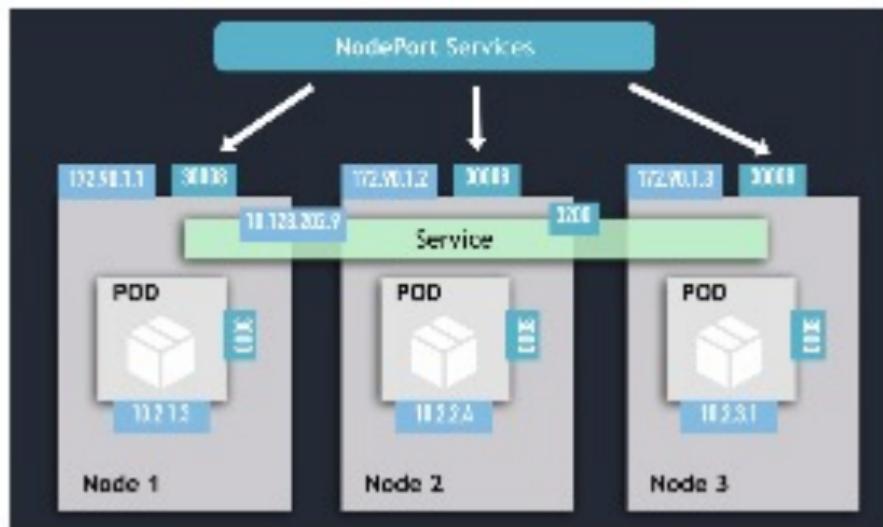
- Exposes the Service **externally using a cloud provider's load balancer**
- NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created



```
spec:  
  type: LoadBalancer  
  selector:  
    app: microservice-one  
  ports:  
    - protocol: TCP  
      port: 3200  
      targetPort: 3000  
      nodePort: 30010
```

[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
	monodb-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

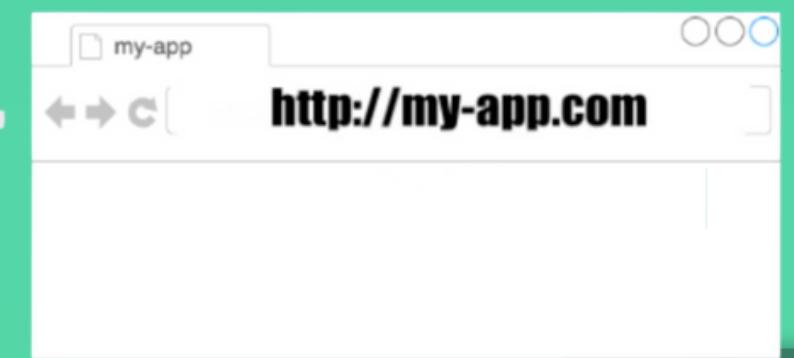
NodePort service not for external connection
Configure ingress or loadlance for production environment



Ingress - 1

- External Services are a way to access applications in K8s from outside
- **In production a better alternative is Ingress!**
- Not a Service type, but acts as the entry point for your cluster
- **More intelligent and flexible:** Let's you consolidate your routing rules into a single resource as it can expose multiple services under the same IP address

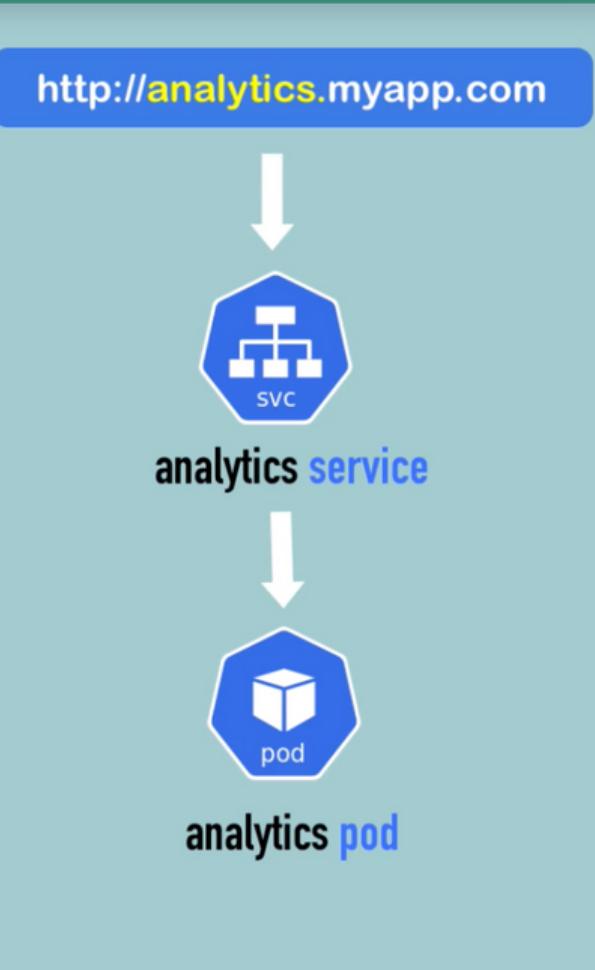
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
spec:
  rules:
  - host: myapp.com
    http:
      paths:
      - backend:
          serviceName: myapp-internal-service
          servicePort: 8080
```



Ingress - 2

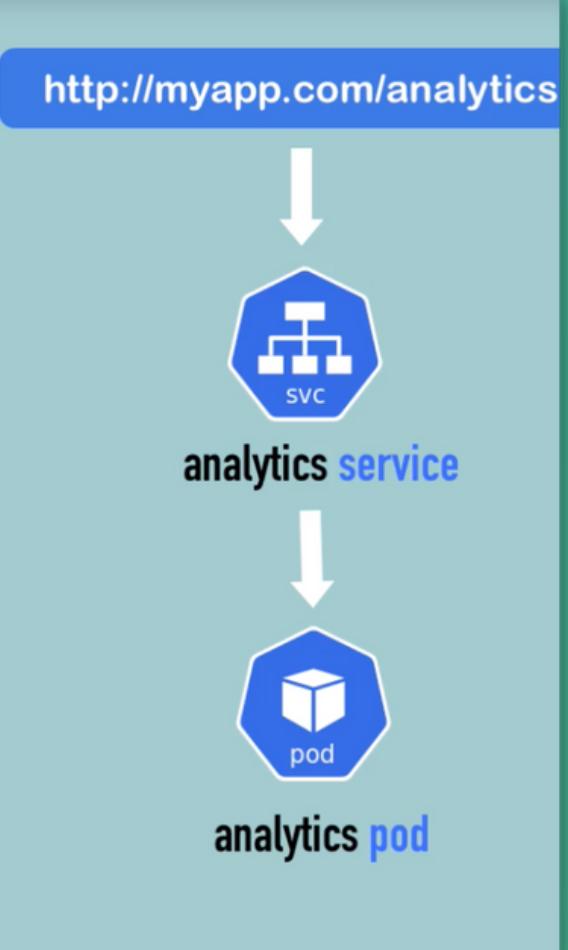
- Configure multiple sub-domains or domains:

```
apiVersion: networking.k8s.io/v1
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
```



- Configure multiple paths for same host:

```
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /analytics
            backend:
              serviceName: analytics-service
              servicePort: 3000
          - path: /shopping
            backend:
              serviceName: shopping-service
              servicePort: 8080
```



- Configure TLS

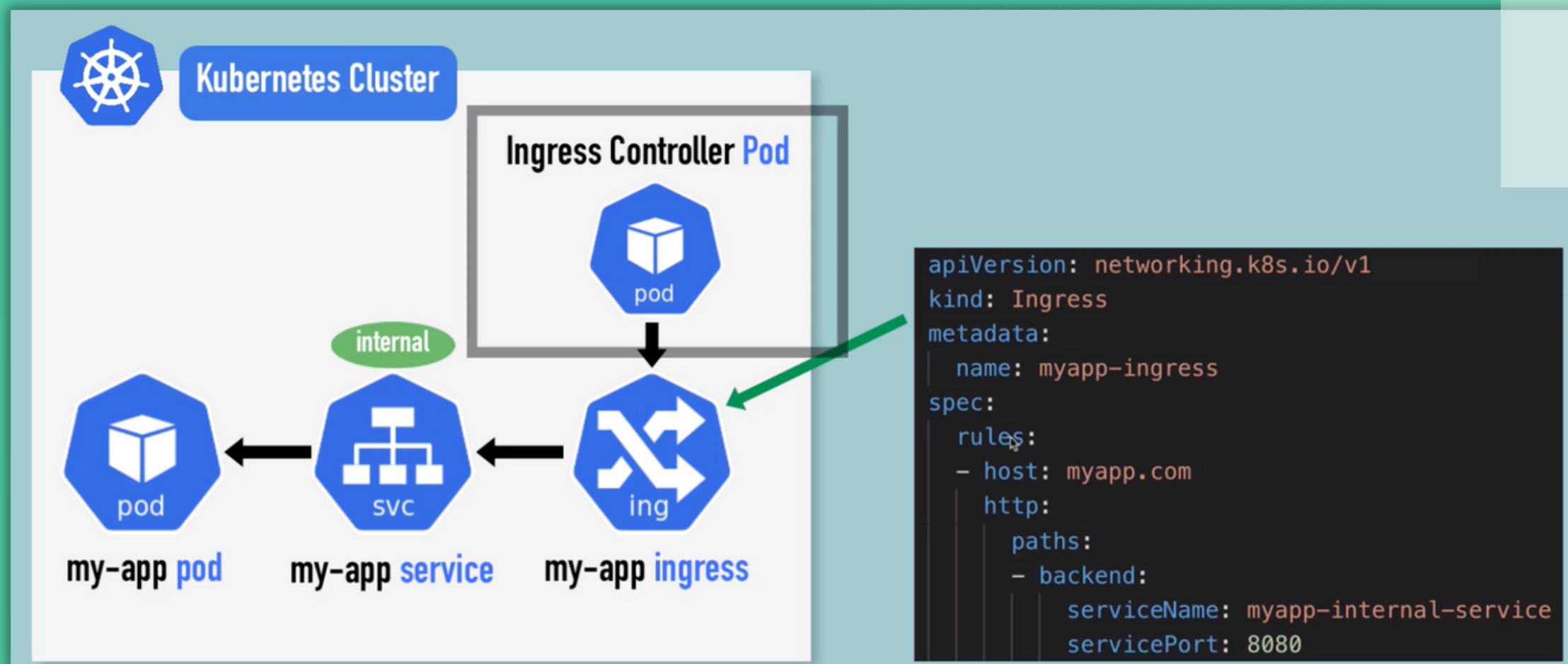
Certificate - `https://`

```
spec:
  tls:
    - hosts:
      - myapp.com
      secretName: myapp-secret-tls
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /
            backend:
              serviceName: myapp-internal-service
              servicePort: 8080
```

Ingress - 3

How to configure Ingress in your cluster

- You need an **implementation for Ingress**
- Which is Ingress Controller



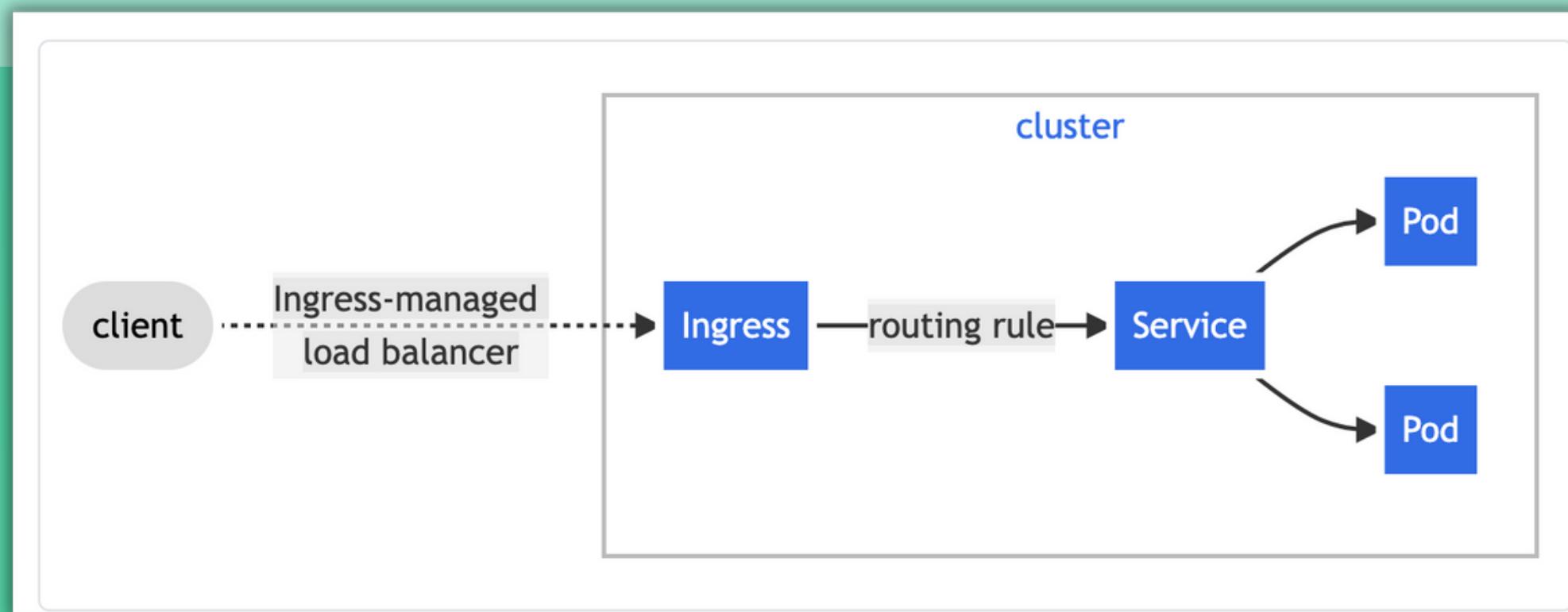
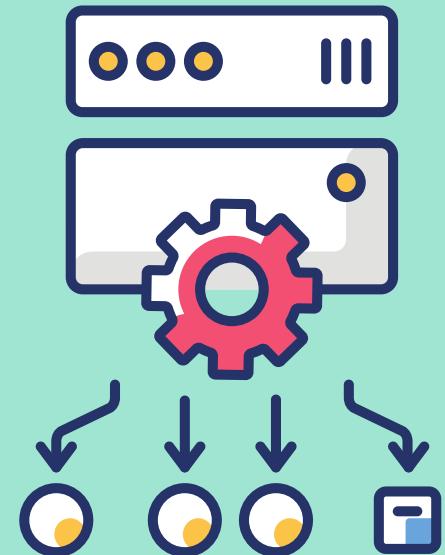
Ingress Controller

- Evaluates all the rules
- Manages redirections
- Entry point to the cluster
- Many third-party implementations
- K8s Nginx Ingress Controller

Ingress - 4

Before Ingress Controller you still need 1 load balancer

- **Option 1 - Cloud service provider:** have out-of-the-box K8s solutions
- **Option 2 - Bare Metal:** You need to configure some kind of entry point. Either inside the cluster or outside as separate server



Source: <https://kubernetes.io/docs/concepts/services-networking/ingress/>

Ingress and Internal Service configuration

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /internal-service
            pathType: Prefix
            service:
              name: my-ingress-service
              port:
                number: 8080
```

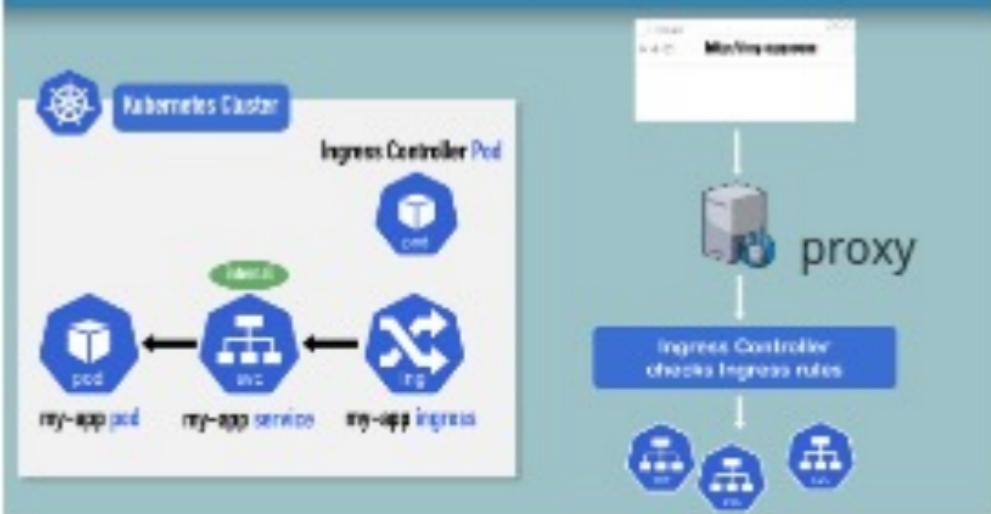
Internal Service

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

if use ingress: ingress reference the internal service -> servicename and serviceport should be redirected to internal service.
In this case: internal service **no nodeport** and instead of loadbalancer is default type **clusterIP**



Environment on which your cluster runs



To realize how the whole cluster works:

- install ingress controller in minikube

Install Ingress Controller in Minikube



```
[=]$ minikube addons enable ingress
✓ ingress was successfully enabled
```

NAME	READY	STATUS	RESTARTS	AGE
kube-apiserver-minikube	True	Running	0	32h
kube-controller-manager-minikube	True	Running	0	32h
kube-scheduler-minikube	True	Running	0	32h
kube-dns-minikube	True	Running	0	32h
kubelet-minikube	True	Running	0	32h
ingress-ingress-controller-6f888efc9-wqg9	True	Running	0	38m
metrics-scraper	True	Running	0	38m

- then create ingress rule
 - first start minikube dashboard.
 - then create ingress rules to access dashboard from a host

```
(base) [git:(origin-gh-pages)]$ kubectl get all -n kube-system-kubernetes-dashboard
NAME                                         READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-7dd55cbbf5-es67x   1/1     Running   0          3d
pod/kube-dns-kubernetes-dashboard-6f888efc9-wqg9   1/1     Running   0          38m

```

what do you have

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/dashboard-metrics-scraper	ClusterIP	10.96.226.24	<none>	80/TCP
service/kubernetes-dashboard	ClusterIP	10.97.224.7	<none>	443/TCP

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/dashboard-metrics-scraper	1/1	1	1	5d
deployment.apps/kubernetes-dashboard	1/1	1	1	5d

```
(base) ➜ giga-GIGA-2PC:~$ kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
coredns-d69d75c6d-hbfxt           1/1    Running   1 (22h ago)   23h
etcd-minikube                   1/1    Running   1 (22h ago)   23h
kube-apiserver-minikube          1/1    Running   1 (22h ago)   23h
kube-controller-manager-minikube  1/1    Running   1 (22h ago)   23h
kube-proxy-d5pb                1/1    Running   1 (22h ago)   23h
kube-scheduler-minikube          1/1    Running   1 (5h43m ago)  23h
storage-provisioner              1/1    Running   3 (5h43m ago)  23h
(base) ➜ giga-GIGA-2PC:~$ kubectl get ns
NAME        STATUS   AGE
default     active  23h
ingress-nginx   Active  2m44s
kube-node-lease  Active  23h
kube-public    Active  23h
kube-system    Active  23h
(base) ➜ giga-GIGA-2PC:~$ kubectl get all -n kubernetes-dashboard
No resources found in kubernetes-dashboard namespace.
(base) ➜ giga-GIGA-2PC:~$ minikube dashboard
[+] Aktiviere Dashboard ...
  ■ Verwende Image kubernetesui/metrics-scraper:v1.0.0
  ■ Verwende Image kubernetesui/dashboard:v2.0.8
```

```
[ mongo-expressagent ] [ dashboard-ingress.yaml.v3 ]
```

```
done@project > [ dashboard-ingress ] > {spec} > {index} > {id} > {http} > {path} > {1}
  .json
  1 apiVersion: networking.k8s.io/v1
  2 kind: Ingress
  3 metadata:
  4   name: dashboard-ingress
  5   namespace: kubernetes-dashboard
  6 spec:
  7   rules:
  8     - host: dashboard.com
  9       http:
 10         paths:
 11           - path: /
 12             pathType: Prefix
 13             backend:
 14               service:
 15                 name: kubernetes-dashboard
 16                 port:
 17                   number: 80
 18
```

check minikube system and start dashboard

```
(base) g@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Basic  
$ kubectl get ingress -n kubernetes-dashboard -w  
NAME          CLASS   HOSTS   ADDRESS      PORTS   AGE  
dashboard-ingress   nginx   dashboard.com   192.168.49.2   80     5m13s  
exit  
NANO(hp) ~ %  
tccamp$ nano project$ sudo vim /etc/hosts  
[sudo] Password for gu:  
(base) g@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Basic
```

create an ingress -> get the ip address -> map it
the dashboard.com in the /etc/hosts file

```
127.0.0.1      localhost  
127.0.1.1      gu-GE60-2PC  
192.168.49.2    dashboard.com  
# The following lines are desirable for IPv6 support:  
::1      ip6-localhost ip6-loopback  
fe00::0 ip6-localnet
```

request to minikube -> handel to ingress controller
-> go to validate the mapping -> then go to service

Configure Default Backend in Ingress

```
$ kubectl getcm ingress myapp-ingress
Name:           myapp-ingress
Namespace:      default
Address:        default
Default backend: default-http-backend:80 (Kubernetes)
Rules:
  Host   Path   Backend
  myapp.com   /<path>/internal-service:8080 (Kubernetes)
```

```
apiVersion: v1
kind: Service
metadata:
  name: default-http-backend
spec:
  selector:
    app: default-response-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Configuring TLS Certificate - https://

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
    - hosts:
        - myapp.com
      secretName: myapp-secret-tls
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /
            backend:
              service:
                name: myapp-service
                port: 8080
```

```
apiVersion: v1
kind: Secret
metadata:
  name: myapp-secret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

Configuring TLS Certificate - https://



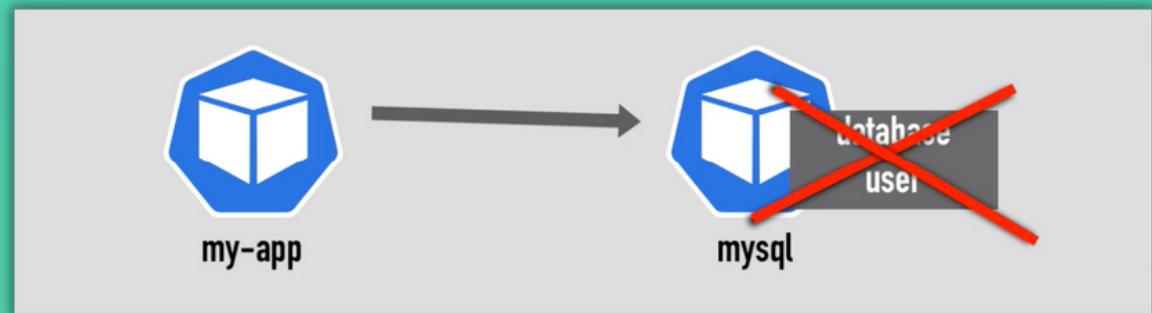
```
apiVersion: v1
kind: Secret
metadata:
  name: myapp-secret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

- 1) Data keys need to be "tls.crt" and "tls.key"
- 2) Values are file contents
NOT file paths/locations
- 3) Secret component must be in the
same namespace as the Ingress
component

Deep Dive into Kubernetes Volumes

Volumes - 1

- K8s offers no data persistence out of the box



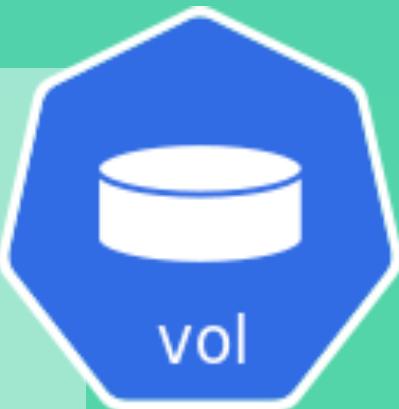
- At its core, a **volume** is a directory (with some data in it), which is accessible to the containers in a Pod
- K8s supports **many types of volumes**
- Ephemeral volume types have a lifetime of a Pod, persistent volumes exist beyond the lifetime of a Pod

K8s volume pod as interface

- In this lecture we talk about **persistent volumes**, with these storage requirements:

Storage Requirements

- Storage that **doesn't depend on pod lifecycle**
- Storage must be **available on all Nodes**
- Storage needs to **survive even if cluster crashes**



Volumes - 2

- The way to persist data in K8s using Volumes is with these 3 resources:



Persistent Volume (PV)

- Storage** in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes



Storage Class (SC)

- SC provisions PV dynamically when PVC claims it



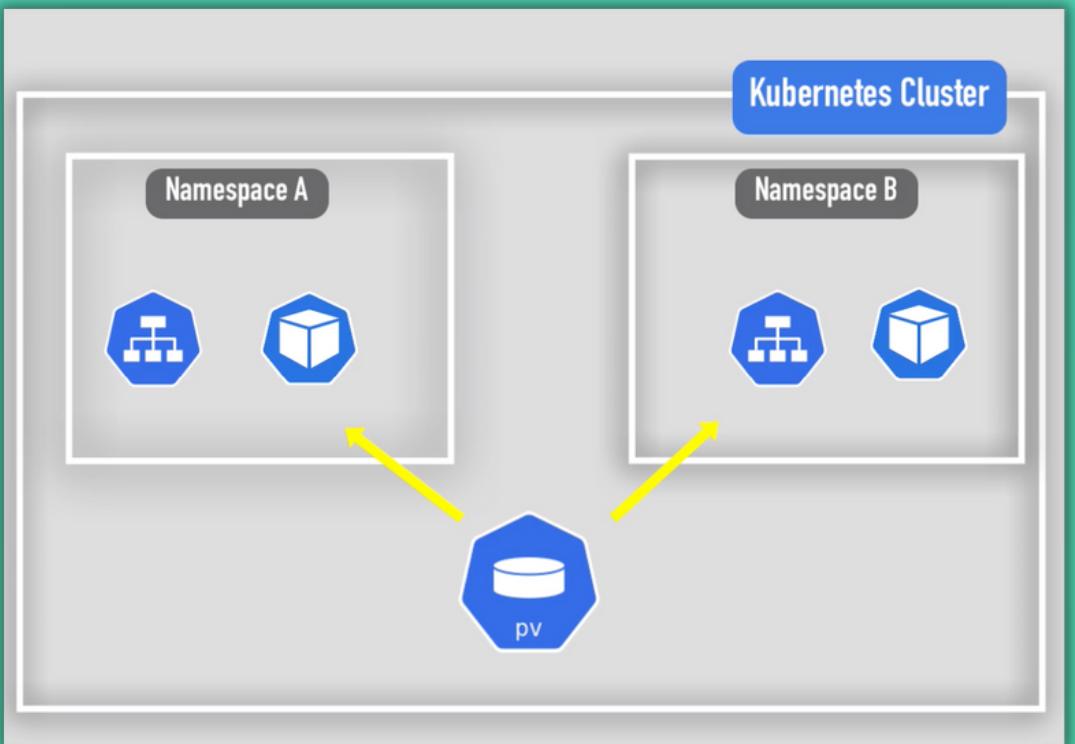
Persistent Volume Claim (PVC)

- A **request for storage** by a user
- Similar to Pods. While Pods consume node resources, PVCs consume PV resources

Persistent Volume - 1



- Persistent Volumes are **NOT namespaced**, so PV resource is accessible to the whole cluster:



- Depending on storage type, spec attributes differ
- In official documentation you can find a complete list of storage backends supported by K8s:

Documentation Blog Training Partners Community Ca

the Pod must independently specify where to mo

Types of Volumes

Kubernetes supports several types of Volumes:

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi

- Configuration Example

```
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 - 2

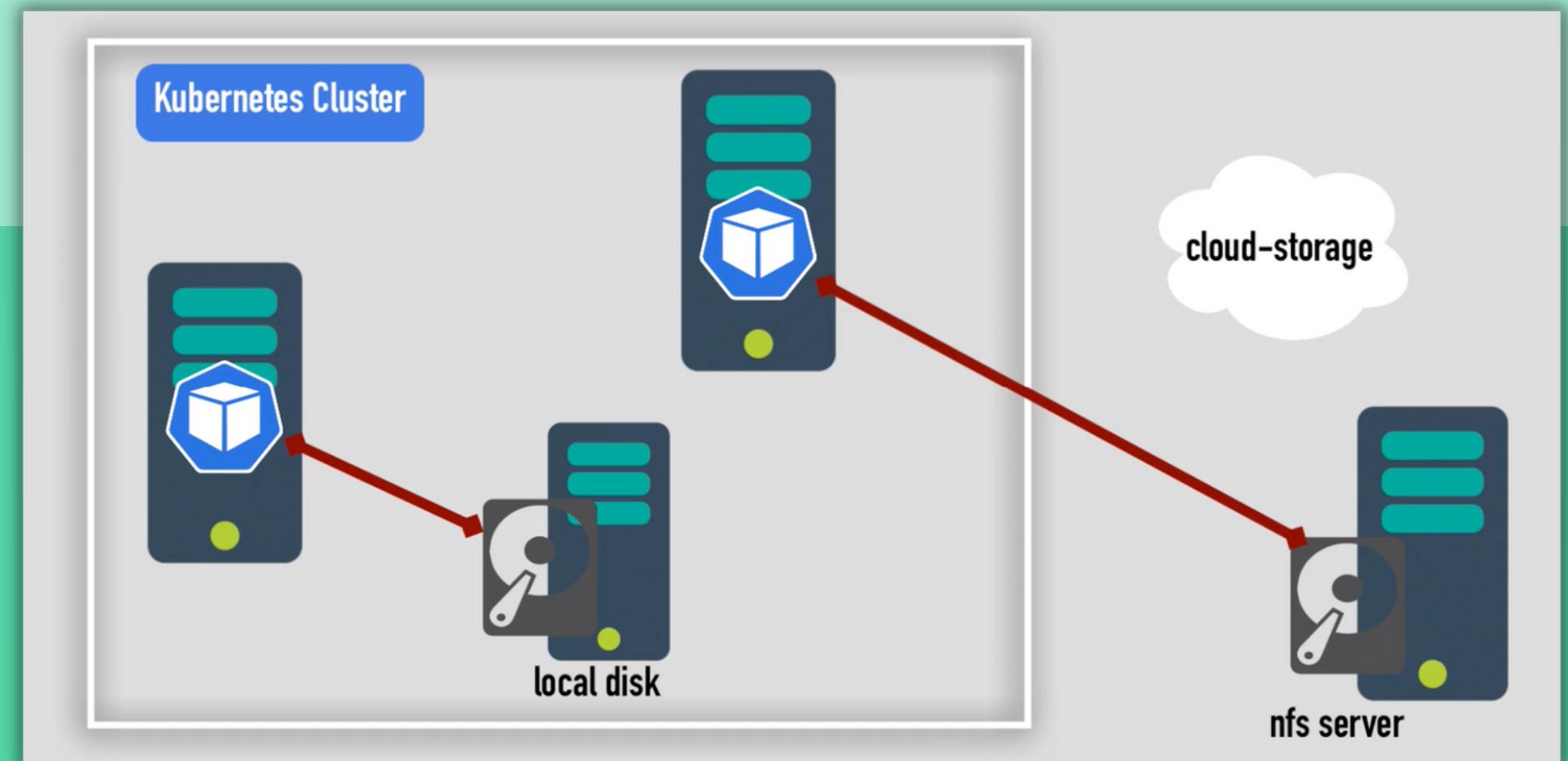
Local vs Remote Volume Types

- Each volume type has its own use case!
- **Local volume types** violate 2. and 3. requirement for data persistence:

- ✖ Being tied to 1 specific Node
- ✖ Not surviving cluster crashes



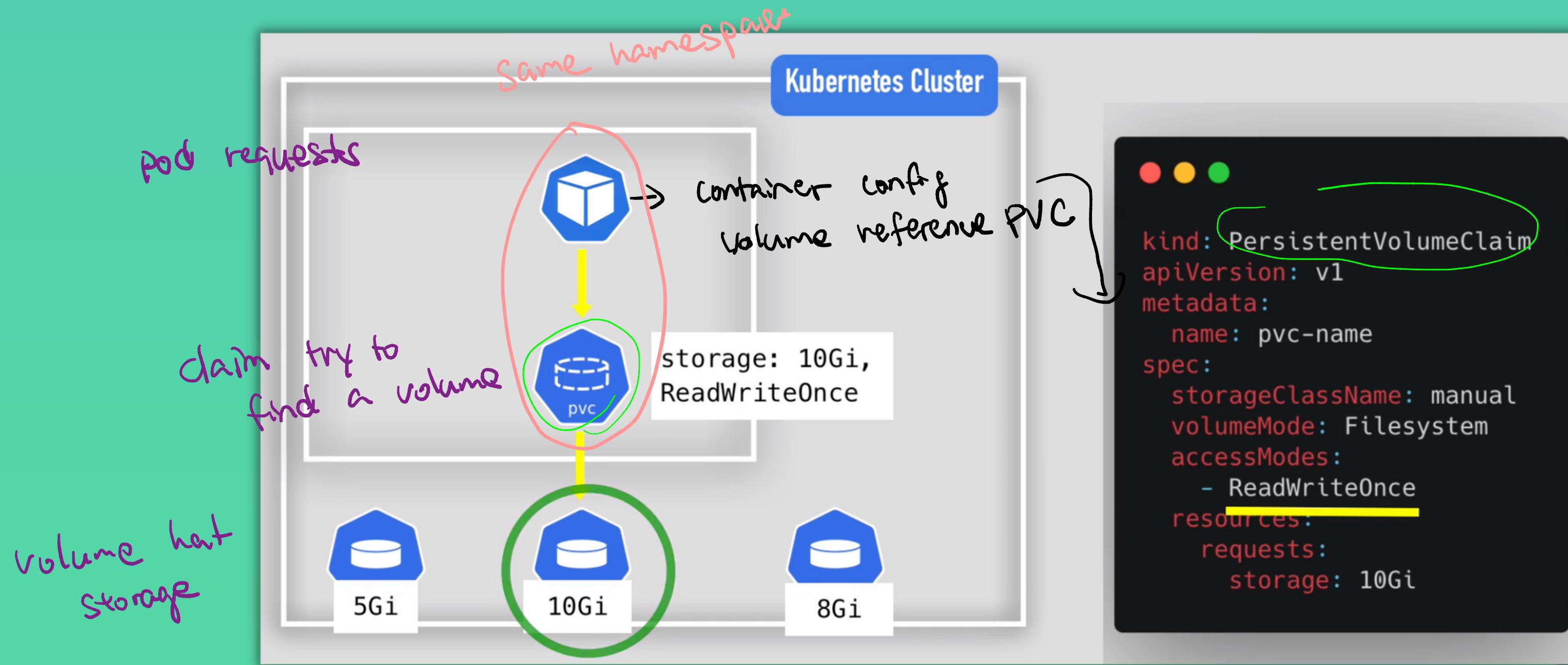
For DB persistence use **remote storage!**



Persistent Volume Claim PVC



- Request for storage by a user
- Claims can **request specific size and access modes** (e.g. they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany).



Levels of Volume abstractions

Kubernetes Cluster



```
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
```

pod we g
↓
still access
storage

Volume is mounted into Container

Volume is mounted into the Pod

1 pod has three different volume type

```
containers:
- image: elastic:latest
  name: elastic-container
  ports:
  - containerPort: 9200
  volumeMounts:
  - name: es-persistent-storage
    mountPath: /var/lib/data
  - name: es-secret-dir
    mountPath: /var/lib/secret
  - name: es-config-dir
    mountPath: /var/lib/config
volumes:
- name: es-persistent-storage
  persistentVolumeClaim:
    claimName: es-pv-claim
- name: es-secret-dir
  secret:
    secretName: es-secret
- name: es-config-dir
  configMap:
    name: es-config-map
```

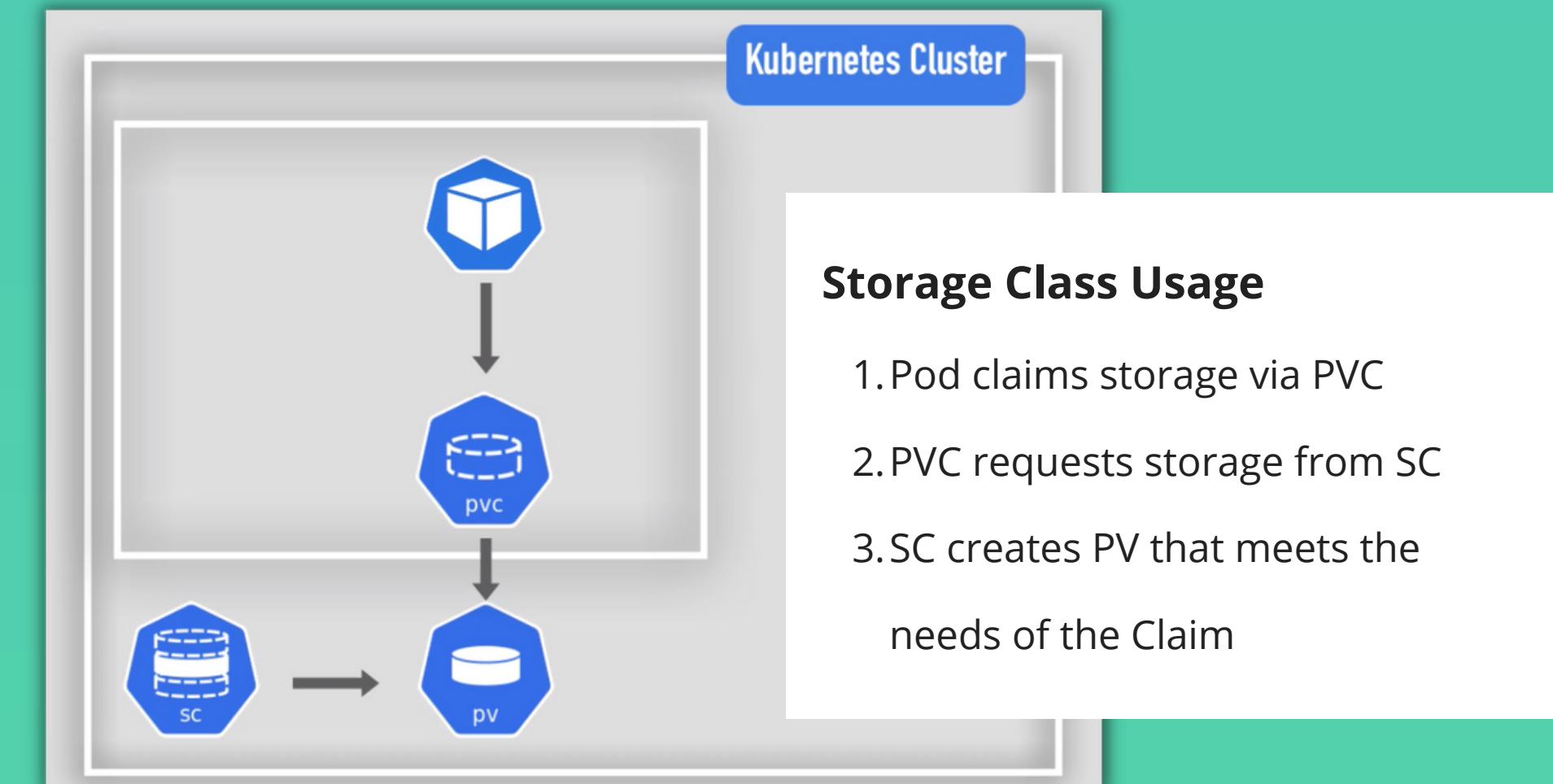
这节课可以创建 PV dynamically

Storage Class



- Provisions PV dynamically

- ConfigMap + secret are local volume
↓
connect to PV (is just)



- **StorageBackend** is defined in the SC resource

- 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

```
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



PVC Config

Storage Class Config



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



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

Deep Dive into StatefulSets

When to use ConfigMap and Secret volumes?

ConfigMap and Secret are used for **external configuration** of individual values.

```
1 | mongo-express.yaml
2 | -----
3 |   name: mongo-express
4 |   image: mongo-express
5 |   ports:
6 |     - containerPort: 8081
7 |   env:
8 |     - name: ME_CONFIG_MONGODB_ADMINUSERNAME
9 |       valueFrom:
10 |         secretKeyRef:
11 |           name: mongodb-secret
12 |           key: admin
13 |     - name: ME_CONFIG_MONGODB_ADMINPASSWORD
14 |       valueFrom:
15 |         secretKeyRef:
16 |           name: mongodb-secret
17 |           key: password
18 |     - name: ME_CONFIG_MONGODB_SERVER
19 |       valueFrom:
20 |         configMapKeyRef:
21 |           name: mongodb-configmap
22 |           key: db_host
23 |
24 |   apiVersion: v1
25 |   kind: Service
26 |   metadata:
27 |     name: mongo-express-service
```

use of configmap and secret of mongodb in the mongoexpress

ConfigMap and Secret:

- could be individual key-value pairs
- or files mounted in the pod and then into the container
-

Demo project:

- create a mosquito pod and get into it. The default files. The default mosquito-config file is inside. We need to overwrite the default config file

```
(base) guiguo-DESKTOP-1E0C1:~/Documents/learn_develops/learn_kubernetes_bootcamp/configMapAndSecretDemo$ kubectl exec -it mosquitto-5d8547446c-845sh -- /bin/sh
/ # ls
bin          media          abin
dev          net            srv
docker-entrypoint.sh mosquitto  sys
etc          proc           tmp
home         root           usr
lib          run            var
/ # cd mosquitto/
/mosquitto # ls
config  data  log
mosquitto #
```

ConfigMap and Secret must be created and exist before Pod starts in the kubernetes cluster

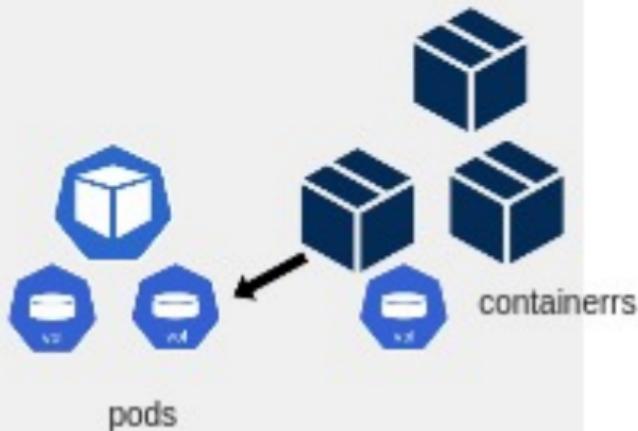
```
echo -n 'super-secret-password' | base64
```

- create configmap and secret.
 - create mosquito deployment using these two files -> mount the volumes in the spec ->template ->spec -> volumes then in the spec->template->spec-> container->volumes
 - logic:
 - volumes mounted into Pod
 - mount volumes into container

```
app: mosquitto
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mosquitto
  template:
    metadata:
      labels:
        app: mosquitto
  spec:
    containers:
      - name: mosquitto
        image: eclipse-mosquitto:1.6.2
        ports:
          - containerPort: 1883
    volumeMounts:
      - mountPath: /mosquitto/config
        name: mosquitto-config
      - mountPath: /mosquitto/secret
        name: mosquitto-secret
        readOnly: true
    volumes:
      - name: mosquitto-config
        configMap:
          name: mosquitto-config-file
      - name: mosquitto-secret
        secret:
          secretName: mosquitto-secret-file
```

configmap and secret have different attributes

Abstraction useful for multiple containers



if the container is in containerCreating status. you can use kubectl describe pod to find the details.

configmap two usages:

- mount as volume types

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mosquitto-config-file
data:
  mosquitto.conf: |
```

```
  log_dest stdout
  log_type all
  log_timestamp true
```

```
  listener 9081
```



create files

- usage as env variables

Individual key-value pairs

```
data:
  db_host: mongo-service
```

```
key: password
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongo-tattoo-root-secret
data:
  mongo-root: |
```

```
  - name: mongo-root
    valueFrom:
      secretKeyRef:
        name: mongo-root-secret
        key: username
```

```
  - name: mongo_db_root_password
    valueFrom:
      secretKeyRef:
        name: mongo-db-secret
        key: password
```



StatefulSet

- Used to **manage stateful applications**, like databases
- Manages the deployment and scaling of a set of Pods and **provides guarantees about the ordering and uniqueness of these Pods**

Stateless Applications

- Doesn't depend on previous data
- Deployed using Deployment



Stateful Applications

- Update data based on previous data
- Query data
- Depends on most up-to-date data/state

Both manage Pods based on container specification

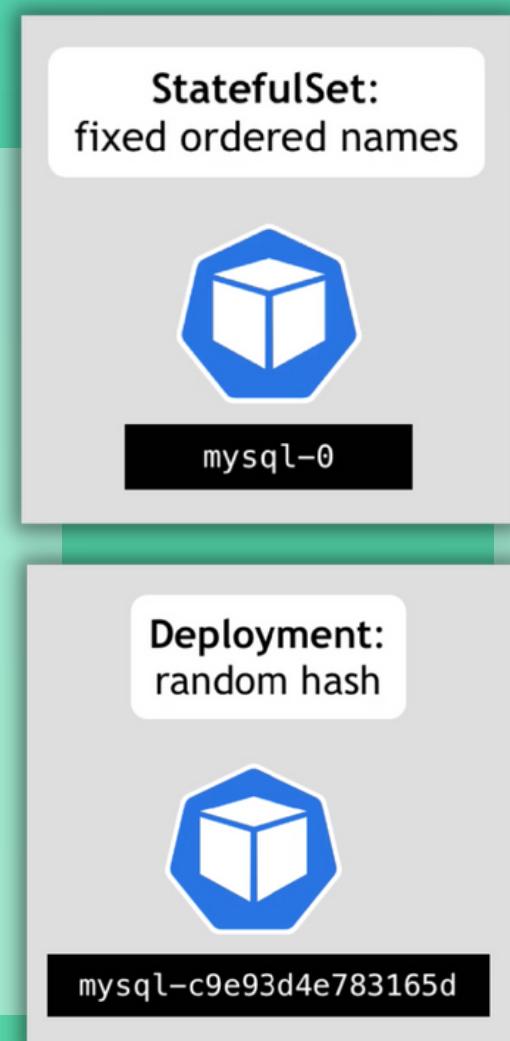
Deployment vs StatefulSet

- Unlike a Deployment, a StatefulSet maintains a **sticky identity** for each of their Pods
- These pods are created from the **same spec, but are not interchangeable**: each has a persistent identifier that it maintains across any rescheduling

Pods created from StatefulSet

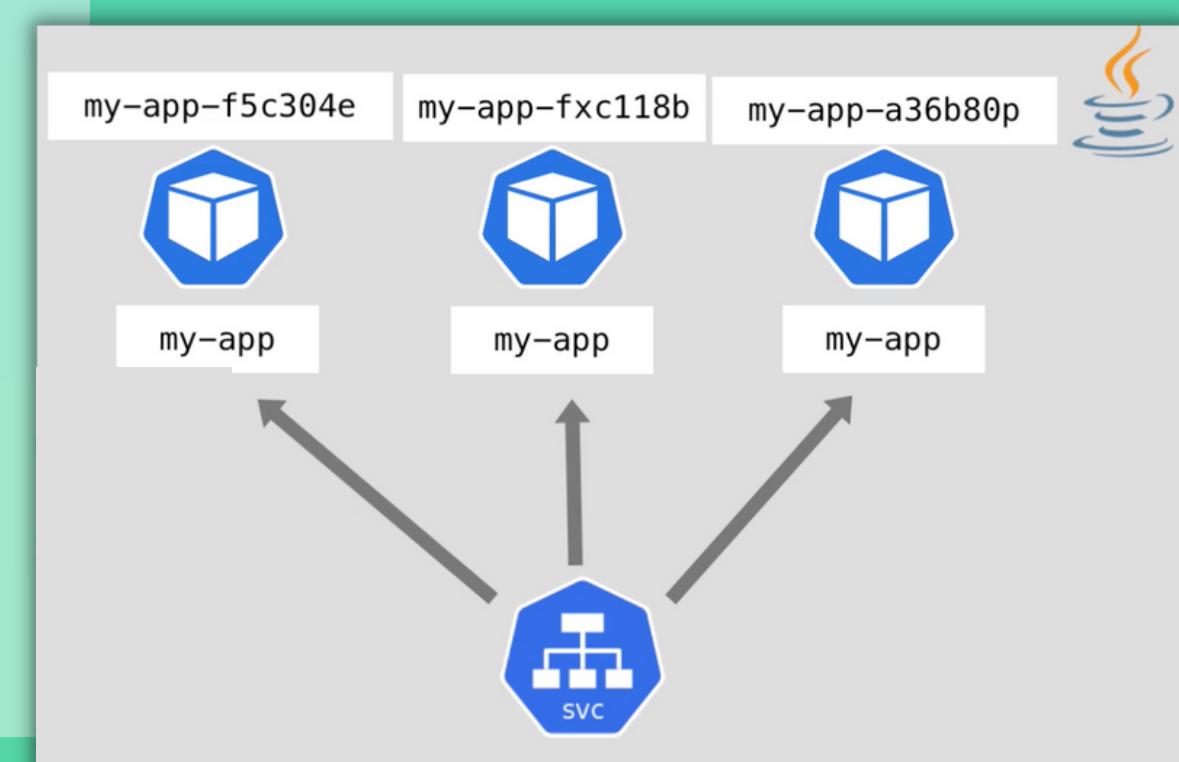
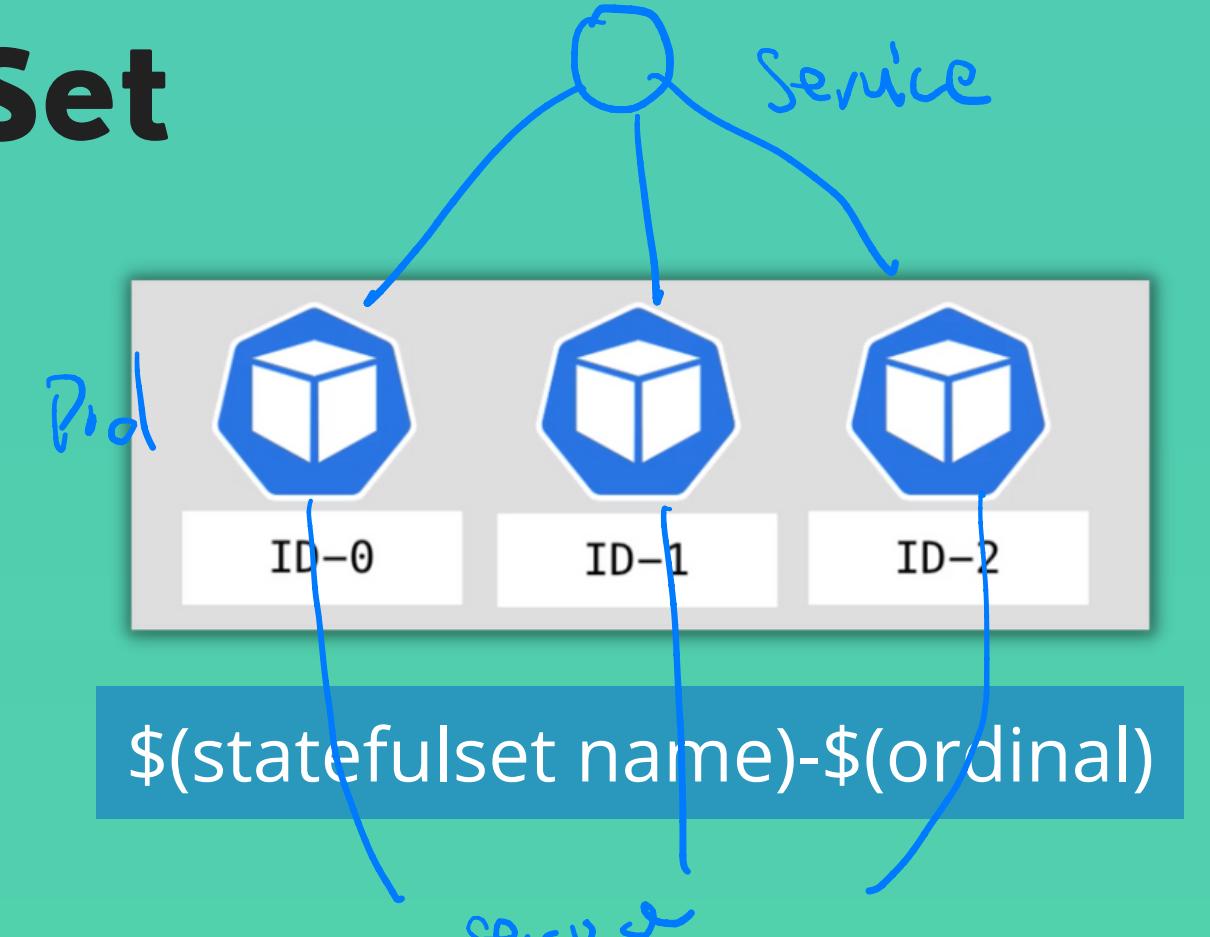
- More difficult
- Can't be created/deleted at same time
- Can't be randomly addressed

↓
because of replica Pods
are not identical



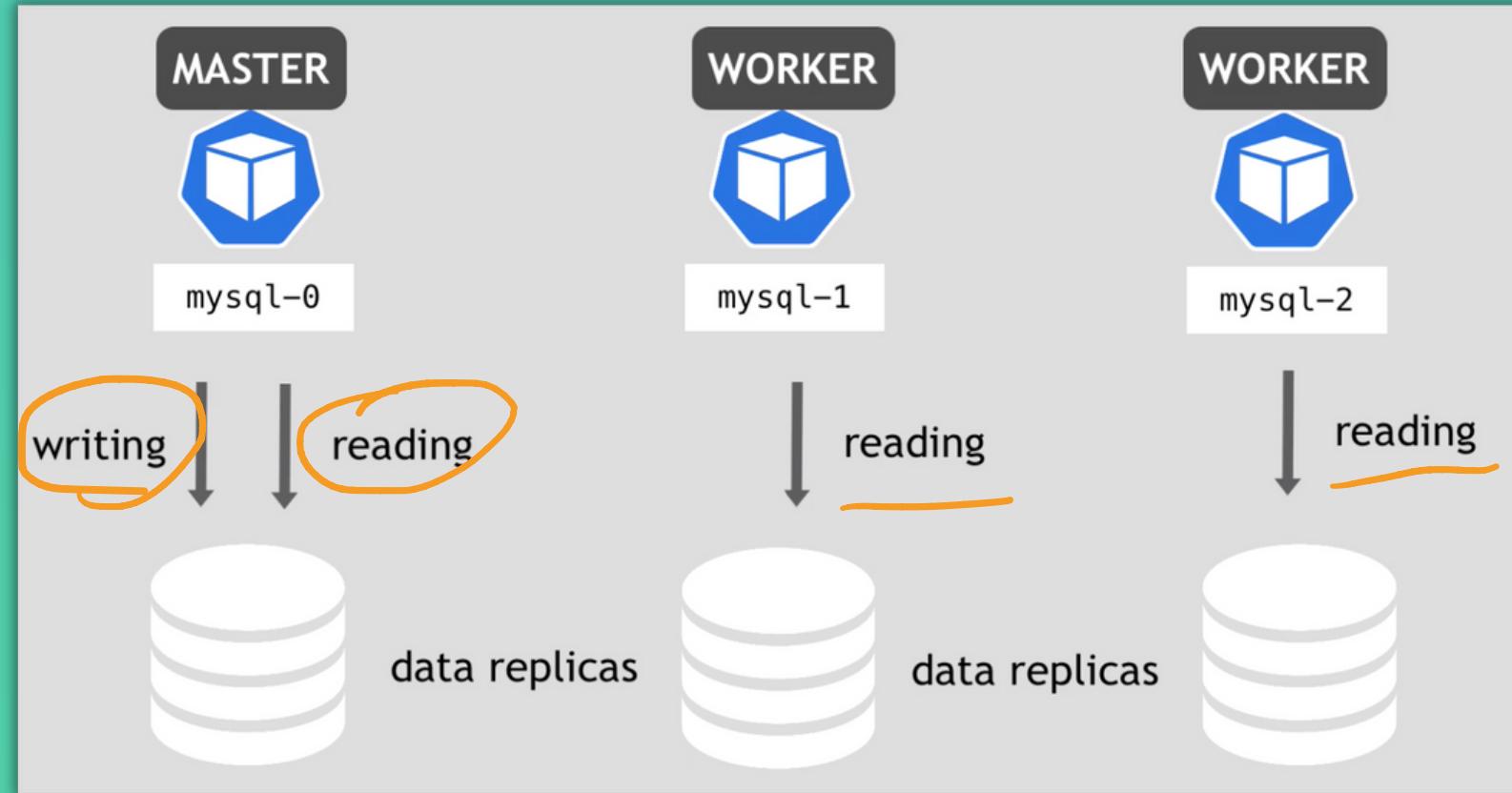
Pods created from Deployment

- Identical and interchangeable
- Created in random order with random hashes
- 1 Service that load balances to any Pod



↓ why so
is stateful

Scaling database applications



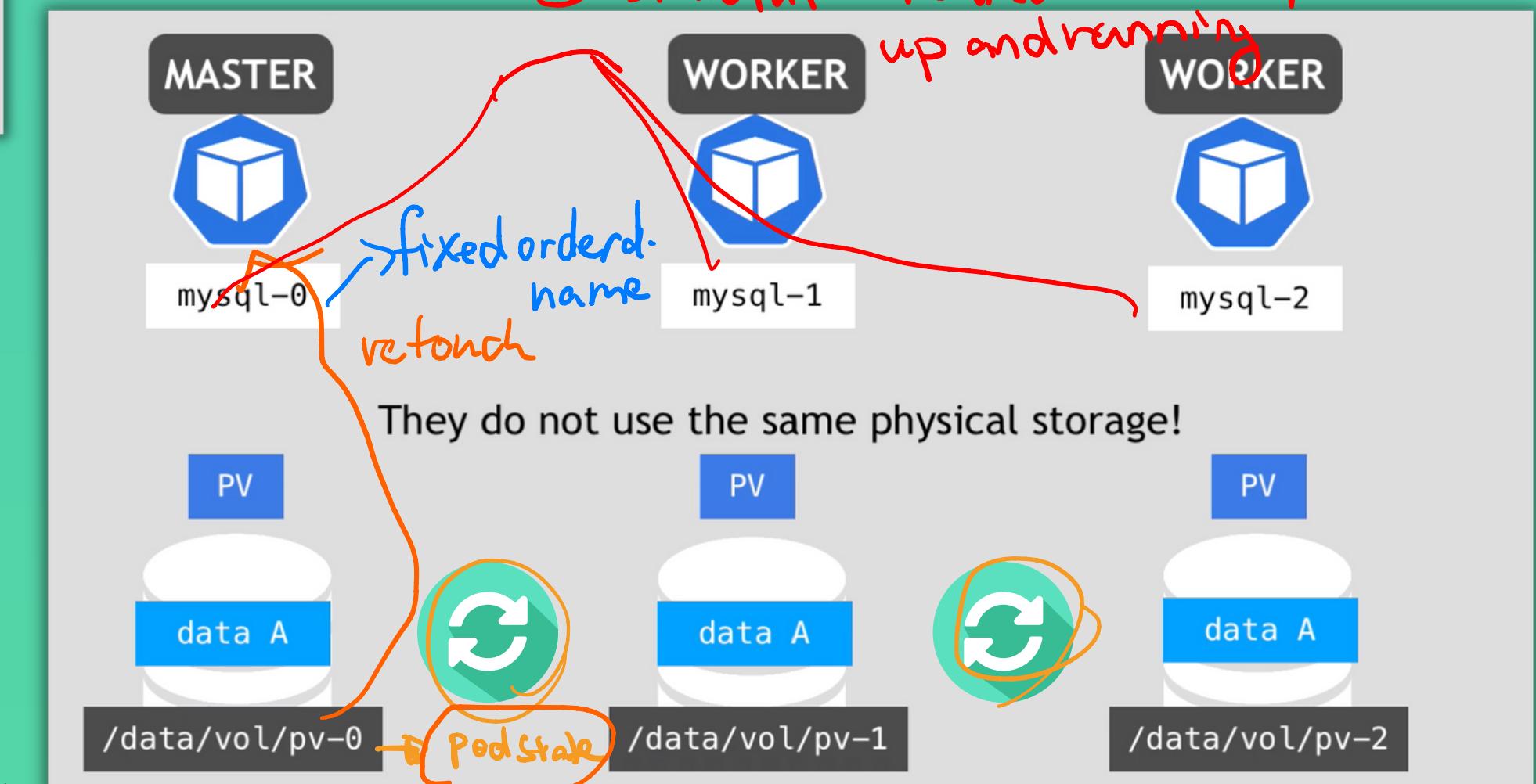
- Each replica has its own storage
- These storages are constantly

synchronized

! in the PV we need **data**
pod state = info of the pod

- Only 1 replica, can make changes
- So replicas are not identical

! data persistence PV for StatefulSet
① delete start the last one
② stateful created when previous
up and running



⇒ Pods died , → replaced Pod state → Pod volume will be reattached
→ using remote storage (local store is tied to specific pod)

Managed K8s Service

Kubernetes on Cloud platform

2 options to create a Kubernetes cluster on a cloud platform

Create own cluster from scratch

- ✖ You need to manage everything yourself
- ✖ Not practical, when you want to setup things fast and easy



Use Managed K8s Service

- You only care about Worker Nodes
- Everything pre-installed
- Control Plane Nodes created and managed by cloud provider
- You only pay for the Worker Nodes
- Use cloud native load balancer for Ingress controller
- Use cloud storage
- Less effort and time



Example Managed Kubernetes Services

- **AWS:** Elastic Kubernetes Service (EKS)
- **Azure:** Azure Kubernetes Service (AKS)
- **Google:** Google Kubernetes Engine (GKE)
- **Linode:** Linode Kubernetes Engine (LKE)



Helm Package Manager

Helm - Package Manager

- Helm is the **package manager for Kubernetes**. Think of it like apt/yum for Kubernetes
- Packages YAML files and distributes them in public and private repositories

Helm

- Tool that installs and manages K8s applications
- This is done via Charts, so Helm manages these Charts



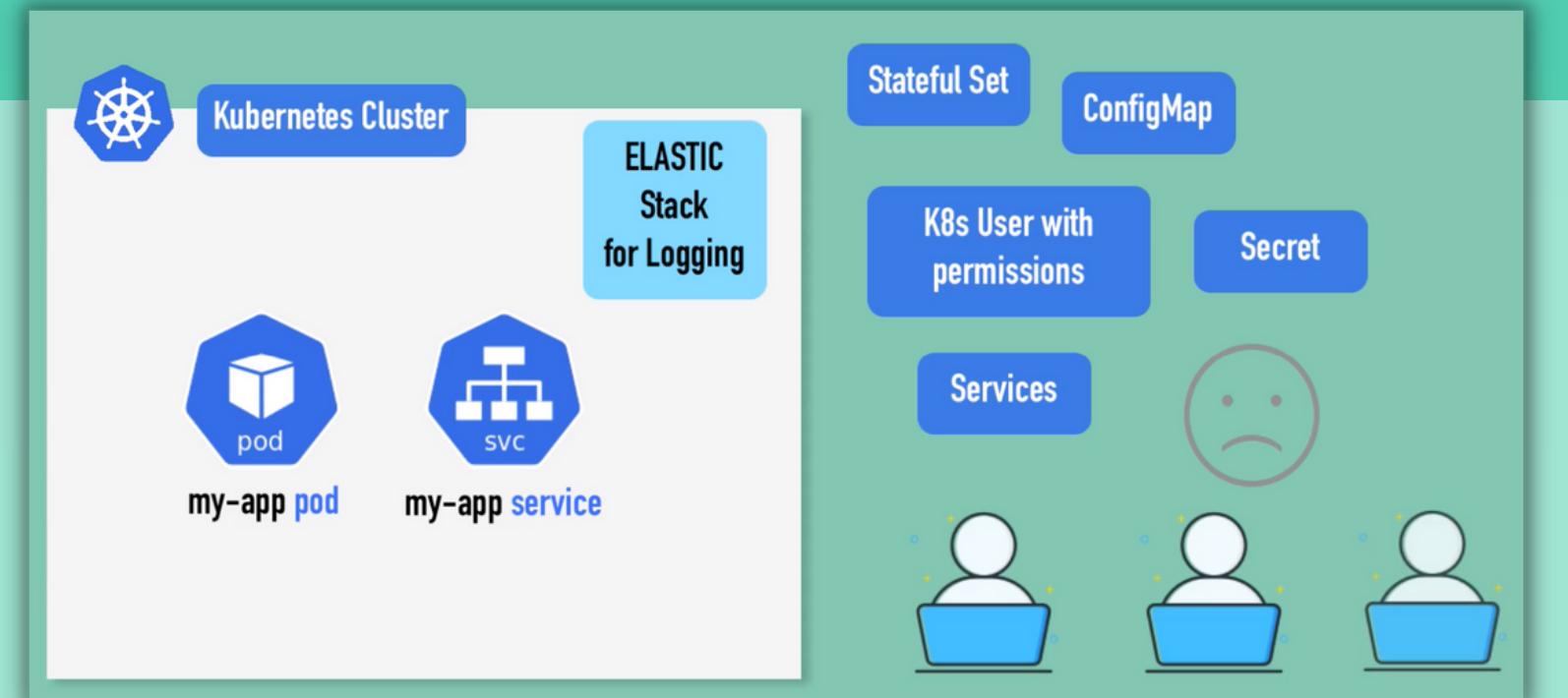
Helm Chart

- **Helm package** that contain
 - description of the package Chart.yaml
 - 1 or more templates, which contain K8s manifest files
- You can create your own Helm Charts with Helm, push to Helm Repository
- Download and use existing ones

Helm Charts

Why Helm Charts?

- Instead of everyone creating their own K8s config files, 1 bundle of all needed K8s manifests



Use existing official Charts:

- Many of these created by the official sources.
 - Mysql Chart from Mysql or ElasticSearch Chart from ElasticSearch

Sharing Helm Charts:

- Charts are hosted on their own repositories
- There are public repos, but you can have own private repos in your company. E.g. Nexus

`helm search <keyword>`

or Helm Hub website

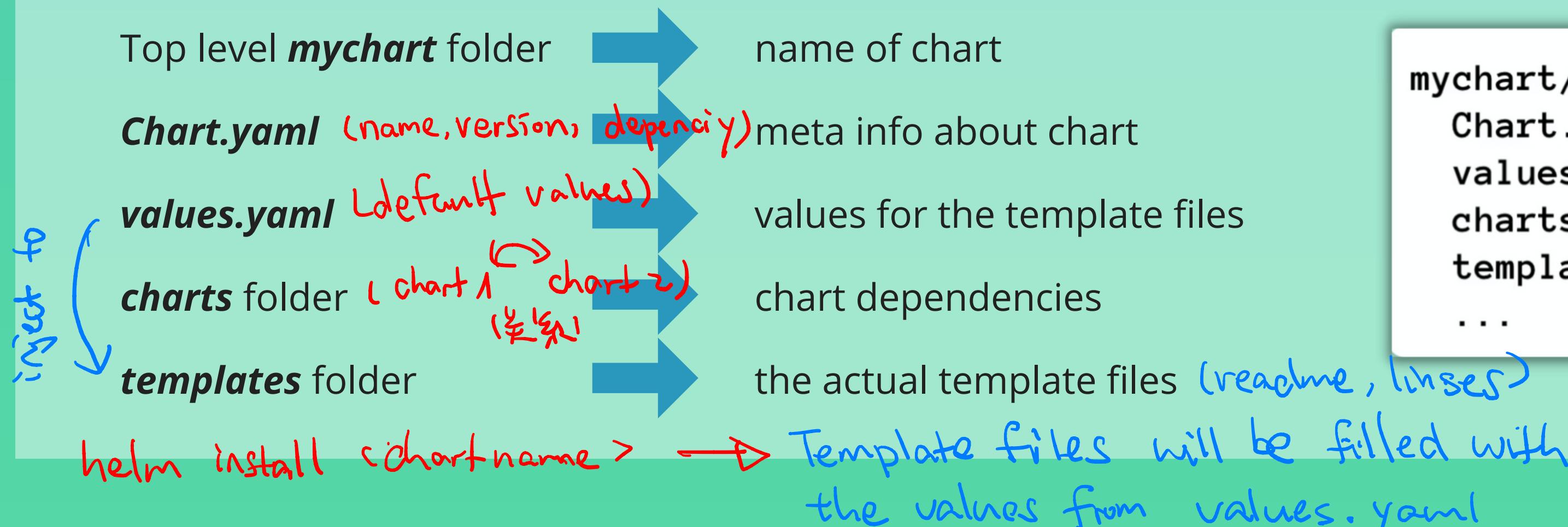
Helm as Templating Engine - 1

- Helm **renders the templates** and communicates with the **Kubernetes API**

How to:

1. Define a common blueprint
2. Dynamic values are replaced by placeholders

Helm Chart Structure



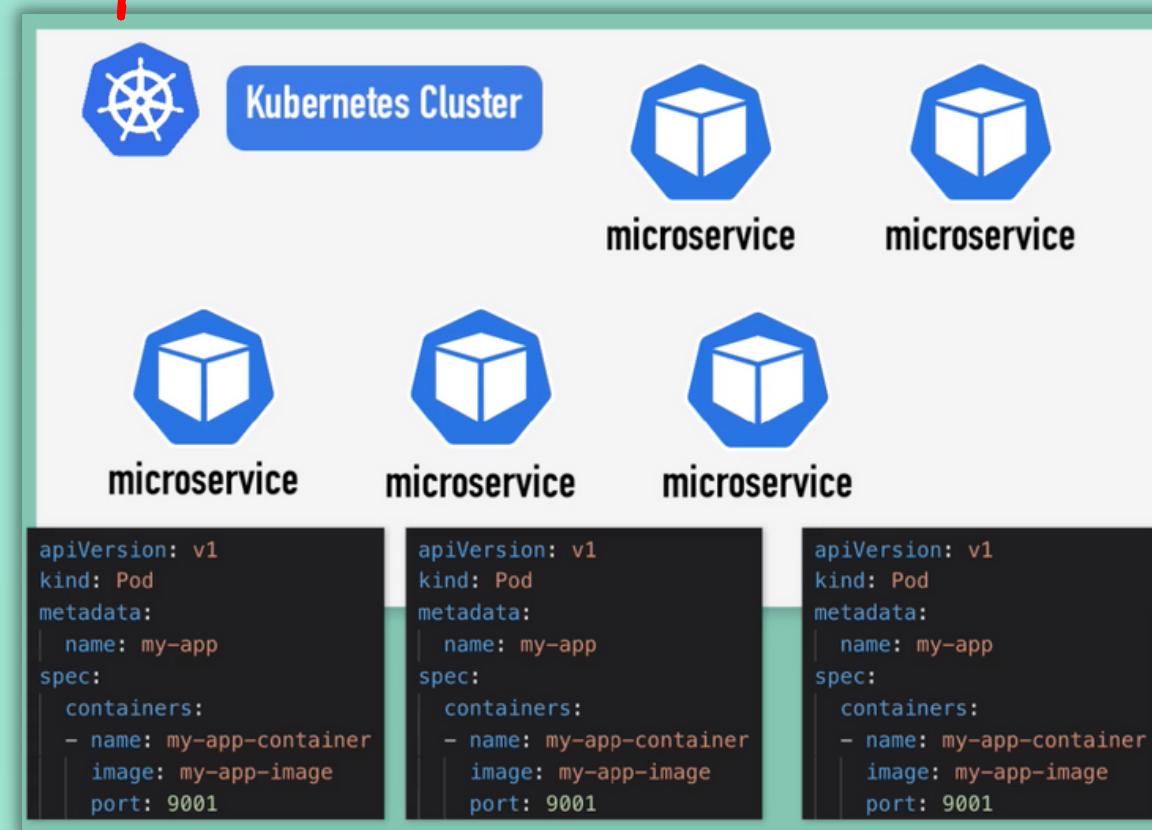
```
mychart/
  Chart.yaml
  values.yaml
  charts/
  templates/
  ...
```

Helm as Templating Engine - 2

①

Practical for CI/CD

- Many values are the same!



many YAML files



just 1 YAML file

```
apiVersion: v1
kind: Pod
meta[redacted]
spec[redacted]
```

A large white arrow points from the 'many YAML files' section to this block of repetitive YAML code.

```
apiVersion: v1
kind: Pod
meta[redacted]
spec[redacted]
```

```
apiVersion: v1
kind: Pod
meta[redacted]
spec[redacted]
```

```
apiVersion: v1
kind: Pod
meta[redacted]
spec[redacted]
```

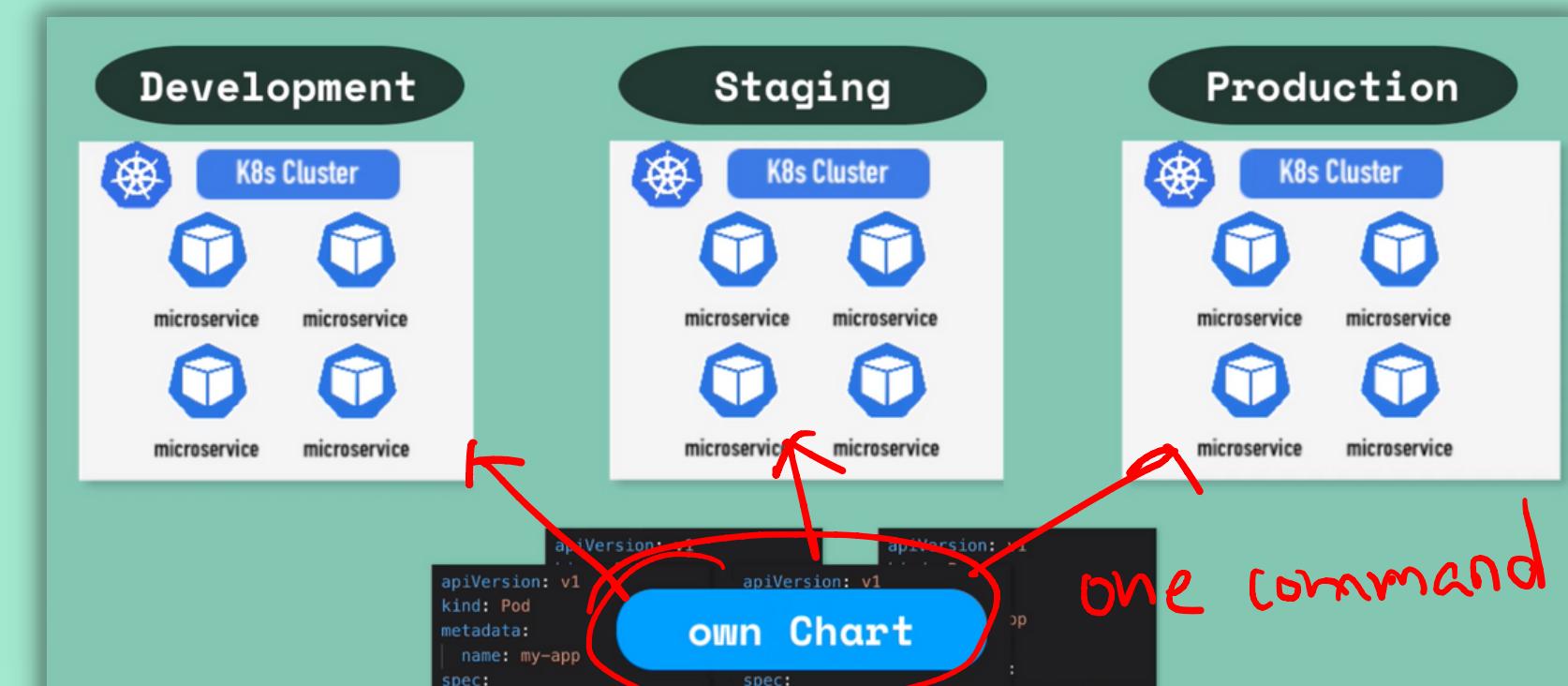
```
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Values.name }}
spec:
  containers:
    - name: {{ .Values.container.name }}
      image: {{ .Values.container.image }}
      port: {{ .Values.container.port }}
```

Annotations in the original code are circled in red:

- Annotation 1: {{ .Values.name }}
- Annotation 2: {{ .Values.container.name }}
- Annotation 2.1: {{ .Values.container.image }}
- Annotation 2.2: {{ .Values.container.port }}

②

- Another use case: Deploy the same bundle of K8s YAML files **across multiple clusters**:



Values injection into template files

`values.yaml`

```
imageName: myapp
port: 8080
version: 1.0.0
```

default

`my-values.yaml`

```
version: 2.0.0
```

override values

```
helm install --values=my-values.yaml <chartname>
```

Values injection into template files

values.yaml

```
imageName: myapp  
port: 8080  
version: 1.0.0
```

default

my-values.yaml

```
version: 2.0.0
```

override values

result

```
imageName: myapp  
port: 8080  
version: 2.0.0
```

.Values object

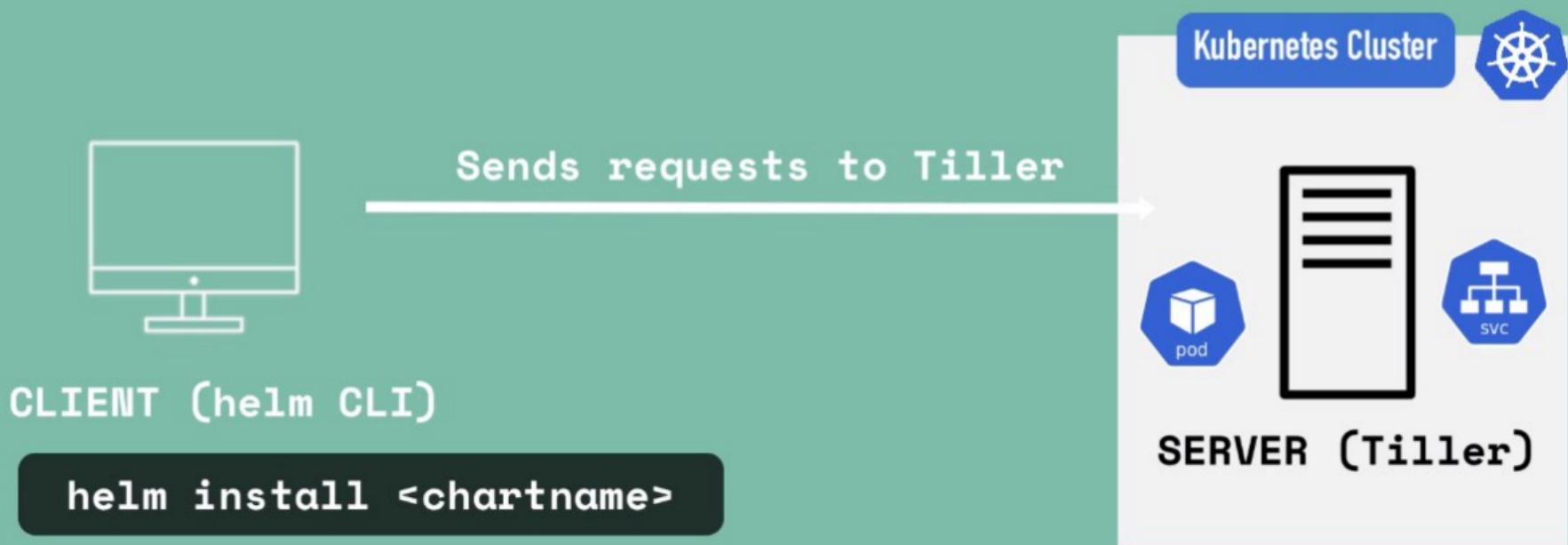
OR on Command Line:

```
helm install --set version=2.0.0
```

Release Management

Helm Version 2 vs. 3

Helm Version 2 comes in two parts:



Release Management

Keeping track of all chart executions:

Revision	Request
1	Installed chart
2	Upgraded to v 1.0.0
3	Rolled back to 1

```
helm install <chartname>
```

```
helm upgrade <chartname>
```

```
helm rollback <chartname>
```

- Changes are applied to existing deployment instead of creating a new one
- Handling rollbacks

Downsides of Tiller

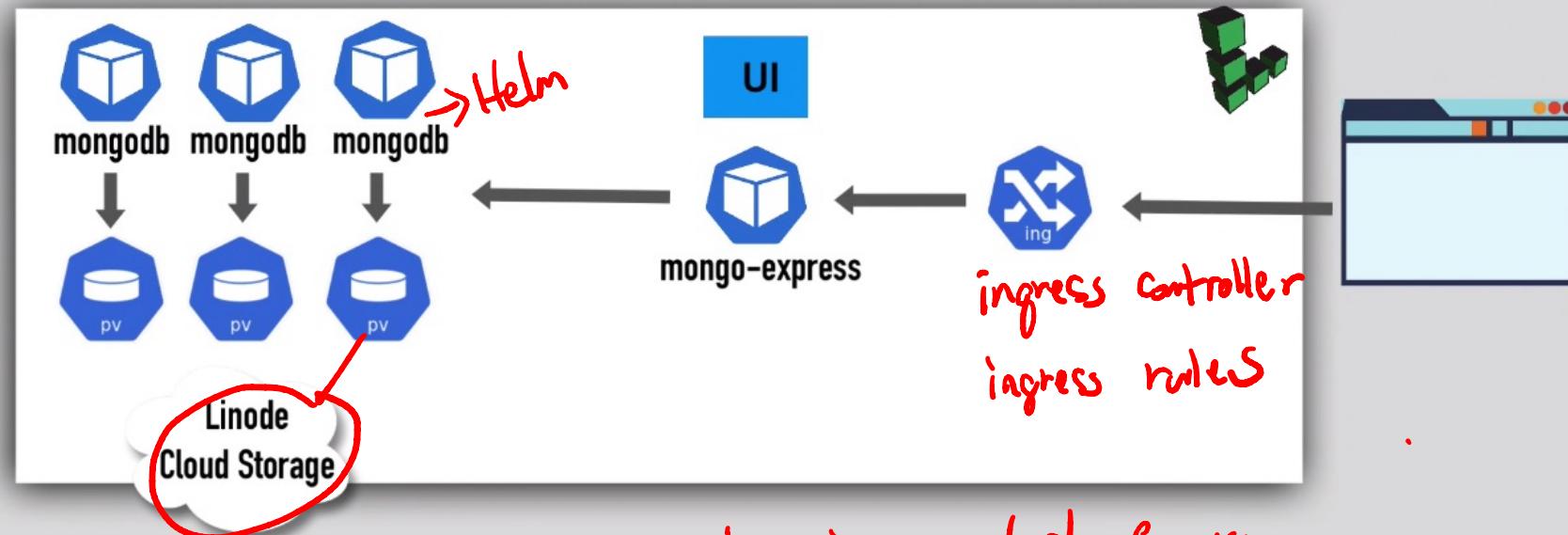
- Tiller has too much power inside of K8s cluster
 - Security Issue
 - Solves the Security Concern 
- In Helm 3 Tiller got removed!



Project

Overview of what we build/deploy 💪

► You will need this setup almost always for your K8s cluster



master-node is created for you

1. create k8s in Linode

create → k8s

cluster label

region

version latest

worknode

- Linode 4 GiB

- 2 nodes

You see dashboard + node pools
nodes green → running

linode

Dashboard

Linodes

Volumes

Object Storage

NodeBalancers

Domains

Marketplace

Longview

Kubernetes

StackScripts

Images

Account

Get Help

Search for Linodes, Volumes, NodeBalancers, Domains, Tags...

Create

NanaJanashia 0

Documentation

Kubernetes / test

Summary

Version 1.17

4 CPU Cores

Frankfurt, DE

8 GB RAM

\$40/month

160 GB Storage

Kubernetes API Endpoint:
https://efcede3e-34a7-49b0-ad19-77eac7d73984.eu-central-1.linodelke.net:443

+ Add a tag

Kubeconfig:
[test-kubeconfig.yaml](#)

local to cluster
credential for cluster

Add a Node Pool

Node Pools

Linode 4GB

Linode

Ike7209-9035-5f05b70e9fa1

Ike7209-9035-5f05b70f195f

Pool ID 9035

Resize Pool Delete Pool

Kubeconfig necessary to access cluster from your local machine

Running 139.162.188.203

2. download this file

3. set env variable for file = `export KUBECONFIG = test-kubeconfig.yaml`

完成这连结，只要设置这个 env

4. kubectl get node

24%
stu

5. deploy mongoDB

1. create all files | StatefulSet config file
2. use bundle of those config file Services
Other configuration) ↳ helm chart

6. search helm chart

↳ search mongo db helm chart (in bitnami)

7.

```
[Downloads]$ export KUBECONFIG=test-kubeconfig.yaml
[Downloads]$ kubectl get node
NAME           STATUS   ROLES      AGE    VERSION
lke7209-9035-5f05b70e9fa1   Ready    <none>    2m8s   v1.17.3
lke7209-9035-5f05b70f195f   Ready    <none>    2m7s   v1.17.3
[Downloads]$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
[Downloads]$ helm search repo bitnami/mongo
```

1. add repo

↳ repo mongo

```
for maintaining configura...
[Downloads]$ helm search repo bitnami/mongo
NAME          CHART VERSION APP VERSION DESCRIPTION
bitnami/mongodb 8.0.9       4.2.8      NoSQL document-oriented database that stores J
S...
bitnami/mongodb-sharded 1.5.5 4.2.8      NoSQL document-oriented database that stores J
S...
[Downloads]$
```

MongoDB parameters

env variable in chart

gitbook

Parameter	Description	Default
architecture	MongoDB architecture (standalone or replicaset)	standalone
useStatefulSet	Set to true to use a StatefulSet instead of a Deployment (only when architecture=standalone)	false
auth.enabled	Enable authentication	true
auth.rootPassword	MongoDB admin password	random 10 character long alphanumeric string
auth.username	MongoDB custom user (mandatory if auth.database is set)	nil
auth.password	MongoDB custom user password	random 10 character long alphanumeric string
auth.database	MongoDB c	Key used for the StorageClass of Linode's Cloud Storage
auth.replicaSetKey	Key used for the StorageClass of Linode's Cloud Storage	
auth.existingSecret	Existing secret with MongoDB credentials	nil
replicaSetName	Name of the replica set (only when architecture=replicaset)	rs0
replicaSetHostnames	Enable DNS hostnames in the replicaset config (only when architecture=replicaset)	true
enableIPv6	Switch to enable/disable IPv6 on MongoDB	false
directoryPerDB	Switch to enable/disable DirectoryPerDB on MongoDB	false

8. overwrite the default value using values.yaml

Untitled-1 test-mongo-express.yaml

test-mongodb.yaml

```

1 architecture: replicaset
2 replicaCount: 3
3 persistence:
4   storageClassName: "linode-block-storage"
5 auth:
6   rootPassword: secret-root-pwd

```

in persistence variable
 helm install this file
 ! helm install mongodb --name --values valuefile.yaml

g. we have three pods running

10. three PV in Linode volumes

The screenshot shows the Linode Volumes page. On the left, a sidebar menu includes Dashboard, Linodes, Volumes (which is selected), Object Storage, NodeBalancers, Domains, Marketplace, Longview, and Kubernetes. The main area displays a table of volumes:

Label	Region	Size	File System Path	Attached To
pvc27eb4242eb3f42f7	Frankfurt, DE	10 GiB	/dev/disk/by-id/scsi-0Linode_Volume_pvc27eb4242eb3f42f7	lke7209-9035-5f05b70e9fa1
pvc3111db9b503-4911	Frankfurt, DE	10 GiB	/dev/disk/by-id/scsi-0Linode_Volume_pvc3111db9b5034911	lke7209-9035-5f05b70f195f
pvc3bb8e9ff3f6543eb	Frankfurt, DE	10 GiB	/dev/disk/by-id/scsi-0Linode_Volume_pvc3bb8e9ff3f6543eb	lke7209-9035-5f05b70e9fa1

Below the table, there is a diagram illustrating the connection between the volumes and the MongoDB pods. Three blue cubes, each labeled "mongodb", are shown at the bottom. Arrows point from each cube to one of the three volumes listed in the table.

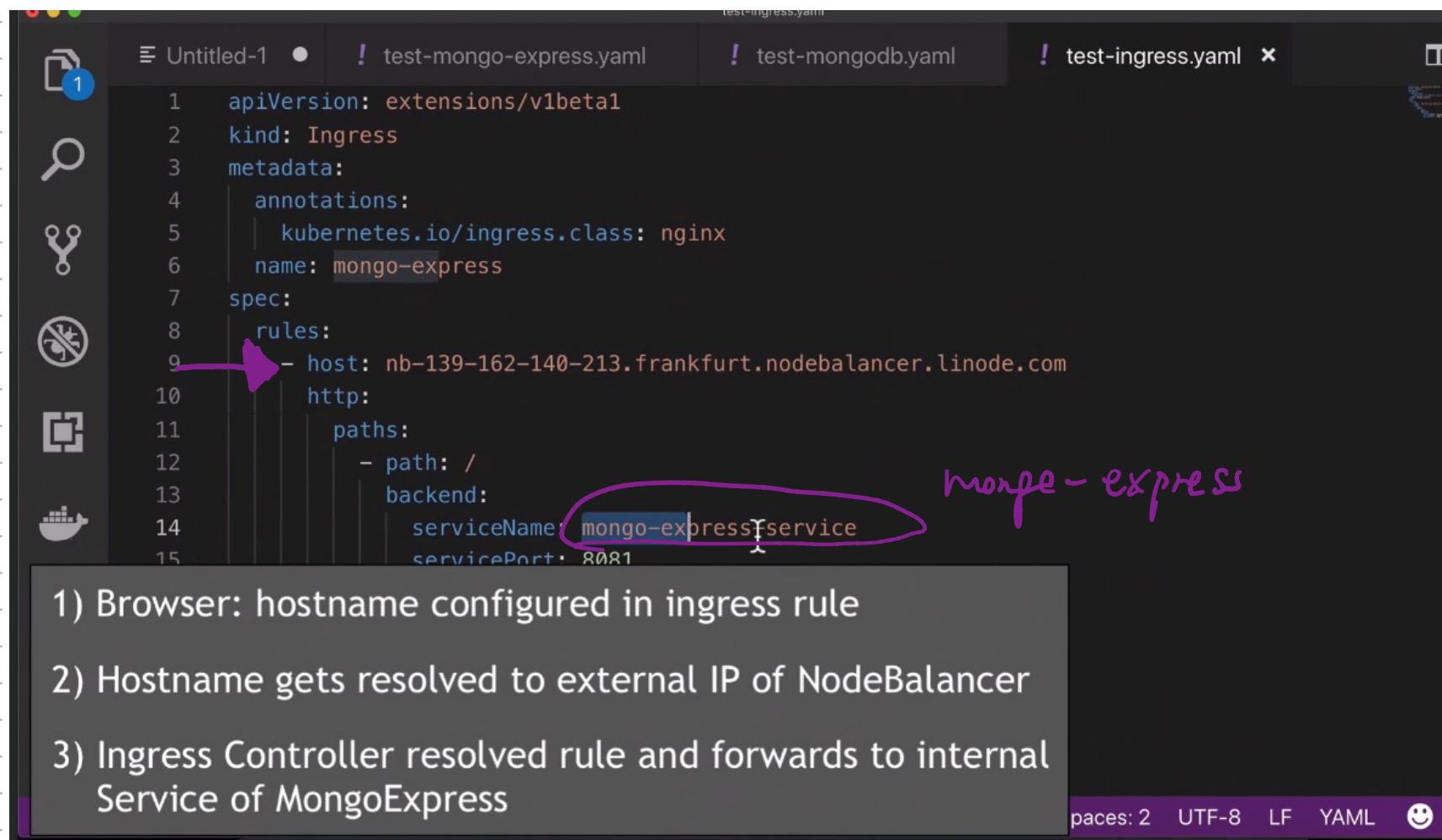
11. deploy mongo-express + mongo-express-service to k8s ↳ apply ...
 env ↳ internal service to k8s service
 So we need ingress ↳ endpoint ↳ kubectl log pod
 ↳ user passport
12. Install Ingress controller in the Linode cluster
 cloud native loadbalancer ↳ helm chart nginx-ingress ↳ add repo ingress + install ingress
13. check if ingress controller deployed → pod
 check in the nodebalance

The screenshot shows the Linode dashboard with the 'NodeBalancers' section highlighted. A large diagram is overlaid on the page, illustrating the deployment architecture. The diagram shows a 'public IP' at the top right pointing to a 'NodeBalancer'. Below the NodeBalancer is a 'mongo-express' service icon. To the left of the mongo-express service are three 'mongodb' icons, each connected to a 'pv' (Persistent Volume) icon. At the bottom left is a 'Linode Cloud Storage' icon. In the center, there is a 'UI' icon. Arrows indicate the flow of data from the public IP through the NodeBalancer to the mongo-express service, and then to the MongoDB instances.

14. Ingress service ← access from external

↓
IP address ^{SAME} = nodebalance IP

15. Create Ingress rule → create Ingress rule



```
test-ingress.yaml
Untitled-1 • ! test-mongo-express.yaml ! test-mongodb.yaml ! test-ingress.yaml ×

1  apiVersion: extensions/v1beta1
2  kind: Ingress
3  metadata:
4    annotations:
5      kubernetes.io/ingress.class: nginx
6      name: mongo-express
7  spec:
8    rules:
9      - host: nb-139-162-140-213.frankfurt.nodebalancer.linode.com
10        http:
11          paths:
12            - path: /
13              backend:
14                serviceName: mongo-express-service
15                servicePort: 8081
```

1) Browser: hostname configured in ingress rule
2) Hostname gets resolved to external IP of NodeBalancer
3) Ingress Controller resolved rule and forwards to internal Service of MongoExpress

paces: 2 UTF-8 LF YAML 😊

17. kubectl scale --replicas=0 Stateful mongoDB !
scale down

18. helm ls
helm unstall mongoDB
↳ pod is gone
↳ pr still there

deploy Images in k8s from private Docker repo

How to get Docker image on k8s cluster ?
private registry

2 Steps

Steps to pull image from private registry

1

Create Secret component



contains **credentials**
for Docker registry

2

Configure Deployment/Pod



method 1: 2 step
method 2: 1 step
login *deploy*

Use Secret using
imagePullSecrets

AWS Services Resource Groups

Administrator-Niki @ 6645-740... Frankfurt Support

Amazon Container Services

Amazon ECS

- Clusters
- Task definitions

Amazon EKS

- Clusters

Amazon ECR

Repositories

ECR > Repositories

Repositories (1)

C View push commands Delete Edit Create repository

Find repositories

Repository name	URI	Created at	Tag immutability	Scan on push
my-app	664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app	11/11/19, 10:15:03 AM	Disabled	Disabled

Already setup

- private docker repo hosted on AWS
- simple Nodejs application
(link in Checklist)

The screenshot shows the AWS ECR console with the repository 'my-app'. In the 'Options' section, three command-line arguments are listed:

Name, shorthand	Default	Description
--password , -p		Password
--password-stdin		Take the password from stdin
--username , -u		Username

→ minikube ssh
→ login to the image registry inside the minikube
→ ls -al
 - dock created - dock config.json
 ↓ 因此 create secret file

```
[~]$ scp -i $(minikube ssh-key) docker@$((minikube ip)):./docker/config.json ./docker/config.json
```

COPY config from minikube to host.
→ so that kubectl can use it. → then cat | base64

```
! docker-secret.yaml • ! my-app-deployment.yaml
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: my-registry-key
5 data:
6   .dockerconfigjson:  同名后添加此行
7 type: kubernetes.io/dockerconfigjson
8
```

Same

```
[~]$ kubectl create secret generic my-registry-key \
> --from-file=.dockerconfigjson=.docker/config.json \
> --type=kubernetes.io/dockerconfigjson
```

method 1
1. (直接image拉取)

method 2
1.

```
[~]$ kubectl create secret docker-registry my-registry-key-two \
> --docker-server=https://664574038682.dkr.ecr.eu-central-1.amazonaws.com \
> --docker-username=AWS \
> --docker-password=eyJwYXlsb2FkIjoiSkc5WG9WYkY1dDdEVnR3eUxHSWJzOTRVQXprS1RBTKEyL1dYRTRGVVpwKzhr
55cHZCc21zUS9hV1pDYVJKVnZad21qR0JyaUcyRlpoTGZjRW9yRVFRNmFvSC9MQ3R3RncvRFVVT11QZHJnWkptb0dvSnd4el
TFA0cFFJMXRvSUNaRFhNSzJPdWhzYURyVFRNWGM3aDBCZWN3M1Jvd2Z20VM0Q1BLTWRhb93dmpnWXFDTkR4NkVtZXh6S3By
1vZTZkTHd1U2NzcWP6ZDBPS3BiWXRUYZlqrY0MydGlxNTBUjhY4aGs1bW5YalF0dGxmNDZkNmNCUktqU3NOSDA5ZzllYzJLZ
N2d1QVFGcjJFajVaRjFpSUhUN2hQM31TQnFTWHLscmFnSWFRdU9vS1JPT2krZk5TZWvaTN1bnZEMkcrMlhuLzNndkQrZEJ4
plsUtXdGJ6KzJZQTBjYzFKZ3g0cTzZQ1I0eXR1c0h4VEc2aHp0TjgvSWpjWHIVbFJSbDhHMDZBUzhMVWwrN1cvdkZMbDBXUk
OHZJY3NhK3pMRnFWOG10Lz16NTZQbjU4TzR6RHdiYytGWS8rTWmWQjNsM1Y4bHV3TytDM0tMeF1YdVVPS0dnatZ4K0pzN1Z4
NlaVBjcnhyNFLwB1FpUhpwVUg3R1VVY1BrdHkyY1dFL0x2d2FqSURad1M2Mko5SGVJN3FCMUZOZ1pEMzJ3ZFdrRFZOMmpRbC
QURCR2V2VnFrTVo0QThTaXQxQTFIUDVXTHI2QVdpL2pTb1FrUTZScT14MU90SXRMZWkrQWU3Sk1tMnF0cGs2NDYydjMwanZp
JTMVdJcnloTm1zTEkyZ3p3a0FZb3ZKRUtYRTBzamtqaDhTQ1duaUhBVGNJ0XJNRFNtY11RbGpmZGJIWEhNYTBVF2pLQTYycv
c1ZsbHhNOWtsN1R3QVJ4dG41Vy9RcWRj0WZldWppSGk3aktLZVVuZmhUVVJzem9QTDhySjhCcC9rZ25zOFpCR0gwR2FYyntJ
JvRWpzYTQzcTBqY0NmYwK2akNLUktWDAt5TUtwMTZoR1ZvRVFzR2tsZ0wxUEV2TzJoL2MxUURKskNtR2IyeWpZTRUTnNHc0
Z0ZBVXhPanBzQnRGMppmM1QxenRzdHVBQ01kOWZReitMeU5PLzJqVzhzb2dQYzQxU2tSZ243MKFOQ2NO0FMrQm9uckZWRUF2
FUMWxoOUFFXV1B1MUpJNXpaNDhYN2hSR1d0QVVHbkJNZ3dBVUhYeFBNRHUxZ3FiSzFYTJER1VWMTFnT3Axc1BDN3dIVi9Kbu
cUN1RmsrT3dSUFTmk9CwMzr2NFTjIxakxyRTF5Qit3UUhwbmJvOFZmWVvxZ24xRG9DSnNwcFhGow9EZGQwR0xsaXJENU90
htUEt10EdGUExqVUpubzZuZzVQDQ05jTUVHVEDLK3d2en1FvjcrV25wdFVpRG5LMFkwTm5W0WRxckJmT1lsRW03ej5kQzEwP
dGFrZXki0iJBUVVCQUhoM3pZT2R0cEJCU1Z3L1kyYW44Q1hJRDRrd0xRZm5vVG01endadEdKwm9aUUFBQUG0d2ZBWUpLb1p
FRY0dvRzh3Y1fJQkFEQm9CZ2txaGtpRzl3MEJCd0V3SGdzS11JWk1BV1VEQkFFdU1CRUVES21FK01sT11xWURPV1VZRmdJQ
VHoya3B1eUsrWDRQMzJoQUFBbWx0YUhNSExDd1RLWjFIQXQvRmhXN0tub3VkejNzVk9QeDFXZjZEVNEYutnRHhJQVlteDd
hVPSISInZlcNpb24i0iIyIiwidHlwZSI6IkRBVEffs0VZIiwiZxhwaXJhdG1vbii6MTU5MDY40DkxNx0=
```

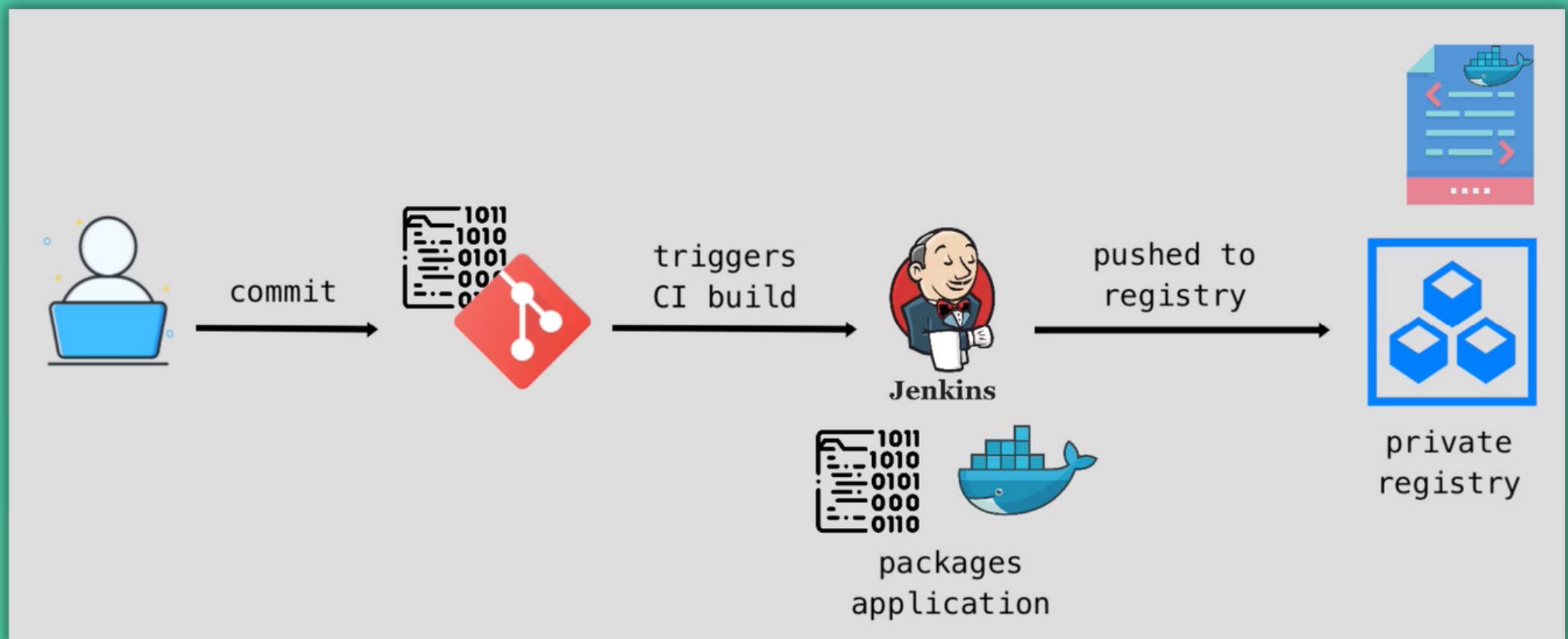
! docker-secret.yaml

```
5   labels:  
6     app: my-app  
7 spec:  
8   replicas: 1  
9   selector:  
10    matchLabels:  
11      app: my-app  
12 template:  
13   metadata:  
14     labels:  
15       app: my-app  
16 spec:  
17   imagePullSecrets: [ ] } → secret so you can pull  
18     - name: my-registry-key  
19 containers:  
20   - name: my-app  
21     image: 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.3  
22     imagePullPolicy: Always ↓ image rep  
23   ports:  
24     - containerPort: 3000 ↴ always pull
```

! my-app-deployment.yaml x

CD - Jenkins & Kubernetes

Pull Docker Images into K8s cluster



- **PRIVATE REPO:** For K8s to fetch the Docker Image into K8s cluster, it **needs explicit access**



- **PUBLIC REPO:** For public images, like mongodb etc. pulled from public repositories **no credentials needed**



Steps to pull image from Private Registry

1 - Create Secret Component

- Contains credentials for Docker registry



```
|my-registry-key      kubernetes.io/dockerconfigjson    1|
```

- docker login token is stored in dockerconfig file

2 - Configure Deployment

- Use Secret using imagePullSecrets

```
6   |   app: my-app
7   spec:
8     replicas: 1
9     selector:
10    matchLabels:
11      app: my-app
12    template:
13      metadata:
14        labels:
15          app: my-app
16    spec:
17      containers:
18        - name: my-app
19          image: 664574038682.dkr.ecr.eu-central-1.amazonaws.com
20          imagePullPolicy: Always
21          ports:
22            - containerPort: 3000
```



Kubernetes Operators

Kubernetes Operators - 1

- Stateful applications need **constant management and syncing after deployment**. So stateful applications, like database need to be operated
- Instead of a human operator, you have an **automated scripted operator**



Operators

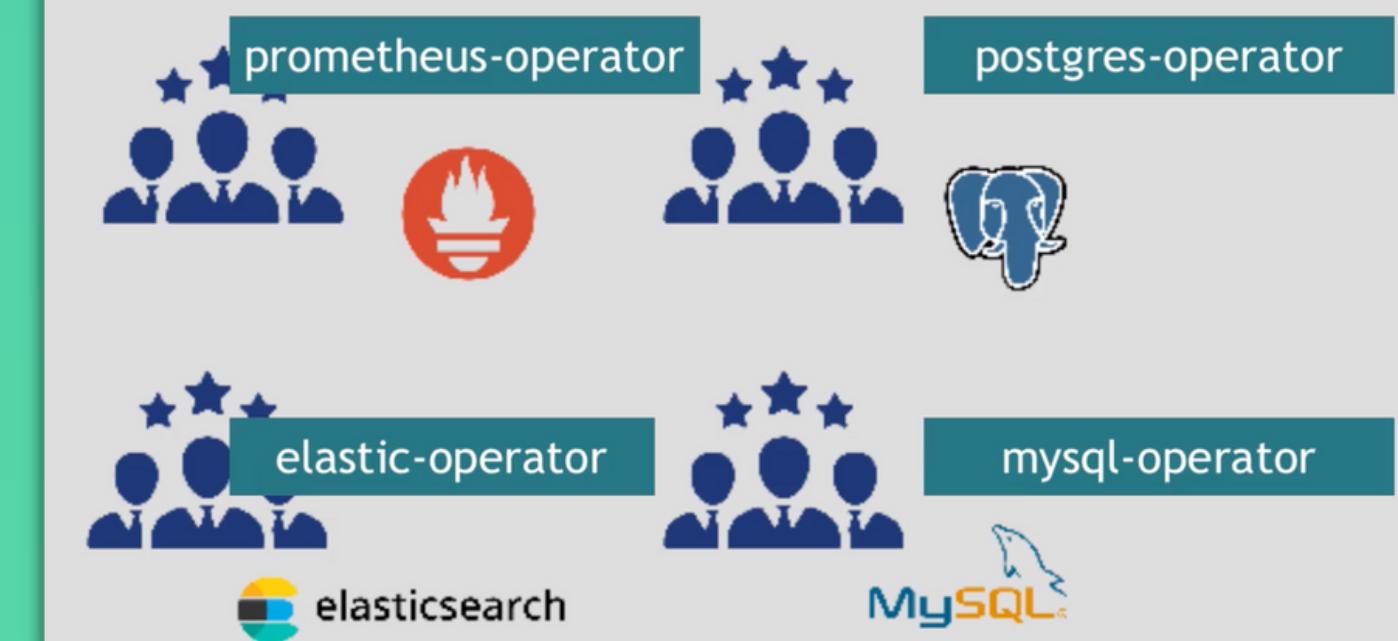
- Stateless applications:

Is **managed by Kubernetes**

- Stateful applications:

K8s can't automate the process natively

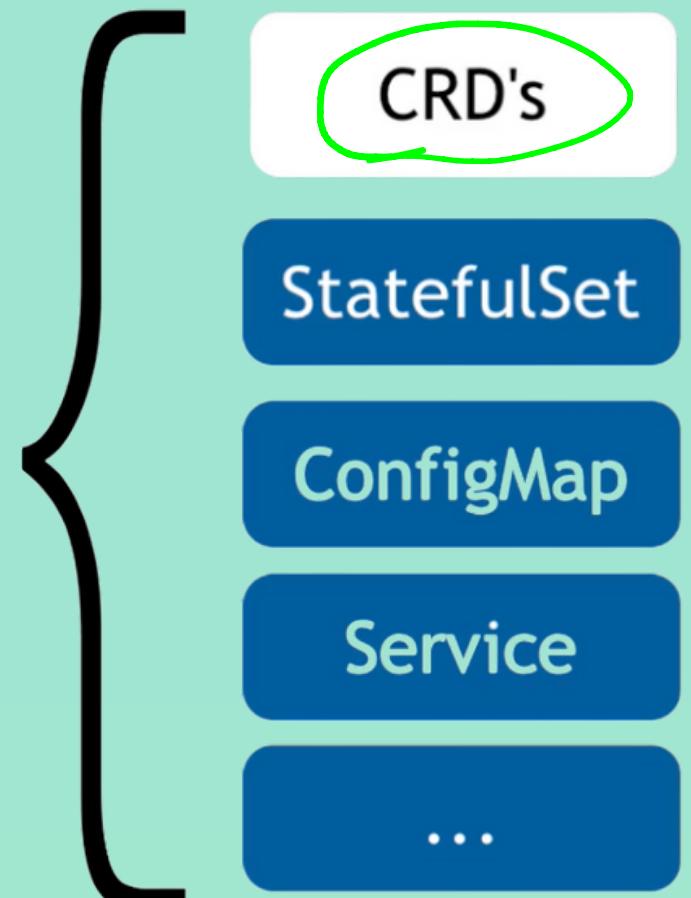
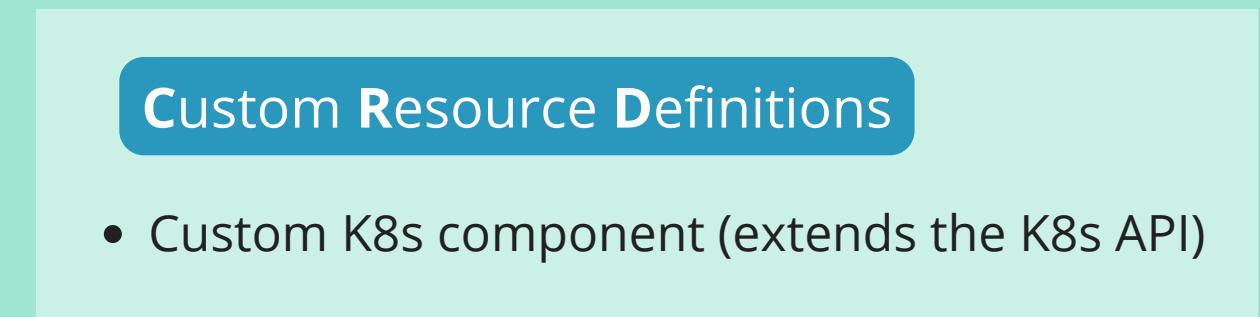
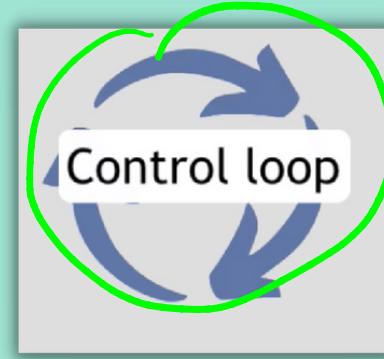
Operators are created
by official maintainers



Kubernetes Operators - 2

How it works:

- Control loop mechanism
- Makes use of CRD's
- Include domain/App-specific-knowledge, e.g. mysql:
 - How to create mysql cluster
 - How to run it
 - How to synchronize the data
 - How to update

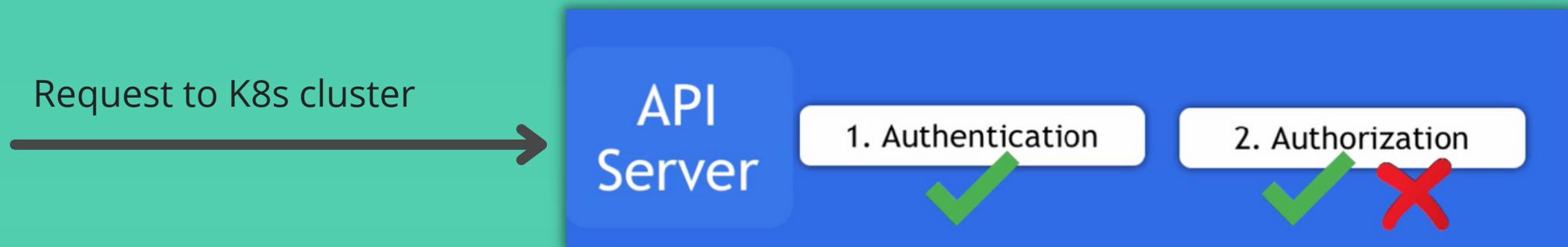


Who create operator?
Operator Hub
Operator SDK (To create)

Secure your Cluster - Authentication & Authorization

Layers of Security

- In K8s, you must be **authenticated** (**logged in**) before your request can be **authorized** (**granted permission to access**)



Authentication

- K8s doesn't manage Users natively
- No K8s resources exist for representing normal user accounts
- Admins can choose from different authentication strategies

Authorization

- K8s supports multiple authorization modules, such as ABAC mode, RBAC Mode and Webhook mode
- On cluster creation, admins configure the authorization modules that should be used
- K8s checks each module, and if any module authorizes the request, then the request can proceed

Authentication

- API server handles authentication of all the requests
- Available **authentication strategies**:



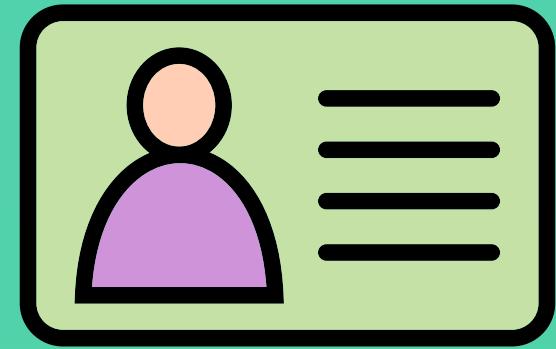
Client Certificates



Static Token File



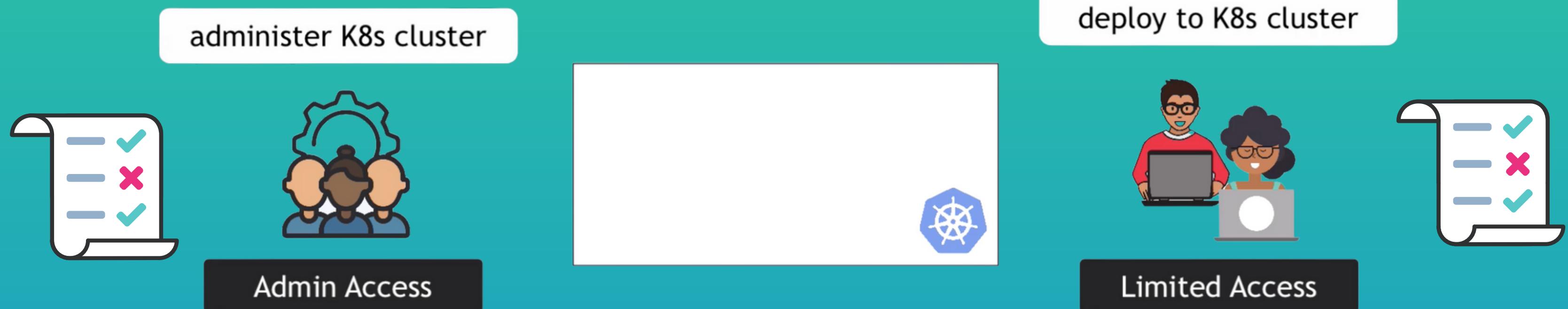
3rd Party Identity Service, like LDAP



- A csv file with a minimum of 3 columns: token, user name, user uid, followed by optional group names

Authorization

- As a security best practice, we only want to give people or services **just enough permission** to do their tasks: **Least Privilege Rule**



- Admins need cluster-wide access to do tasks like configure namespaces etc.

- Developers need only limited access for example to deploy applications to 1 specific namespace

Authorization with RBAC - 1

Role-based access control (RBAC)

- A method of regulating access to resources based on the roles of users within the organization
- Enabling RBAC: `kube-apiserver --authorization-mode=RBAC --other-options`
- 4 kinds of K8s resources: ClusterRole, Role, ClusterRoleBinding, RoleBinding

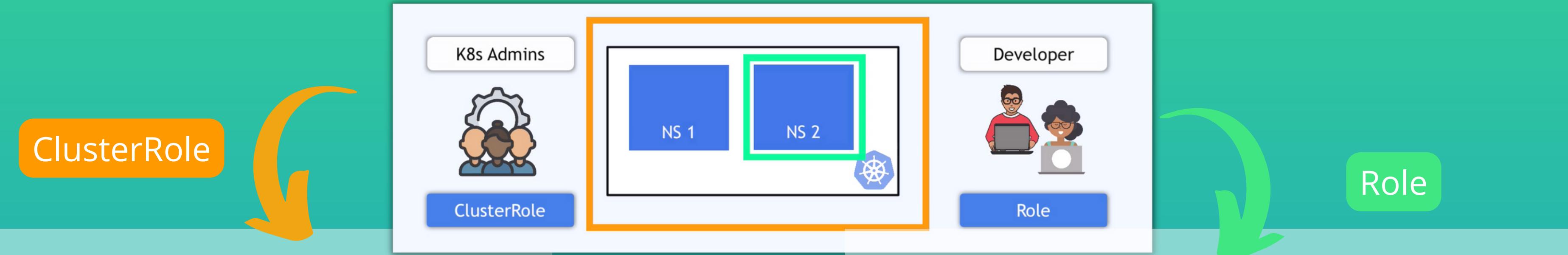
Role and ClusterRole

- Contains rules that represent a **set of permissions**
- Permissions are additive

RoleBinding and ClusterRoleBinding

- **Link ("Bind")** a Role or ClusterRole to a **User or Group**

Authorization with RBAC - 2

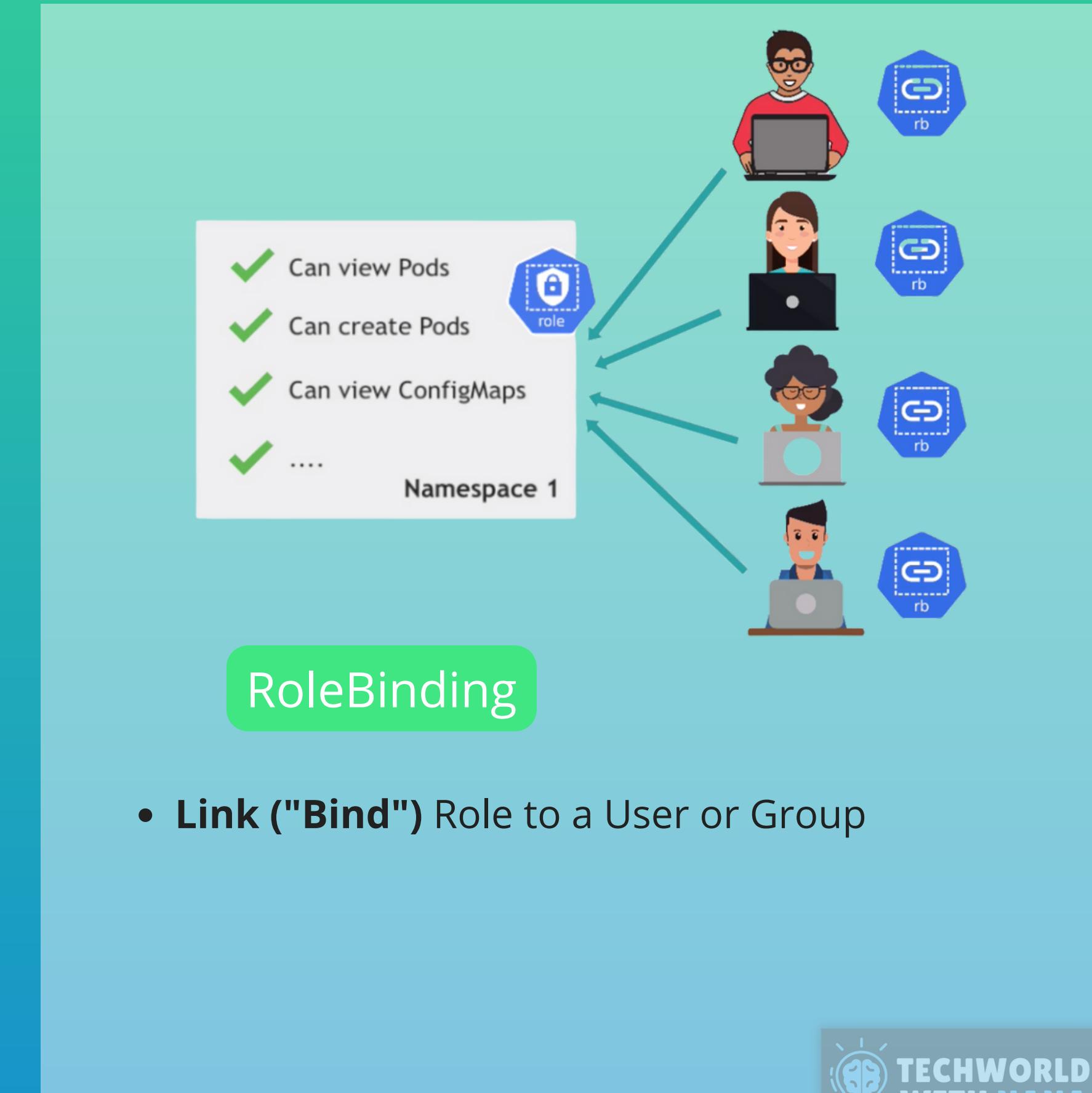
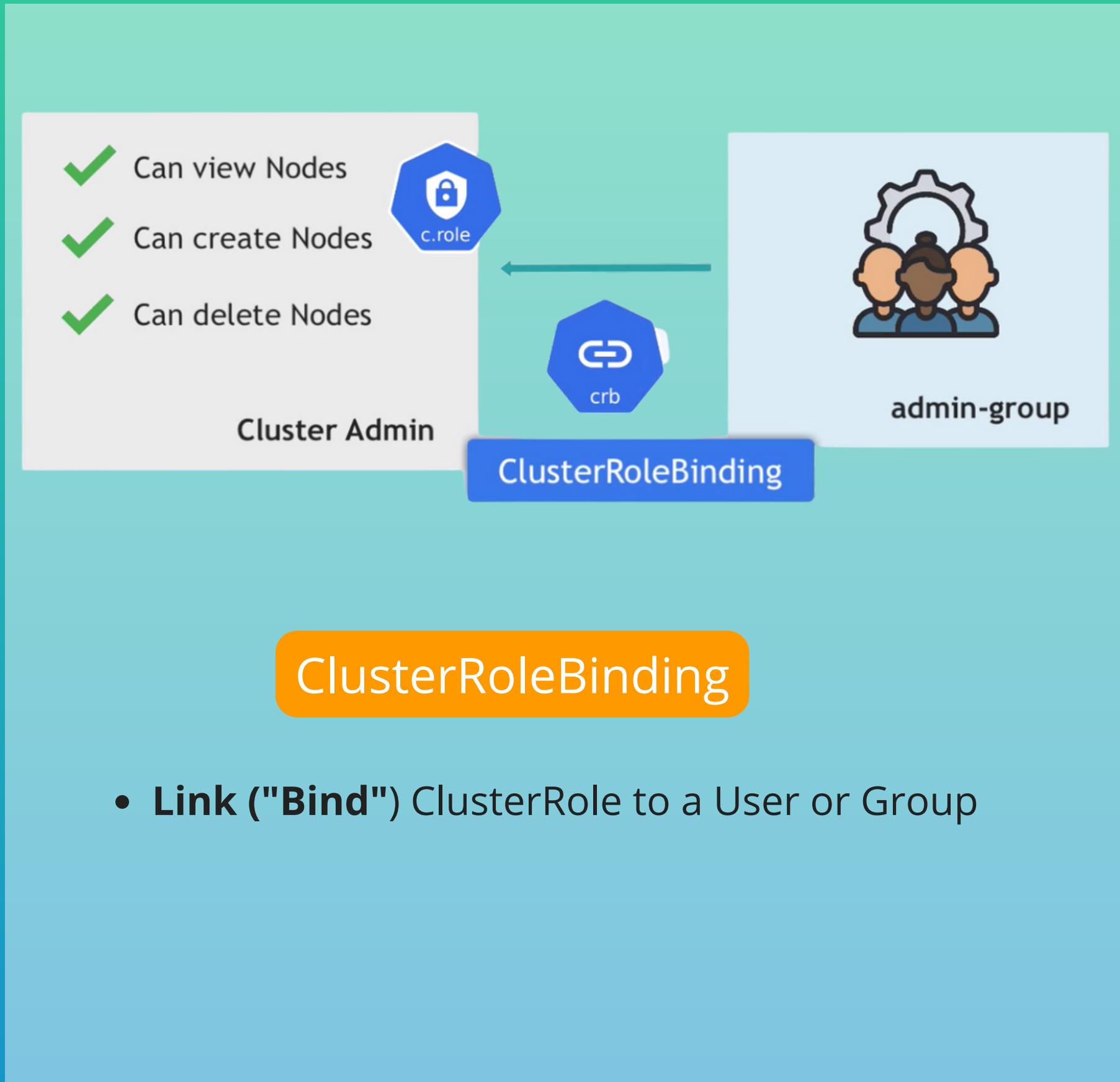


- Define permissions **cluster wide**
- For example: configure cluster-wide volumes, namespaces

```
● ● ●  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
  name: developer  
rules:  
- apiGroups: [""]  
  resources: ["pods"]  
  verbs: ["get", "create", "list"]  
- apiGroups: [""]  
  resources: ["secrets"]  
  verbs: ["get"]
```

- Define **namespaced** permissions through Role
 - Bound to a specific namespace
 - **What resources** in that namespace you can access
 - **What action** you can do with this resource

Authorization with RBAC - 3

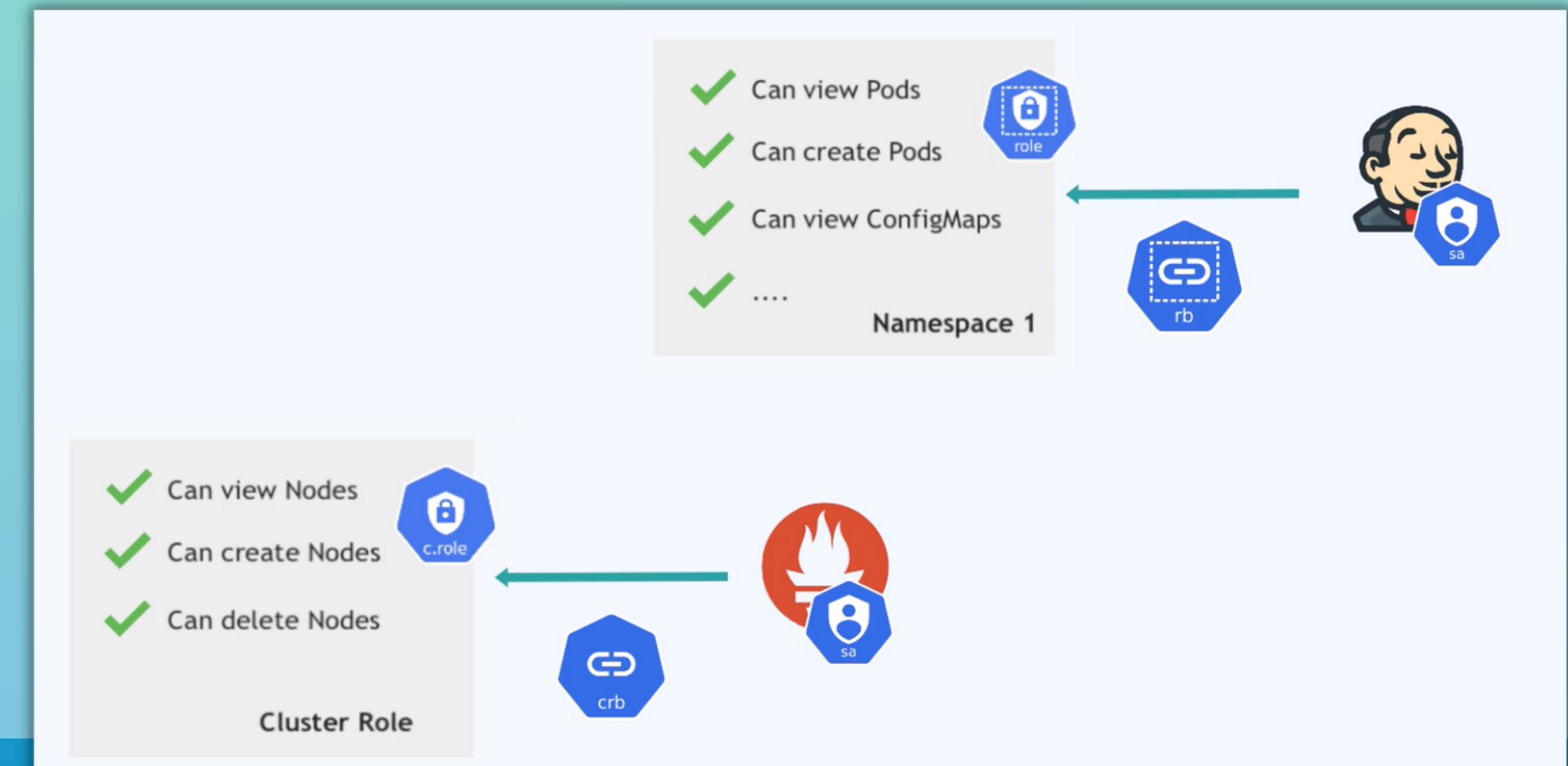


Authorization for Applications

- **ServiceAccounts** provide an identity for processes that run in a Pod, for example Jenkins or Prometheus



- A RoleBinding or ClusterRoleBinding can also bind a role to a **ServiceAccount**



"Role" Configuration File

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "create", "list"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
```

What resources have what kind of access?

apiGroups

- `""` indicates the core API group

resources

- K8s components like Pods, Deployments etc.

verbs

- The actions on a resource
- "get" "list" (read-only) or
"create", "update" (read-write)

"Role" Configuration File

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
  namespace: Default Namespace
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "create", "list"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
```

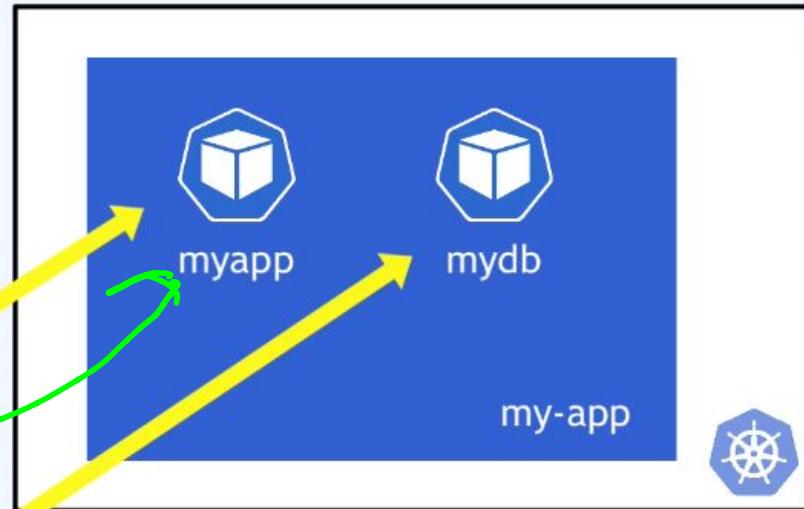
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-app
  name: developer
rules:
- apiGroups: [""]
```





```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-app
  name: developer
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "create", "list"]
    resourceNames: ["myapp"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["list"]
    resourceNames: ["mydb"]
```

Set Access more granular



"RoleBinding" Configuration File

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jane-developer-binding
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer
  apiGroup: rbac.authorization.k8s.io
```

Bind Subject



Bind Subject

```
subjects:
- kind: Group
  name: "devops-admins"
  apiGroup: rbac.authorization.k8s.io
```



Group

name: "devops-admins"

apiGroup: rbac.authorization.k8s.io



ServiceAccount

name: default

namespace: kube-system

"ClusterRole" Configuration File



```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "create", "list", "delete", "update"]
```

"ClusterRole" Configuration File

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin
rules:
- apiGroups: []
  resources: ["namespaces"]
  verbs: ["get", "create", "list", "delete", "update"]
```

Creating & Viewing RBAC Resources

developer-role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-app
  name: developer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "create", "list"]
  resourceNames: ["myapp"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list"]
  resourceNames: ["mydb"]
```

- ▶ Create Role, ClusterRole etc. just like any other Kubernetes component

kubectl apply -f developer-role.yaml

- ▶ View Components with *get* and *describe* command

kubectl get roles

kubectl describe role developer

Checking API Access

- ▶ Kubectl provides a *auth can-i* subcommand
- ▶ To quickly check if current user can perform a given action

```
kubectl auth can-i create deployments --namespace dev
```

- ▶ Admins can also check permissions of other users



Layers of Security



Request to:
Create new Service in default namespace

Request to API Server

API
Server





Layers of Security

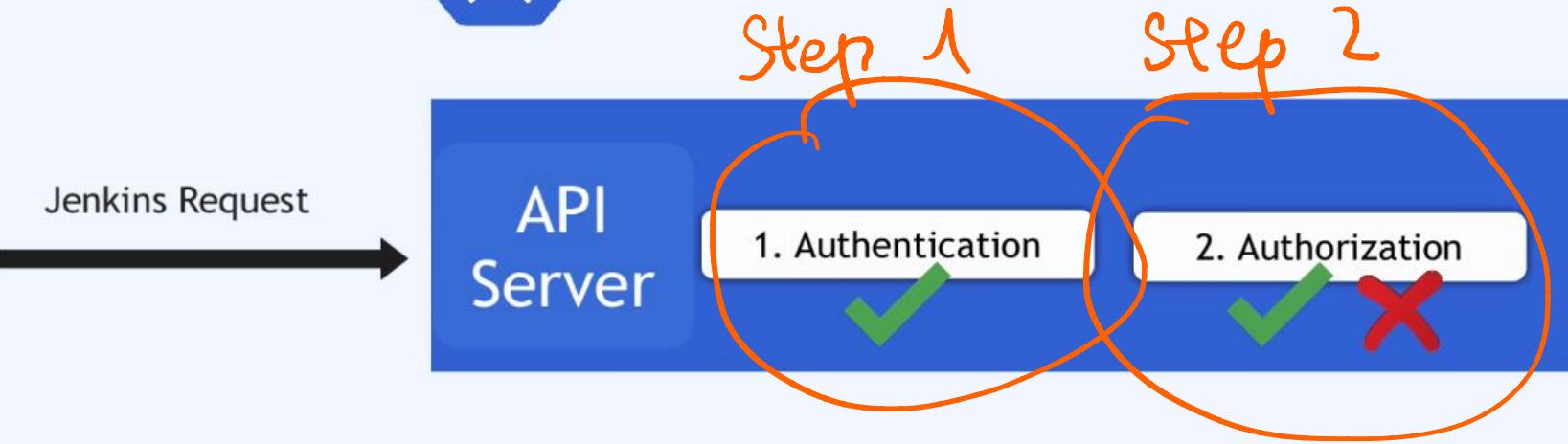
Jenkins Request



- ▶ API Server checks if Jenkins User is authenticated
- ▶ Is Jenkins **allowed** to connect to the cluster at all?
- ▶ You can enable multiple authentication methods at once



Layers of Security



- ▶ RBAC is one of multiple Authorization Modes
- ▶ With RBAC: Role, ClusterRole and Bindings are checked

Microservices in Kubernetes

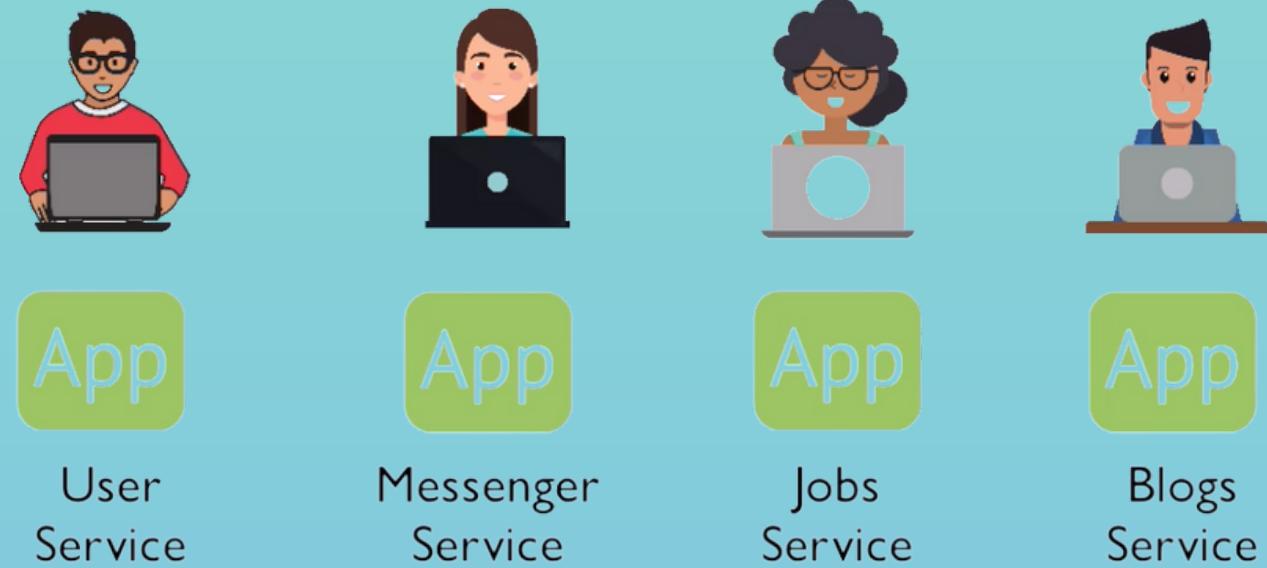
Introduction to Microservices

- Microservices are an architectural approach to software development
- Software is **composed of small independent services** (instead of having a huge monolith)
- Each business functionality is **encapsulated** into own Microservice (MS)



Benefits

- Each MS can be developed, packaged and released **independently**
- Changes in 1 MS doesn't affect other MS
- Less interconnected logic, **loosely coupled**
- Each MS can be developed by separate developer teams

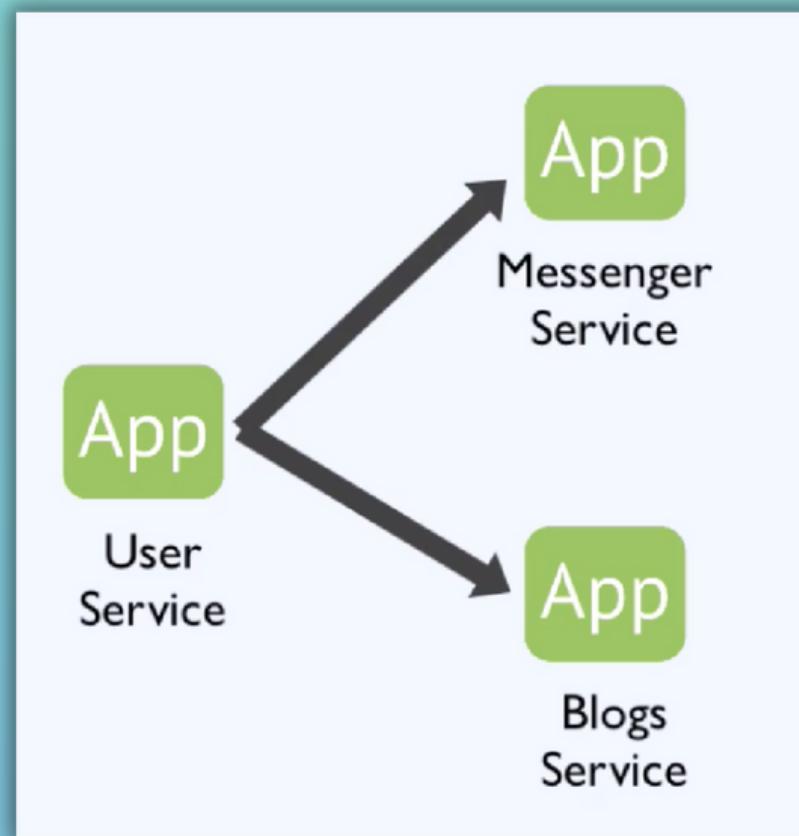


Communication between Microservices

- Communication between those services can happen in different ways:

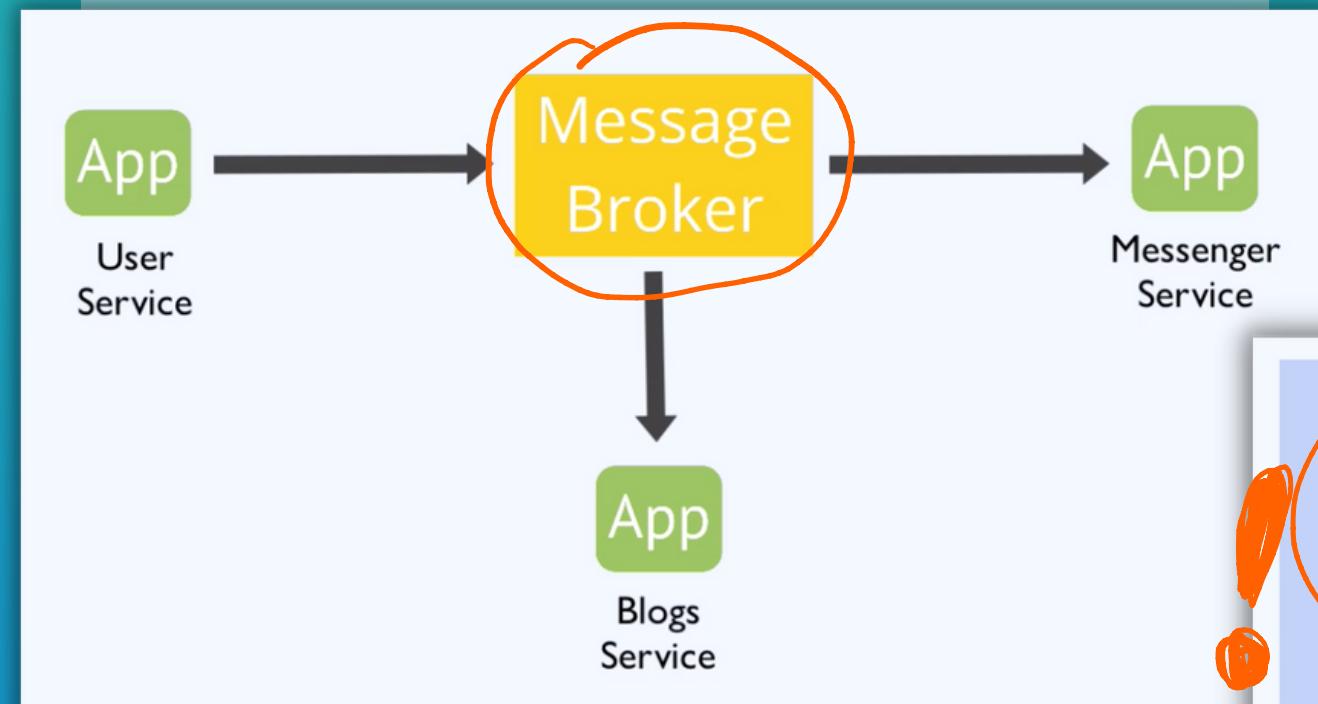
Service-to-Service API Calls

- Direct communication



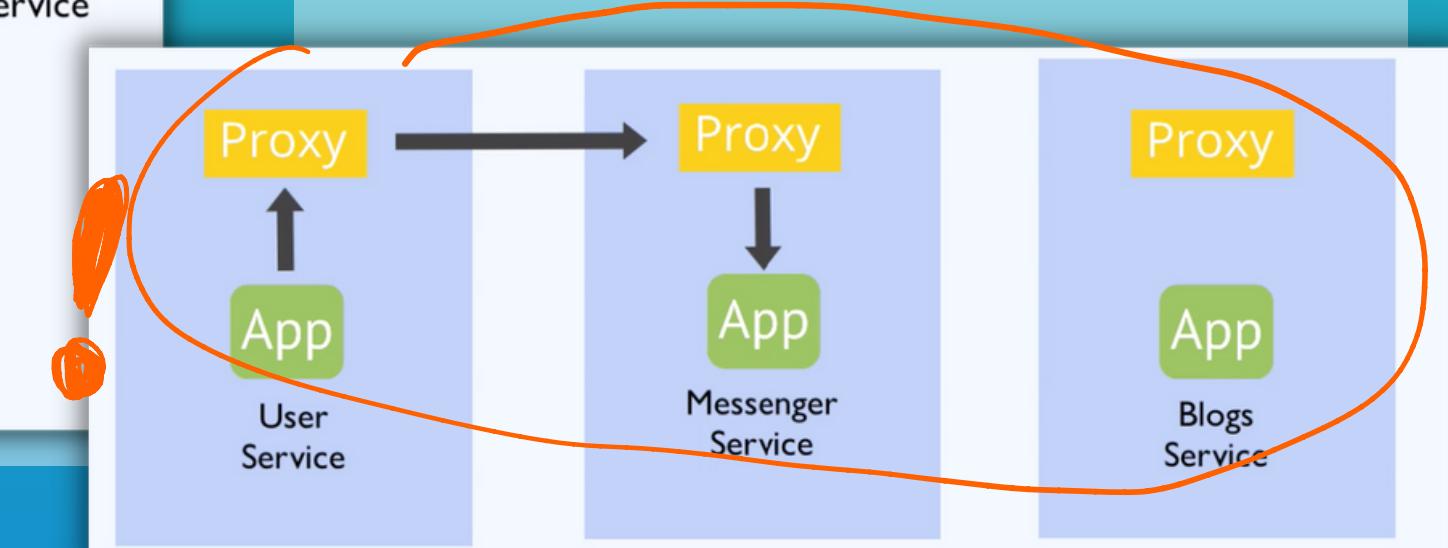
Message-Based Communication

- Communication through a message broker, like Rabbitmq or Redis

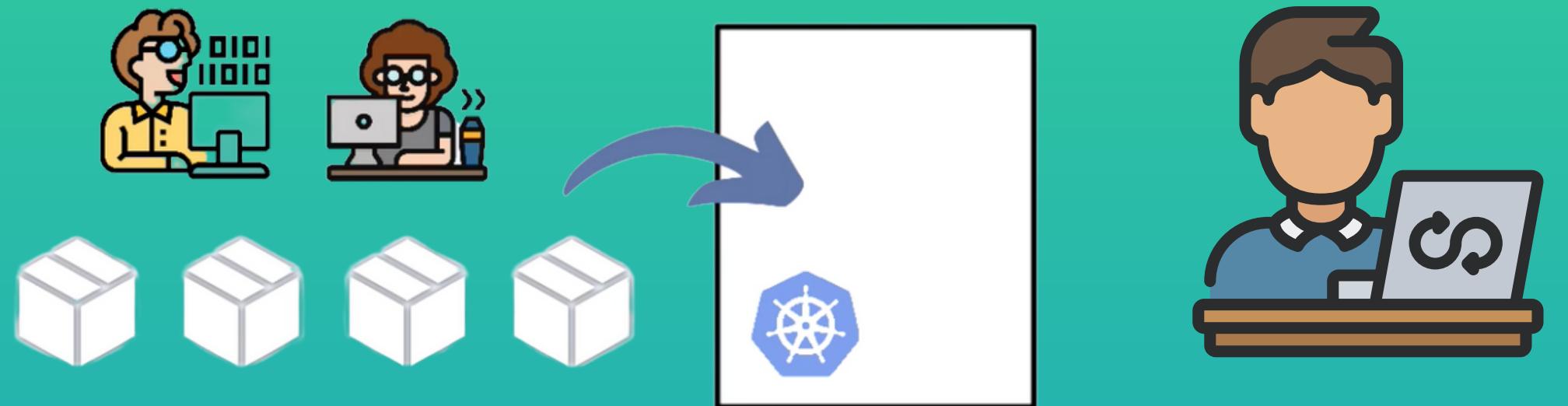


Service Mesh Architecture

- Platform layer on top of the infrastructure layer, like Istio, Linkerd, HashiCorp Consul
- Enables managed, observable and secure communication



DevOps Task: Deploy Microservices App - 1



- Developers develop the microservice applications
- As a DevOps engineer your task would be to **deploy the existing microservices application** in a K8s cluster

Information you need from developer:

1. Which services you need to deploy?
2. Which service is talking to which service?
3. How are they communicating?
4. Which database are they using? 3rd party services
5. On which port does each service run?

DevOps Task: Deploy Microservices App - 2



HOW TO

1. Prepare K8s environment

- a. Deploy any 3rd party apps
- b. Create Secrets and ConfigMaps for microservices

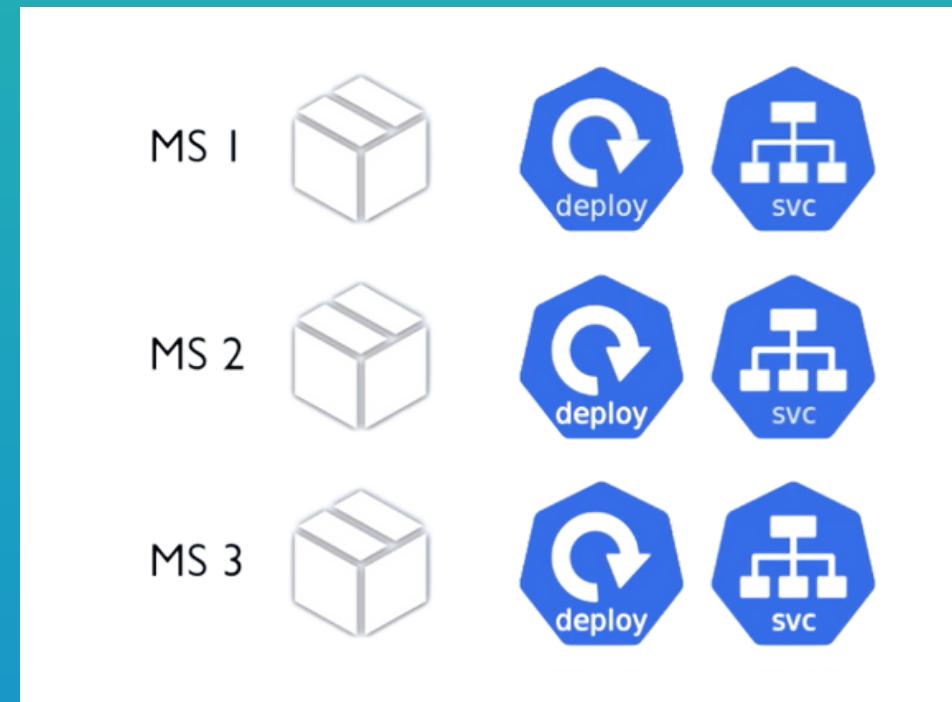
2. Create Deployment and Service for each microservices



1)



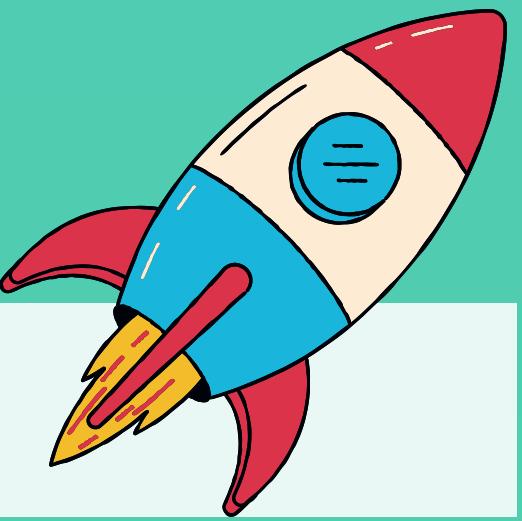
2)



When deploying multiple similar services to K8s, you can **use helm chart with 1 common template and replace specific values for each service on the fly** during deployment

Kubernetes Production & Security Best Practices

Best Practices - 1



- Specify a pinned version on each container image

Why? Otherwise, latest version is fetched, which makes it unpredictable and intransparent as to which versions are deployed in the cluster

- Configure a liveness probe on each container

Why? K8s knows the Pod state, not the application state. Sometimes pod is running, but container inside crashed. With liveness probe we can let K8s know when it needs to restart the container

- Configure a readiness probe on each container

Why? Let's K8s know if application is ready to receive traffic

↳ readiness and liveness

Ready
0/1
1/1 ← Container running / pod running

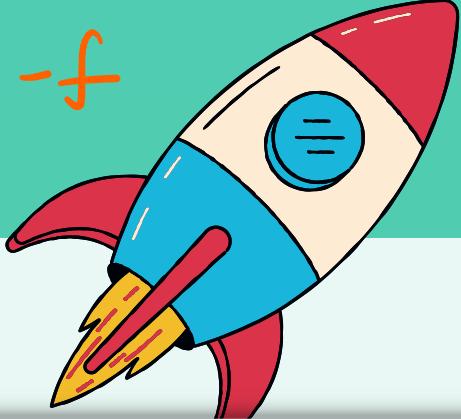
```
spec:  
  containers:  
    - name: service  
      image: gcr.io/google-samples/microservices-demo/emailserv  
      ports:  
        - containerPort: 8080  
      env:  
        - name: PORT  
          value: "8080"  
      livenessProbe:  
        ① periodSeconds: 5  
        ② exec:  
          command: ["/bin/grpc_health_probe", "-addr=:8080"]  
      readinessProbe:  
        periodSeconds: 5  
        exec:  
          command: ["/bin/grpc_health_probe", "-addr=:8080"]  
application executable
```

to run: Command
TCP socket
HTTP

0/1 ← application cannot start → check with

Best Practices - 2

kubectl logs Name -f

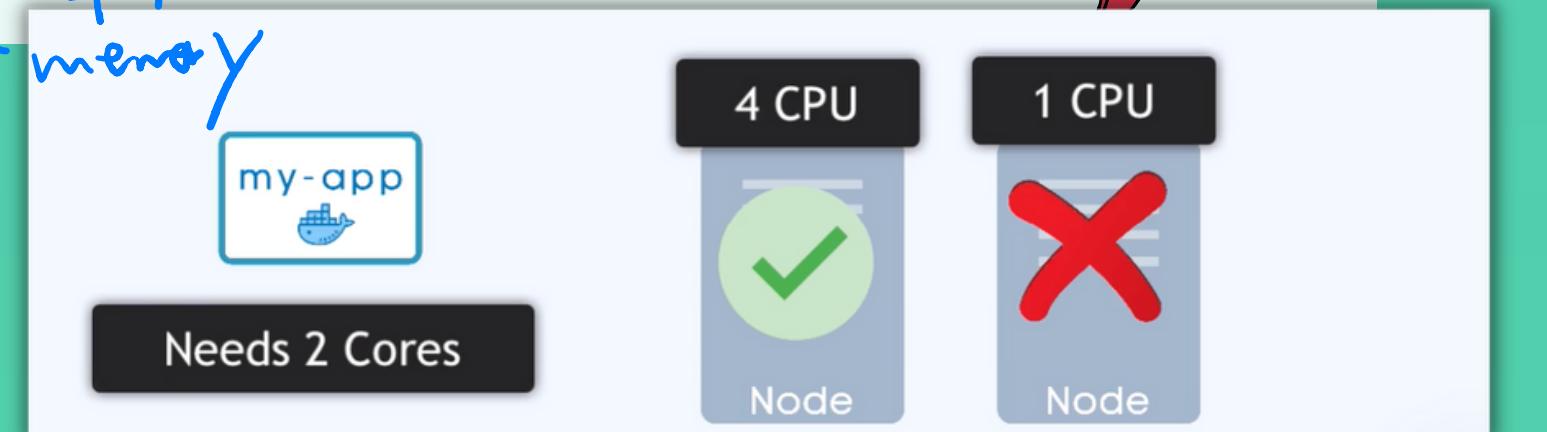


- Configure resource limits & requests for each container

Resources → request → CPU
memory
CPU
memory
limit

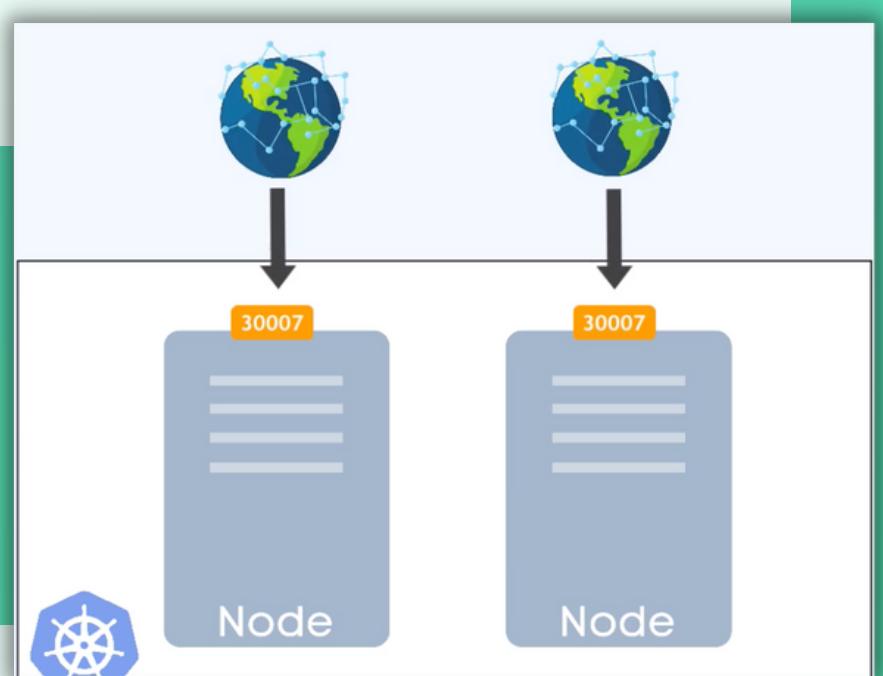
Why? To make sure 1 buggy container doesn't eat up all

resources, breaking the cluster



- Don't use NodePort in production

Why? NodePort exposes Worker Nodes directly, multiple points of entry to secure. Better alternative: Loadbalancer or Ingressss

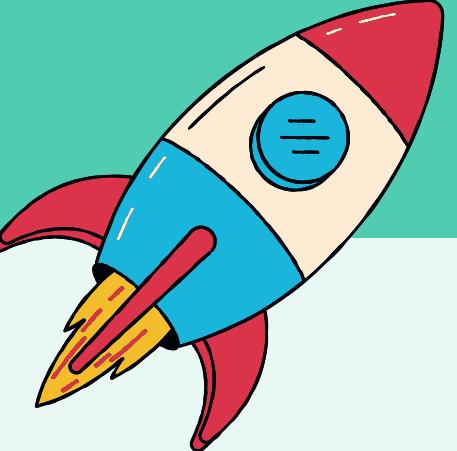


- Always deploy more than 1 replica for each application

Why? To make sure your application is always available, no downtime for users!

replicas = 2

Best Practices - 3



- Always have more than 1 Worker Node

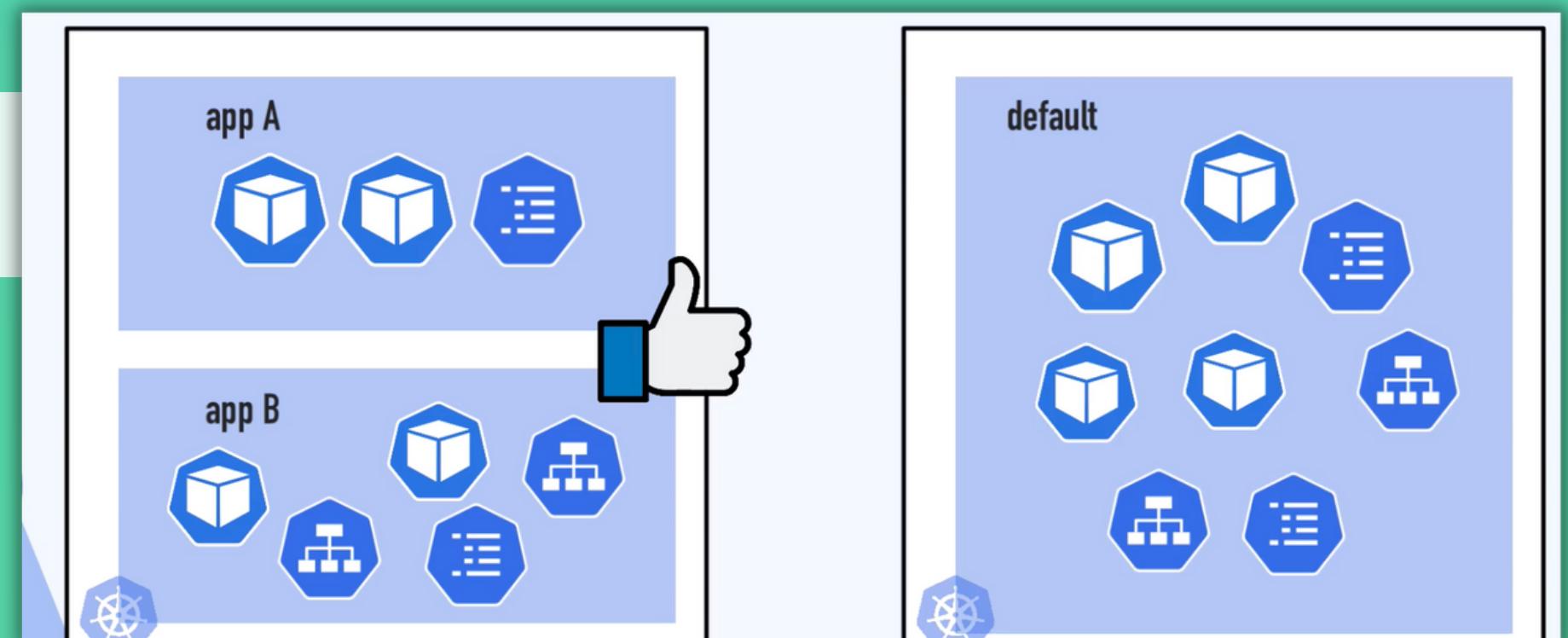
Why? Avoid single point of failure with just 1 Node

- Label all your K8s resources

Why? Have an identifier for your components to group pods and reference in Service e.g.

- Use namespaces to group your resources

Why? To organize resources and to define access rights based on namespaces e.g.



Security Best Practices

- Ensure Images are free of vulnerabilities

Why? Third-party libraries or base images can have known vulnerabilities. You
can do manual vulnerability scans or better automated scans in CI/CD pipeline



- No root access for containers

Why? With root access they have access to host-level resources. Much more damage possible, if container gets **hacked!**

```
YAML
spec:
  containers:
    - name: app
      image: nginx:1.19.8
      securityContext:
        privileged: true
```

A screenshot of a YAML configuration file for a Kubernetes pod. The 'securityContext' section is highlighted with a green box, and the 'privileged: true' line is specifically highlighted with a yellow box. This indicates that the original configuration allowed root access, which is being corrected by setting 'privileged: false'.

- Keep K8s version up to date

Why & How? Latest versions include patches to previous security issues etc.

Upgrade with zero downtime by having multiple nodes and pod replicas on different nodes

~~Set env name for minikube kubectl --alias kubectl="minikube kubectl --"~~

1. minikube start --vm-driver=hyperkit **vm** hyperkit for MacOS

2. Install minikube: dependency like kubectl is also installed.

Interaction with k8s cluster: **kubectl**

Configure anything is talk with Api Server

1. minikube start --vm-driver=docker (minikube has docker runtime preinstalled)

```
(base) gu@gu-GE60-2PC:~$ minikube start --vm-driver=docker
  😊 minikube v1.26.0 auf Ubuntu 20.04
  ✨ Verwende den Treiber docker basierend auf dem existierenden Profil
  🚀 Starte Control Plane Node minikube in Cluster minikube
  🚗 Ziehe das Base Image ...
  ⚙️ Starte existierenden docker container für "minikube" ...

  🚧 Docker is nearly out of disk space, which may cause deployments to fail
  (% of capacity). You can pass '--force' to skip this check.
 💡 Vorschlag:
```

Minikube cluster is setup and kubectl is also connected to the kube cluster

```
(base) gu@gu-GE60-2PC:~$ kubectl get nodes
NAME      STATUS   ROLES      AGE   VERSION
minikube  Ready    control-plane   16h   v1.24.1
(base) gu@gu-GE60-2PC:~$ minikube status
minikube: Befehl nicht gefunden.
(base) gu@gu-GE60-2PC:~$ minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured
```

```
(base) gu@gu-GE60-2PC:~$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short. Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.2", GitCommit:"f60044f4361b9f1f96f0053dd46cb7dce5e990a8", GitTreeState:"clean", BuildDate:"2022-06-15T14:22:29Z", GoVersion:"go1.18.3", Compiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v4.5.4
Server Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.1", GitCommit:"3dd0f45aa91e2f30c70734b175631bec5b5825a", GitTreeState:"clean", BuildDate:"2022-05-24T12:18:48Z", GoVersion:"go1.18.2", Compiler:"gc", Platform:"linux/amd64"}
```

```
kubectl version  
minikube status
```

```
kubectl get nodes
```

Basic kubectl commands:



```
kubectl get {k8s-component}
```

```
kubectl get nodes
```

```
kubectl get pods
```

```
kubectl get services
```

```
kubectl get deployment
```

kubectl get all

```
kubectl create {k8s-component} {name} {options}
```

```
kubectl create deployment my-nginx-depl --image=nginx
```

Get status of different components

CRUD

```
kubectl edit {k8s-component} {name}
```

```
kubectl delete {k8s-component} {name}
```

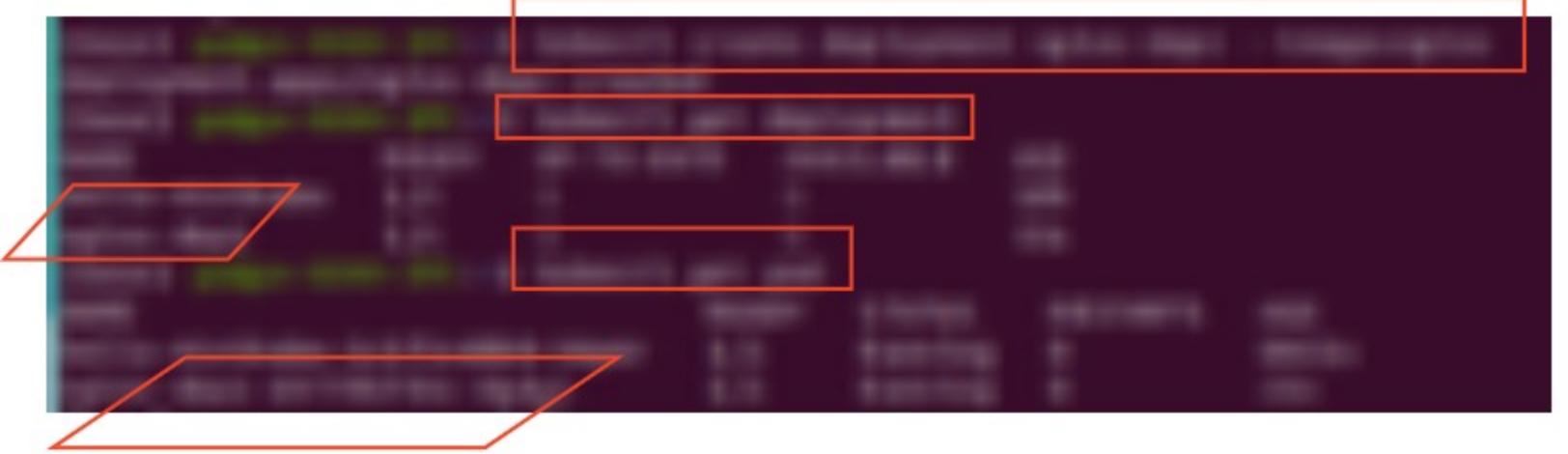
kubectl logs {pod-name}
just name, no component

```
kubectl describe {pod-name}
```

Debugging

```
kubectl exec -it {pod-name} -- bash
```

```
kubectl apply -f config-file.yaml
```



Replicaset is managing the replicas of a pod

```
(base) gu@gu-GE60-2PC:~$ kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
hello-minikube 1/1     1            1           16h
nginx-dep1      1/1     1            1           15s
(base) gu@gu-GE60-2PC:~$ kubectl get pod
NAME                               READY   STATUS    RESTARTS   AGE
hello-minikube-5c5f5cddb9-h6p6r   1/1     Running   0          8m51s
nginx-dep1-64779b795c-8g4vl       1/1     Running   0          25s
(base) gu@gu-GE60-2PC:~$ kubectl get replicaset
NAME           DESIRED   CURRENT   READY   AGE
hello-minikube-5c5f5cddb9         1         1         1       16h
nginx-dep1-64779b795c             1         1         1       3m42s
(base) gu@gu-GE60-2PC:~$
```



```
[base] gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
```

```
(base) gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
error: deployments.apps "nginx-dep1" is invalid
deployment.apps/nginx-dep1 edited
(base) gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
Edit cancelled, no changes made.
```

Auto-generated configuration file with default values -> change the version of nginx image-> the old pod is terminated and a new pod is created

```
(base) gu@gu-GE60-2PC:~$ kubectl get pod
NAME                               READY   STATUS    RESTARTS   AGE
hello-minikube-5c5f5cddb9-h6p6r   1/1     Running   0          26m
nginx-dep1-64779b795c-4qvp2       1/1     Running   0          11s
(base) gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
Error from server (NotFound): deployments.apps "nginx-dep1" not found
(base) gu@gu-GE60-2PC:~$ ^C
(base) gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
error: deployments.apps "nginx-dep1" is invalid
deployment.apps/nginx-dep1 edited
(base) gu@gu-GE60-2PC:~$ kubectl edit deployment nginx-dep1
Edit cancelled, no changes made.
(base) gu@gu-GE60-2PC:~$ kubectl get pod
NAME                               READY   STATUS    RESTARTS   AGE
hello-minikube-5c5f5cddb9-h6p6r   1/1     Running   0          30m
nginx-dep1-5f4fbfdff-m2kqs        1/1     Running   0          2m28s
(base) gu@gu-GE60-2PC:~$
```

```
(base) gu@gu-GE60-2PC:~$ kubectl get replicaset
NAME           DESIRED   CURRENT   READY   AGE
hello-minikube-5c5f5cddb9   1         1         1       16h
nginx-dep1-5f4fbfdff       1         1         1       4m19s
nginx-dep1-64779b795c       0         0         0       6m24s
(base) gu@gu-GE60-2PC:~$
```

Get the logs in a pd. (here nothing done)

```
error from server (NotFound): pods "nginx-dep1" not found
(base) gu@gu-GE60-2PC:~$ kubectl logs nginx-dep1-5f4fbfdff-m2kqs
(base) gu@gu-GE60-2PC:~$
```

```
Error from server (NotFound): pods "mongo-dep1" not found
(base) gu@gu-GE60-2PC:~$ kubectl logs mongo-dep1-6f76f4469-flnmx
{"t":{"$date":"2022-07-13T07:54:39.762+00:00"}, "s": "I", "c": "CONTROL", "i": 285, "ctx": "-", "msg": "Automatically disabling TLS 1.0, to force-enable TLS specify --sslDisabledProtocols 'none'"}
 {"t":{"$date":"2022-07-13T07:54:39.765+00:00"}, "s": "I", "c": "NETWORK", "i": 15701, "ctx": "main", "msg": "Initialized wire specification", "attr": {"spec": {"minExternalClient": {"minWireVersion": 0, "maxWireVersion": 13}, "incomingInternal": {"minWireVersion": 0, "maxWireVersion": 13}, "outgoing": {"minWireVersion": 0, "maxWireVersion": 13}, "isInternalClient": true}}}
 {"t": "2022-07-13T07:54:39.765+00:00", "s": "I", "c": "NETWORK", "i": 15701, "ctx": "main", "msg": "TLS 1.0 will not be forced enabled", "attr": {"spec": {"minExternalClient": {"minWireVersion": 0, "maxWireVersion": 13}, "incomingInternal": {"minWireVersion": 0, "maxWireVersion": 13}, "outgoing": {"minWireVersion": 0, "maxWireVersion": 13}, "isInternalClient": true}}}
```

```
(base) gu@gu-GE60-2PC:~$ kubectl describe pod mongo-dep1-6f76f4469-flnmx
Name:           mongo-dep1-6f76f4469-flnmx
Namespace:      default
Priority:       0
Node:           minikube/192.168.49.2
Start Time:     Wed, 13 Jul 2022 09:54:23 +0200
Labels:         app=mongo-dep1
                pod-template-hash=6f76f4469
Annotations:    <none>
Status:         Running
IP:             172.17.0.3
IPs:
  IP:          172.17.0.3
Controlled By: ReplicaSet/mongo-dep1-6f76f4469
Containers:
  mongo:
    Container ID:  docker://f842989c6d2e210db9b4b423b0639644e322c859e0249b21aa
```

Interactive command line: -- bin/bash

```
(base) gu@gu-GE60-2PC:~$ kubectl exec -it mongo-dep1-6f76f4469-flnmx -- bin/bash
root@mongo-dep1-6f76f4469-flnmx:/# pws
bash: pws: command not found
root@mongo-dep1-6f76f4469-flnmx:/# pwd
/
root@mongo-dep1-6f76f4469-flnmx:/# ls
bin  dev          home      lib32   media   proc   sbin   tmp
boot docker-entrypoint-initdb.d js-yaml.js lib64   mnt     root   srv   usr
data etc          lib        libx32   opt     run     sys   var
root@mongo-dep1-6f76f4469-flnmx:/# █
```

Write too much in the command line -> not good -> configuration file -> using kubectl apply command (takes a file)

```
kubectl version  
minikube status
```

```
kubectl get nodes
```

Basic kubectl commands:

```
kubectl get {k8s-component}
```

```
kubectl get nodes
```

```
kubectl get pods
```

```
kubectl get services
```

```
kubectl get deployment
```

kubectl get all

```
kubectl create {k8s-component} {name} {options}
```

```
kubectl create deployment my-nginx-depl --image=nginx
```

```
kubectl edit {k8s-component} {name}
```

CRUD

```
kubectl delete {k8s-component} {name}
```

```
kubectl logs {pod-name}
```

Debugging

```
kubectl describe {pod-name}
```

```
kubectl exec -it {pod-name} -- bash
```

```
kubectl apply -f config-file.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
```

deployment

pod

With kubectl apply you can create or update a deployment
update config-file -> then kubectl apply again with same name ->
kubectl get deployment not change , but there are two pods, old one
and new one.

status in a yaml format

use get deployment and output as a yaml file

```
$ kubectl get deployment nginx-deployment -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"
        labels":{"app":"nginx"},"name":"nginx-deployment","namespace":"
      de
      {"replicas":2,"selector":{"matchLabels":{"app":"nginx"}}, "templat
      {"labels":{"app":"nginx"}}, "spec": {"containers": [{"image": "nginx:
      ginx", "ports": [{"containerPort": 8080}]}]}]}}
    creationTimestamp: "2022-07-13T09:44:21Z"
    generation: 1
    labels:
```

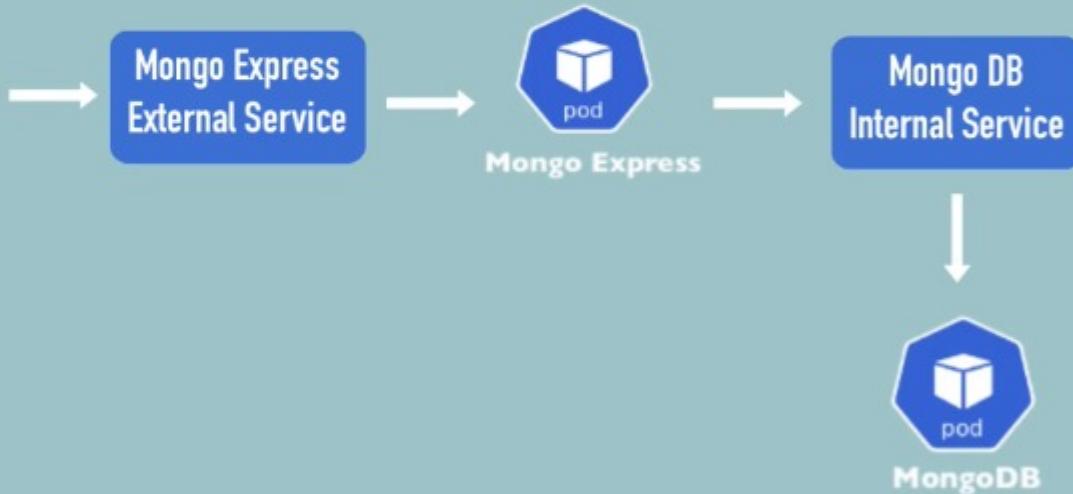
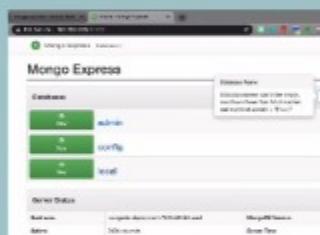
If you want to deploy another deployment, the generated yaml must be cleaned, then use as blueprint

Demo project

mongoDB : internal service (no external request)

mongo Express (database url, connect to mongodb, authenticate) -> deployment,yaml (configMap(db url), secret(DB User, DB pwd))
mongoDB excess from external: External serivce

Browser Request Flow through the K8s components



1. create a mongo DB deployment
2. create secret for mongo user and mongo password
 - a. how to create user and password text
3. ACTUNG: first deploy secret then mongodb so that you can reference it.

```
mongo-secret.yaml mongo.yaml  
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcam  
p/demo project$ kubectl apply -f mongo-secret.yaml  
secret/mongodb-secret created  
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcam  
p/demo project$ kubectl get secret  
NAME          TYPE        DATA   AGE  
mongodb-secret  Opaque      2      9s  
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcam  
p/demo project$
```

Then we can deploy mongo

```
p/demo project$ kubectl get pod --watch  
NAME                           READY   STATUS        RESTARTS   AGE  
mongodb-deployment-778f488b57-cg7br   0/1     ImagePullBackOff   0          77s
```

```
3m46s   172.17.0.3   minikube   <none>           <none>  
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcam  
p/demo project$ kubectl describe pod mongodb-deployment-658bf778dc-bx9nf  
Name:           mongodb-deployment-658bf778dc-bx9nf  
Namespace:      default  
Priority:       0  
Node:          minikube/192.168.49.2  
Start Time:    Wed, 13 Jul 2022 14:10:04 +0200  
Labels:         app=mongodb  
               pod-template-hash=658bf778dc  
Annotations:    <none>  
status:         Pending
```

Create a tunnel service -> communicate with
mongodb

in yaml we can put different files together. --- document separation
We need to create service

```

mongo.yaml u ×
...
emo project > ! mongo.yaml > {} spec > {} template > {} spec > [ ]containers > {} 0 > [ ]ports > {} 0
    all.json
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4      name: mongodb-deployment
5      labels:
6          app: mongodb
7  spec:
8      replicas: 1
9      selector:
10         matchLabels:
11             app: mongodb
12     template:
13         metadata:
14             labels:
15                 app: mongodb
16     spec:
17         containers:
18             - name: mongodb
19                 image: mongodb
20                 ports:
21                     - containerPort: 27017
22             env:
23                 - name: MONGO_INITDB_ROOT_USERNAME
24                     valueFrom:
25                         secretKeyRef:
26                             name: mongodb-secret
27                             key: mongo-root-username
28                 - name: MONGO_INITDB_ROOT_PASSWORD
29                     valueFrom:
30                         secretKeyRef:
31                             name: mongodb-secret
32                             key: mongo-root-password
33

```



```

! mongo-secret.yaml u ×
...
demo project > ! mongo-secret.yaml > {} data > mongo-root-password
    all.json
1  apiVersion: v1
2  kind: Secret
3  metadata:
4      name: mongodb-secret
5  type: Opaque
6  data:
7      mongo-root-username: dXNlcm5hbWU=
8      mongo-root-password: cGFzc3dvcnQ=

```

```

(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/demo project$ kubectl apply -f mongo.yaml
deployment.apps/mongodb-deployment unchanged
service/mongodb-service created
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/demo project$ 

```

write the service deployment together with mongodb deployment seperated with ---

Then kubectl apply the file again.

```
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/demo project$ kubectl describe service mongodb-service
Name:                   mongodb-service
Namespace:              default
Labels:                 <none>
Annotations:            <none>
Selector:               app=mongodb
Type:                  ClusterIP
IP Family Policy:      SingleStack
IP Families:           IPv4
IP:                    10.97.64.177
IPs:                  10.97.64.177
Port:                  <unset>  27017/TCP
TargetPort:             27017/TCP
Endpoints:              172.17.0.3:27017
Session Affinity:       None
Events:                <none>
```

Next step is to create: Mongo express, Mongo express external service
ConfigMap DB url

The screenshot shows the configuration interface for the mongo-express Docker image on Docker Hub. On the left, a table lists environment variables with their default values:

Name	Default
ME_CONFIG_BASICAUTH_USERNAME	''
ME_CONFIG_BASICAUTH_PASSWORD	''
ME_CONFIG_MONGODB_ENABLE_ADMIN	'true'
ME_CONFIG_MONGODB_ADMINUSERNAME	''
ME_CONFIG_MONGODB_ADMINPASSWORD	''
ME_CONFIG_MONGODB_PORT	27017
ME_CONFIG_MONGODB_SERVER	'mongo'
ME_CONFIG_OPTIONS_EDITORTHEME	'default'
ME_CONFIG_REQUEST_SIZE	'100kb'
ME_CONFIG_SITE_BASEURL	'/'
ME_CONFIG_SITE_COOKIESECRET	'cookiesecret'
ME_CONFIG_SITE_SESSIONSECRET	'sessionsecret'
ME_CONFIG_SITE_SSL_ENABLED	'false'
ME_CONFIG_SITE_SSL_CRT_PATH	''
ME_CONFIG_SITE_SSL_KEY_PATH	''

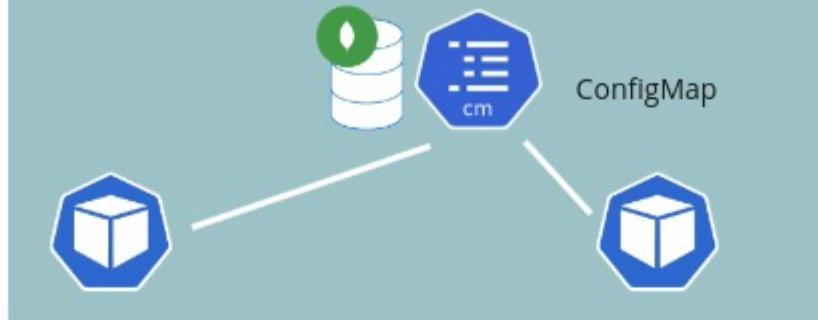
On the right, two modal windows are displayed:

- Which database to connect?** (Modal 1)
- MongoDB Address / Internal Service** (Modal 2)
- Which credentials to authenticate?** (Modal 3)

Below the configuration table, a note states: "The following are only needed if ME_CONFIG_MONGODB_ENABLE_ADMIN is set to true".

ConfigMap

- external configuration
- centralized
- other components can use it



info of database store in the ConfigMap. Other application can also use it

20:05

ConfigMap must already be in the k8s cluster when referencing it!

ConfigMap -> mongo-express (while mongo-express needs the ME_CONFIG_MONGODB_SERVER , which data is database_url

- deploy ConfigMap
- deploy mongo-express
- check if it works with kubectl logs + podID
- next step is to access mongoDB in the browser => Tunnel service

How to make it an External Service?

- type: "Loadbalancer"

..assigns service an external IP address and so accepts external requests

- nodePort: must be between 30000-32767

Port for external IP address

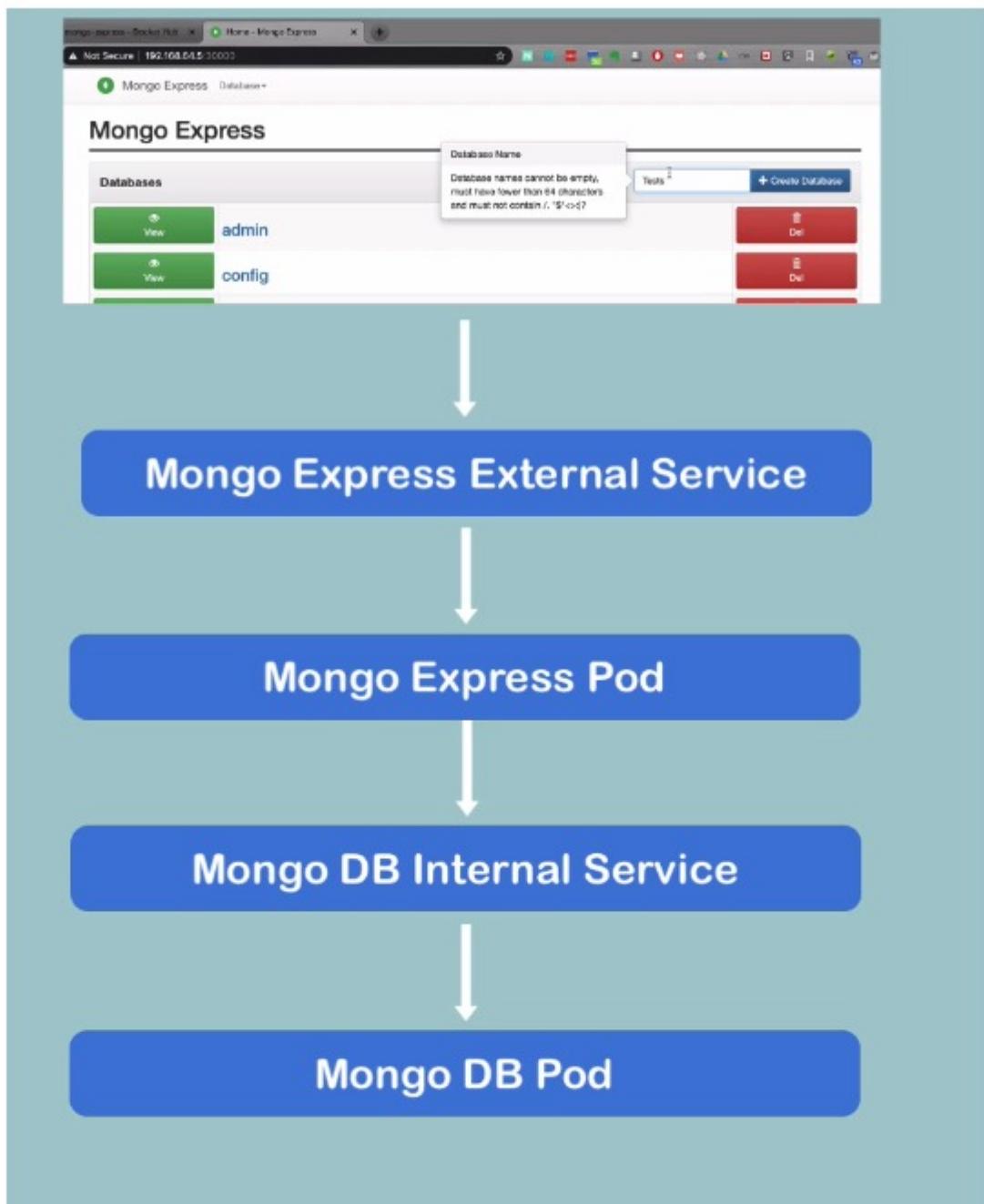
Port you need to put into browser

(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/demo project\$ kubectl apply -f mongo-express.yaml				
deployment.apps/mongo-express-deployment unchanged				
service/mongo-express-service created				
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/demo project\$ kubectl get service				
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
mongo-express-service	LoadBalancer	10.109.83.143	<pending>	8081:30000/
TCP	11s			
mongodb-service	ClusterIP	10.97.64.177	<none>	27017/TCP
17h				

Internal service or cluster IP is default

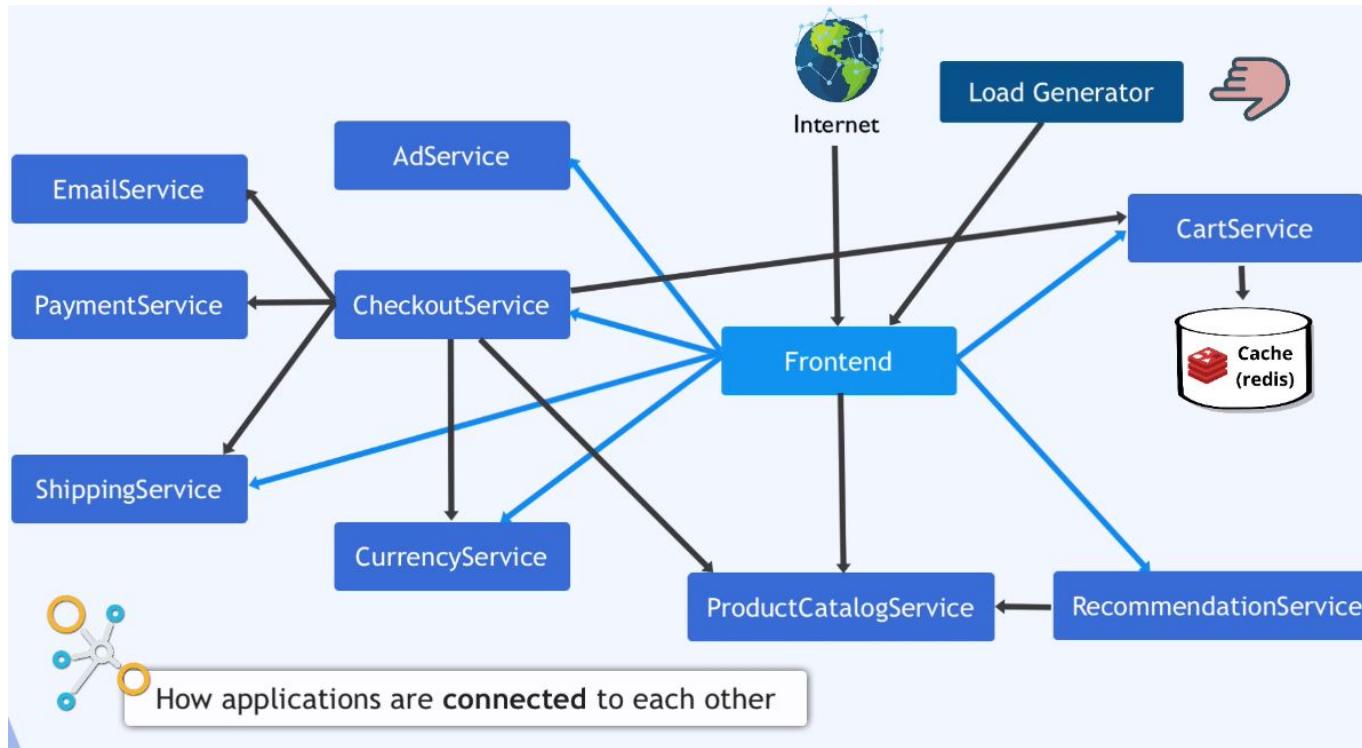
minikube for loadbanlance pending is external IP
do: minikube service mongo-express-service
You get external service

Progresses of a request



-

Deploy microservice app into k8s cluster



how applications are connected to each other

image names for each microservice

environment variables

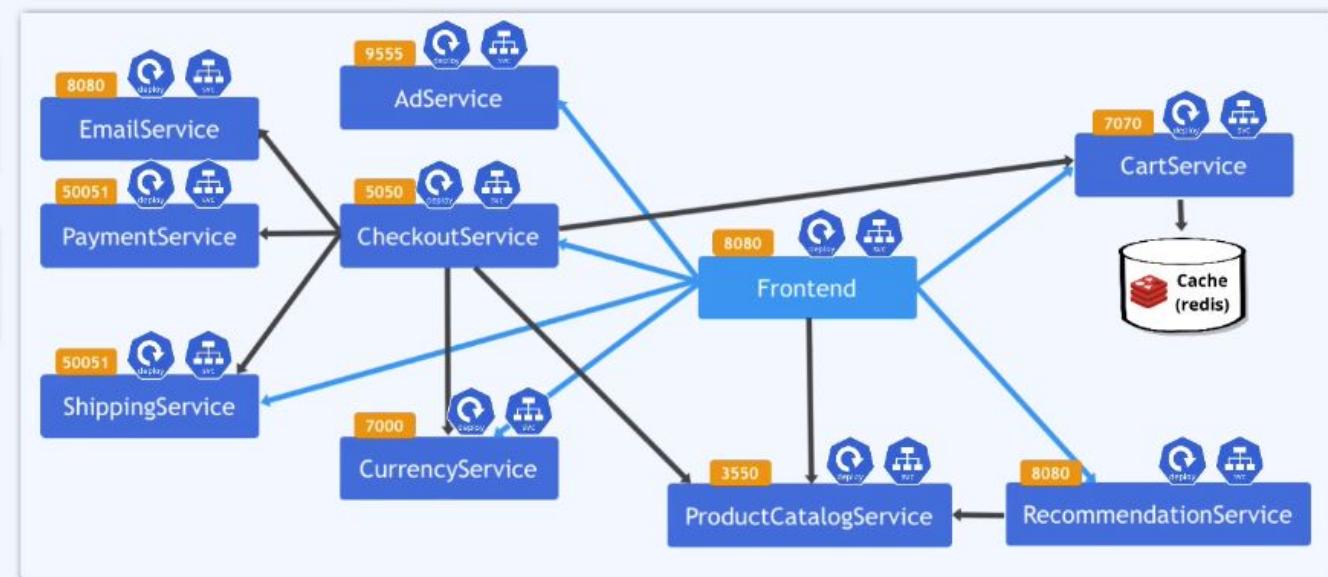
port

deploy ms into single spaces



Create Deployment Config
for each Microservice

Create Service Config
for each Microservice



1. create a config file
 - a. skeleton
2. speed up k8s cluster in linode
 - a. download the kubeconfig file
3. set permission of kubeconfig file

```
[\W]$ chmod 400 ~/Downloads/online-shop-kubeconfig.yaml  
[\W]$ ls -l ~/Downloads/online-shop-kubeconfig.yaml  
-r-----@ 1 nanajanashia  staff  2764 Jun 16 09:54 /Users  
config.yaml
```

4. variable

```
[\W]$ export KUBECONFIG=~/Downloads/online-shop-microservices-kubeconfig.yaml
```

5. kubectl get node check if the cluster right configured
6. kubectl create ns microservices
7. kubectl apply -f config.yaml -n microservices
8. kubectl get pod -n microservices
9. ubectl get svc -n microservices

 Linodes Volumes NodeBalancers Firewalls StackScripts Images Domains Kubernetes Object Storage Longview Marketplace Account Help & Support

A new version of Kubernetes is available (1.21).

Kubernetes / **online-shop-microservices**

Version 1.20



3 CPU Cores



Frankfurt, DE



6 GB RAM



\$30/month



150 GB Storage

Kubernetes API Endpoint:

<https://7dc7925e-a17e-4b3f-8fb9-bb3724f8ac43.eu-central-2.linode.net:443>

Kubeconfig:

[online-shop-microservices-kubeconfig.yaml](#)

Node Pools

Linode 2 GB

Linode ^	Status ^	IP Address ^
lke28633-42767-60c34bde69f3	Running	192.195.240.188
lke28633-42767-60c34bdecc8a	Running	194.233.164.143
lke28633-42767-60c34bdf2dc8	Running	194.195.241.114



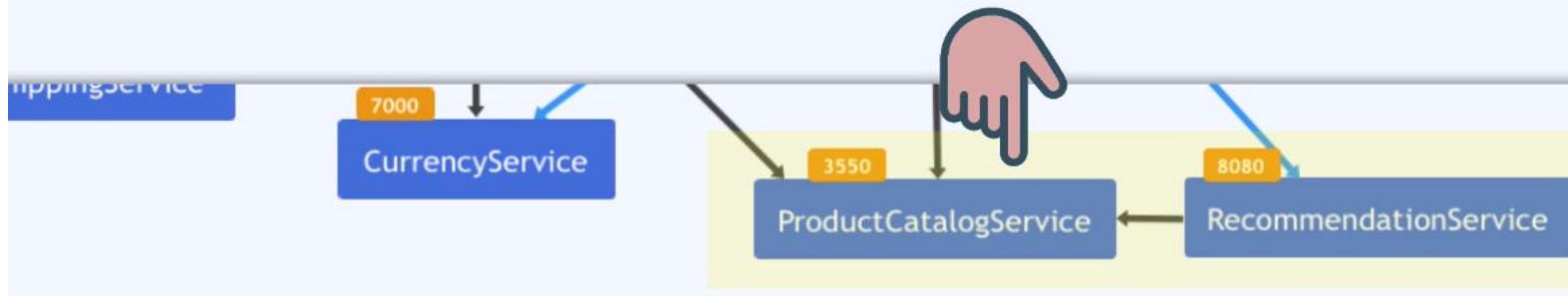
Not Secure | 194.195.240.188:30007

Free shipping with \$75 purchase!



Recommendation Service needs to know the endpoint of ProductCatalogService

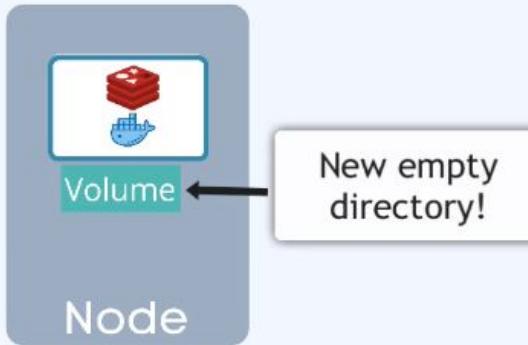
Endpoint = Service Name + Service Port





emptyDir

- ▶ Is initially empty
- ▶ First created when a Pod is assigned to a Node
- ▶ Exists as long as the Pod is running





emptyDir

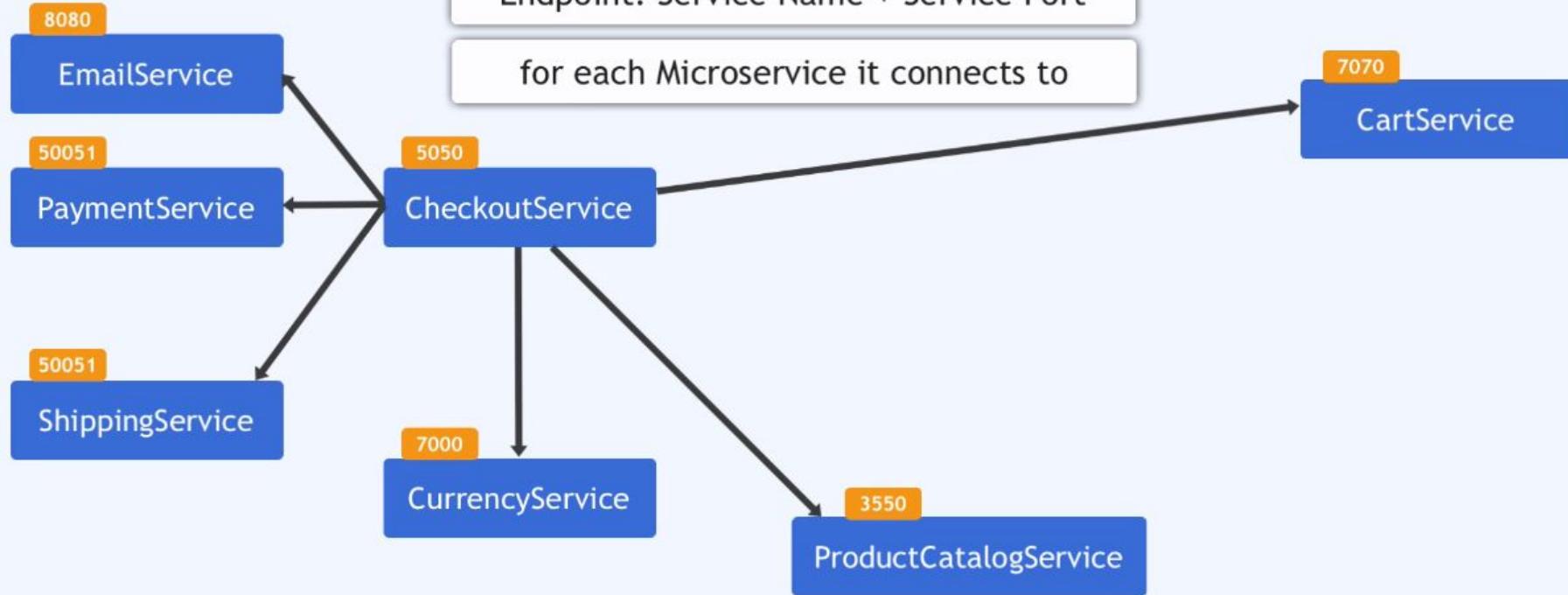
- ▶ Is initially empty
- ▶ First created when a Pod is assigned to a Node
- ▶ **Exists as long as the Pod is running**
- ▶ Container crashing does NOT remove a Pod from a Node
- ▶ Therefore, **data safe across container crashes!**



CheckoutService needs to know:

Endpoint: Service Name + Service Port

for each Microservice it connects to

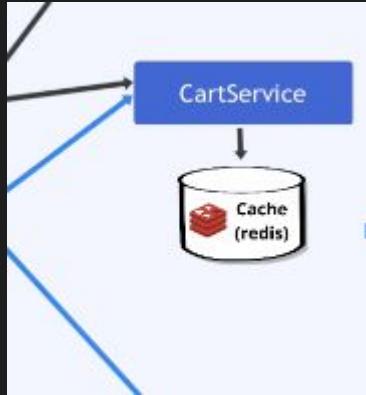


```
88             port = os.Getenv("PORT")
89         }
90
91         svc := new(checkoutService)
92         mustMapEnv(&svc.shippingSvcAddr, "SHIPPING_SERVICE_ADDR")
93         mustMapEnv(&svc.productCatalogSvcAddr, "PRODUCT_CATALOG_SERVICE_ADDR")
94         mustMapEnv(&svc.cartSvcAddr, "CART_SERVICE_ADDR")
95         mustMapEnv(&svc.currencySvcAddr, "CURRENCY_SERVICE_ADDR")
96         mustMapEnv(&svc.emailSvcAddr, "EMAIL_SERVICE_ADDR")
97         mustMapEnv(&svc.paymentSvcAddr, "PAYMENT_SERVICE_ADDR")
98
99
100        image: gcr.io/google-samples/microservices-demo/checkoutservice
101
102        ports:
103            - containerPort: 5050
104
105        env:
106            - name: PORT
107                value: "5050"
108            - name: PRODUCT_CATALOG_SERVICE_ADDR
109
110        ---
111
112        apiVersion: v1
113        kind: Service
114
115        metadata:
116            name: checkoutservice
117
118        spec:
119            type: ClusterIP
```

```
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: emailservice  
spec:  
  selector:  
    matchLabels:  
      app: emailservice  
  template:  
    metadata:  
      labels:  
        app: emailservice  
    spec:  
      containers:  
        - name: service  
          image: gcr.io/google-samples/emailservice  
          ports:  
            - containerPort: 8080  
          env:  
            - name: PORT  
              value: "8080"
```

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: emailservice  
spec:  
  type: ClusterIP  
  selector:  
    app: emailservice  
  ports:  
    - protocol: TCP  
      port: 5000  
      targetPort: 8080
```

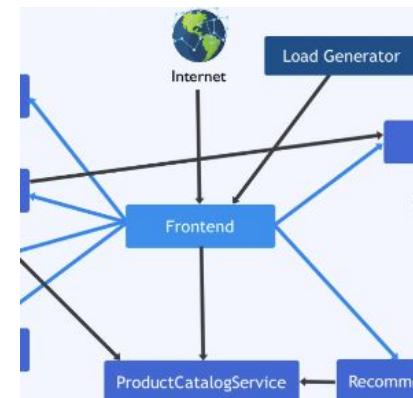
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cartservice
spec:
  selector:
    matchLabels:
      app: cartservice
  template:
    metadata:
      labels:
        app: cartservice
  spec:
    containers:
      - name: service
        image: gcr.io/google-samples/cartservice
        ports:
          - containerPort: 7070
        env:
          - name: PORT
            value: "7070"
          - name: REDIS_ADDR
            value: "redis-cart:6379"
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-cart
spec:
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
  spec:
    containers:
      - name: redis
        image: redis:alpine
        ports:
          - containerPort: 6379
        volumeMounts:
          - mountPath: /data
            name: redis-data
    volumes:
      - name: redis-data
        emptyDir: {}
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: service
          image: gcr.io/google-samples/frontend
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: "8080"
            - name: SHIPPING_SERVICE_ADDR
              value: "shippingservice:50051"
            - name: PRODUCT_CATALOG_SERVICE_ADDR
              value: "productcatalogservice:3550"
            - name: CART_SERVICE_ADDR
```

```
...
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30007
...
```



**best
practices**

1. define image version
2. liveness
3. readiness

when the container is not running
kubectl logs checkout
disable functions

```
config.yaml x ! tmp.yaml
! config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] env
66     - name: service
67       image: gcr.io/google-samples/microservices-demo/recommendati
68       ports:
69         - containerPort: 8080
70       env:
71         - name: PORT
72           value: "8080"
73         - name: PRODUCT_CATALOG_SERVICE_ADDR
74           value: "productcatalogservice:3550"
75         - name: DISABLE_TRACING
76           value: "1"
77         - name: DISABLE_PROFILER
78           value: "1"
79         - name: DISABLE_DEBUGGER
80           value: "1"
81       readinessProbe:
```

```
[\w]$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
emailservice-5787b7575-jhtrk	1/1	Running	0	36s
productcatalogservice-54d7fc98bb-nj7f2	1/1	Running	0	36s
recommendationservice-68956c9949-2z5md	1/1	Running	0	36s

```
[\w]$
```

```
! config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} readinessProbe > {} exec > [ ] command
```

```
23           value: "8080"
```

2) TCP probes

```
26       - name: DISABLE_P
27           value: "1"
28       readinessProbe:
29           periodSeconds: 5
30           exec:
31               command: ["/bin/grpc_health_probe", "-addr=:8080"]
32       livenessProbe:
33           periodSeconds: 5
34           exec:
35               command: ["/bin/grpc_health_probe", "-addr=:8080"]
36 ---
```

Kubectl makes probe connection at
the node, not in the pod

? config.yaml ×

```
! config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livenessProbe > {} tcpSocket > +  
360     template:  
361         metadata:  
362             labels:  
363                 app: redis-cart  
364             spec:  
365                 containers:  
366                     - name: redis  
367                         image: redis:alpine  
368                         ports:  
369                             - containerPort: 6379  
370                         readinessProbe:  
371                             periodSeconds: 5  
372                             tcpSocket:  
373                                 port: 6379  
374                         livenessProbe:  
375                             periodSeconds: 5  
376                             tcpSocket:  
377                                 port: 6379|  
378                         volumeMounts:  
379                             - name: redis-data
```

methods:
command
tcpsocket
http

```
spec:  
  containers:  
    - name: redis  
      image: redis:alpine  
      ports:  
        - containerPort: 6379  
      readinessProbe:  
        initialDelaySeconds: 5  
        periodSeconds: 5  
        tcpSocket:  
          port: 6379  
      livenessProbe:  
        initialDelaySeconds: 5  
        periodSeconds: 5  
        tcpSocket:
```

wait 5 seconds before performing probe

config.yaml — temp

```
... ! config.yaml ×  
! config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livenessProbe > {} tcpSocket > ## port  
118     containers:  
119         - name: service
```

3) HTTP probes

```
122             - containerPort: 3550  
123             env:  
124                 - name: PORT  
125                     value: "3550"  
126             livenessProbe:  
127                 periodSeconds: 5  
128             exec:  
129                 command: ["/bin/grpc_health_probe", "-addr=:3550"]  
130             readinessProbe:  
131                 periodSeconds: 5
```

Kubelect sends an HTTP request to specified path and port

```
yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livenessProbe
    app: productcatalogservice
spec:
  containers:
  - name: service
    image: gcr.io/google-samples/microservices
    ports:
    - containerPort: 3550
    env:
    - name: PORT
      value: "3550"
    livenessProbe:
      periodSeconds: 5
      httpGet:
        path: /health
        port: 3550
    readinessProbe:
      periodSeconds: 5
      exec:
        command: ["/bin/grpc_health_probe", "
```

```
define resource
```

```
  - name: PORT
    value: "8080"
  livenessProbe:
    periodSeconds: 5
  exec:
    command: ["/bin/grpc_health_probe", "-ad"]
  readinessProbe:
    periodSeconds: 5
  exec:
    command: ["/bin/grpc_health_probe", "-ad"]
  resources:
    requests:
      cpu: 100m
      memory: 64Mi
```

```
    command: ["/bin/grpc_heal
resources:
  requests:
    cpu: 100m
    memory: 64Mi
  limits:
    cpu: 200m
    memory: 128Mi
```



CPU resources are defined in millicores

Memory resources are defined in bytes

m = "millicore"

Mi = "mebibyte"

Version: v1

d: Service

adata:

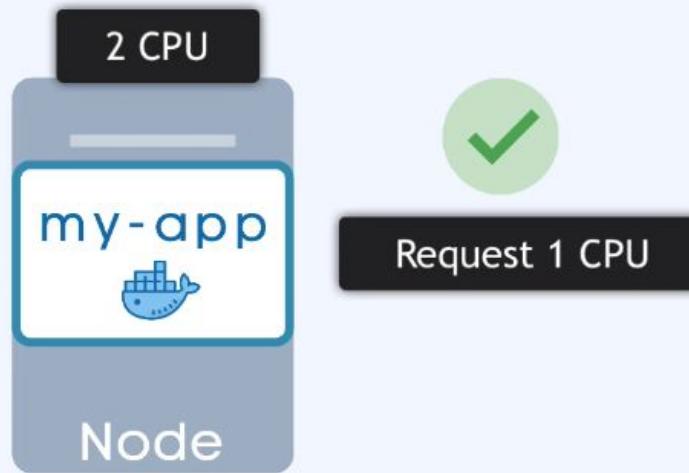
ame: emailservic

Why Resource Limits are important?

What if application needs
more resources?



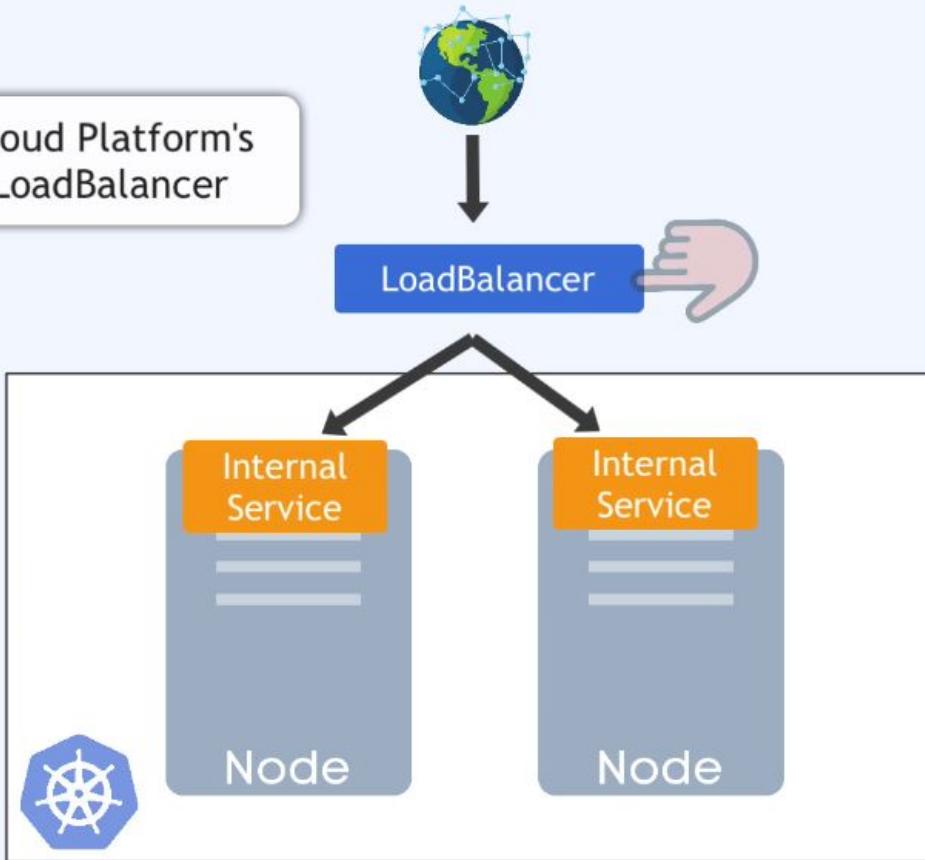
Needing more..



- ▶ Container will consume more than the requested resources
- ▶ If not limited, container could **consume all the Node's resources**

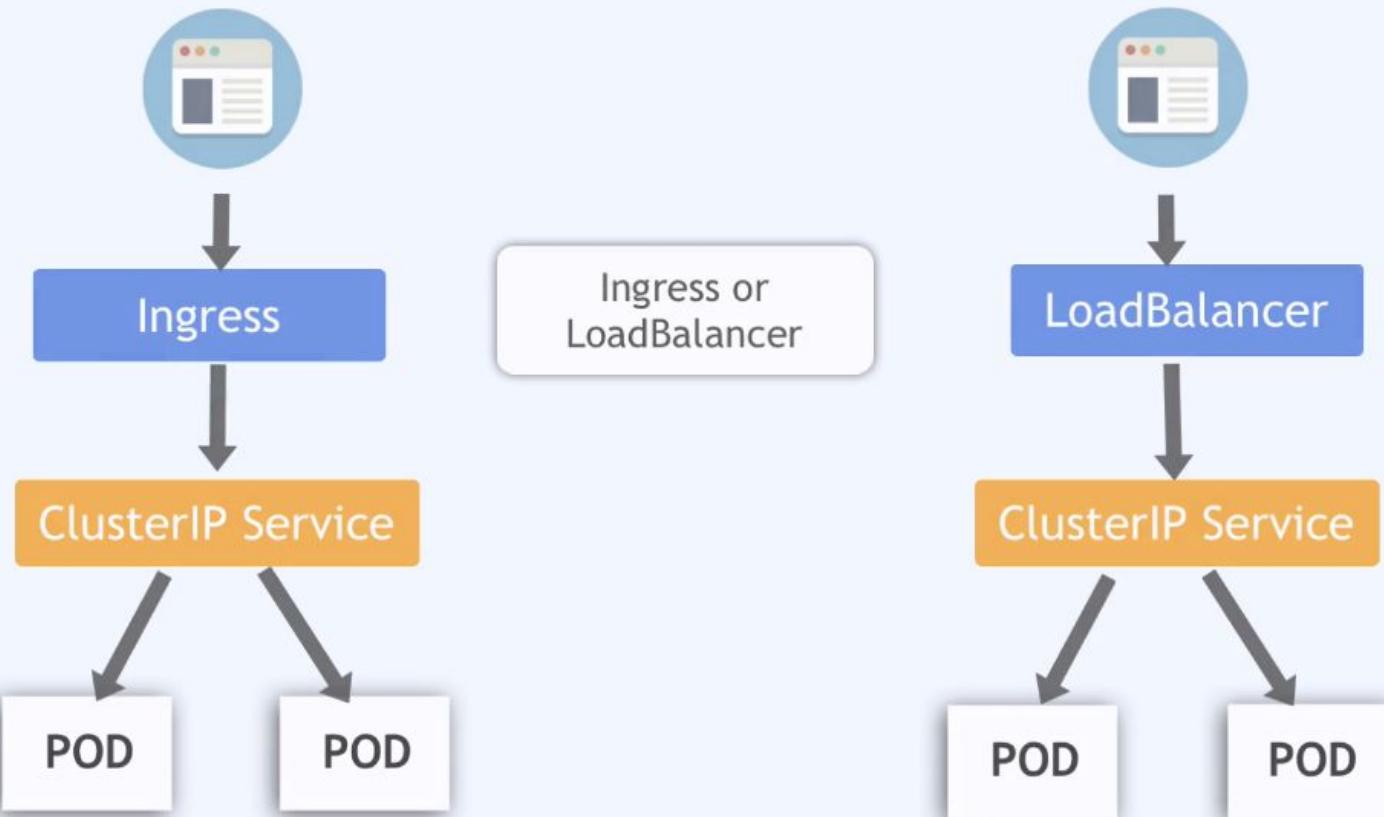
Why it's a Bad Practice?

Cloud Platform's
LoadBalancer



```
apiVersion: v1
kind: Service
metadata:
  name: frontend-external
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
    - name: http
      port: 80
      targetPort: 8080
```

Ingress as alternative



Why it's important?

- ▶ Custom Identifier for your components

1. Group Pods with Labels

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: emailservice
spec:
  selector:
    matchLabels:
      app: emailservice
  template:
    metadata:
      labels:
        app: emailservice
>   spec: ...
```

2. Reference in Service Component

```
apiVersion: v1
kind: Service
metadata:
  name: emailservice
spec:
  type: ClusterIP
  selector:
    app: emailservice
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 8080
```

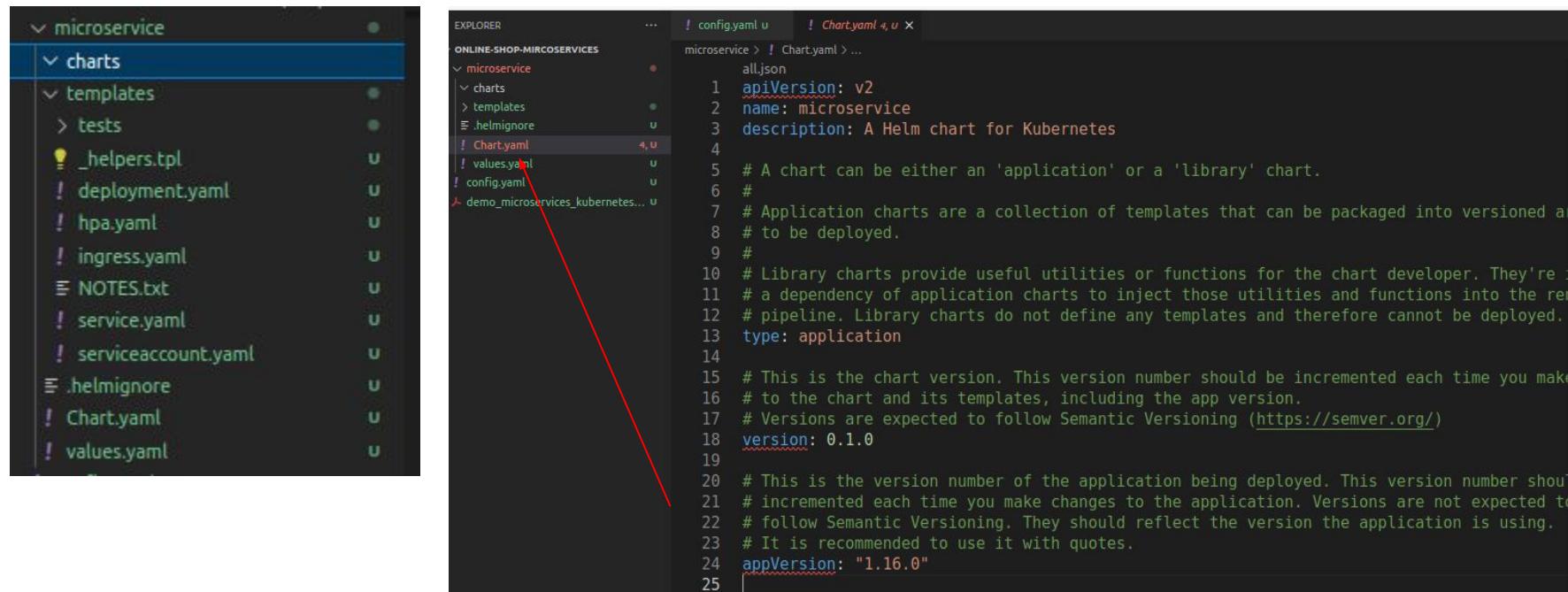

helm chart: reuseable k8s configuration

two ways creating helm chart

- helm chart for each microservice, when configuration are very different
- 1 shared helm chart for all microservice
- combine of the both options

```
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/online-shop-mircoservices$ helm create microservice
Creating microservice
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/online-shop-mircoservices$
```

1. create a helm chart with name micrioservice



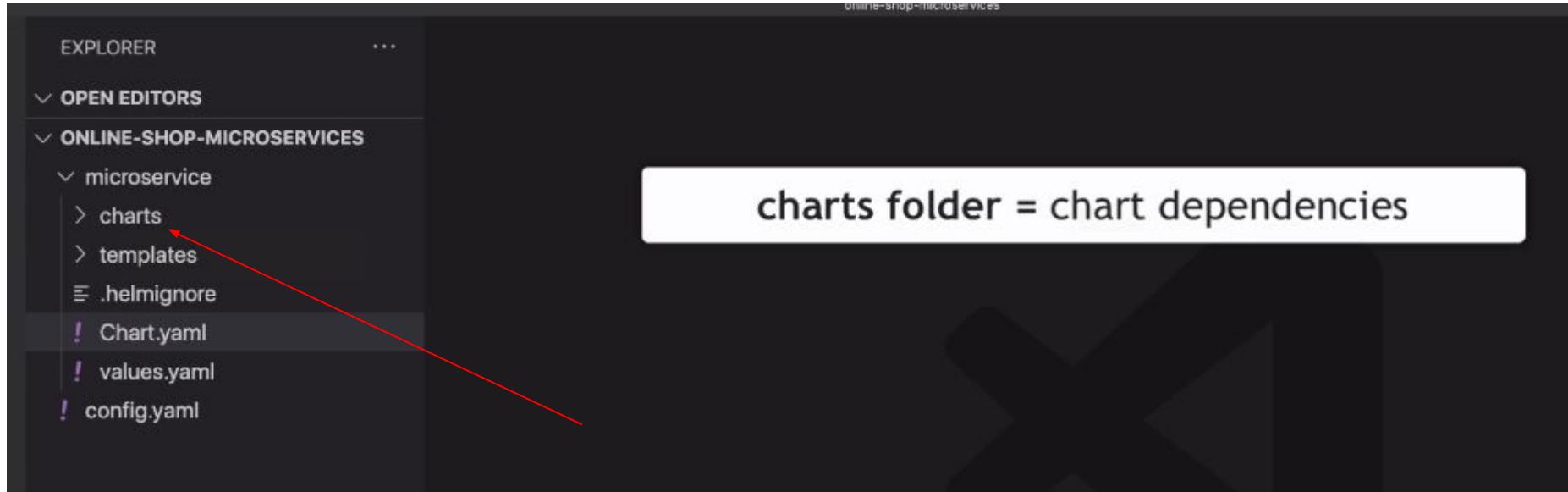
The screenshot shows a terminal window with the following command history:

```
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/online-shop-mircoservices$ helm create microservice
Creating microservice
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/online-shop-mircoservices$
```

The current working directory is `online-shop-mircoservices`. A new Helm chart named `microservice` has been created. The chart structure includes a `charts` folder containing a `microservice` sub-chart, which contains a `Chart.yaml` file.

The `Chart.yaml` file content is as follows:

```
apiVersion: v2
name: microservice
description: A Helm chart for Kubernetes
# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into versioned artifacts to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer. They're typically a dependency of application charts to inject those utilities and functions into the release pipeline. Library charts do not define any templates and therefore cannot be deployed.
type: application
# This is the chart version. This version number should be incremented each time you make changes to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.1.0
# This is the version number of the application being deployed. This version number should be incremented each time you make changes to the application. Versions are not expected to follow Semantic Versioning. They should reflect the version the application is using.
# It is recommended to use it with quotes.
appVersion: "1.16.0"
```



EXPLORER

...

! config.yaml u

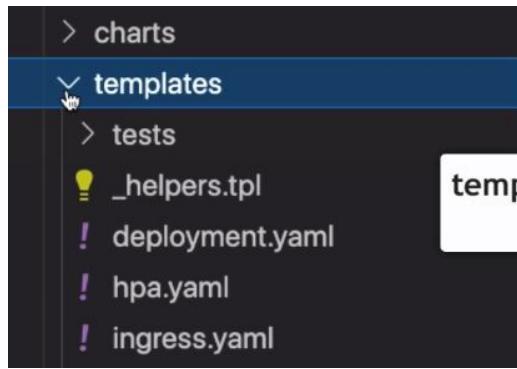
! .helmignore 7,u x

✓ ONLINE-SHOP-MIRCOSE... D+ E+ O S

- microservice
- charts
- templates
- .helmignore 7,u
- Chart.yaml u
- values.yaml u
- config.yaml u
- demo_microservices_kubernetes... u

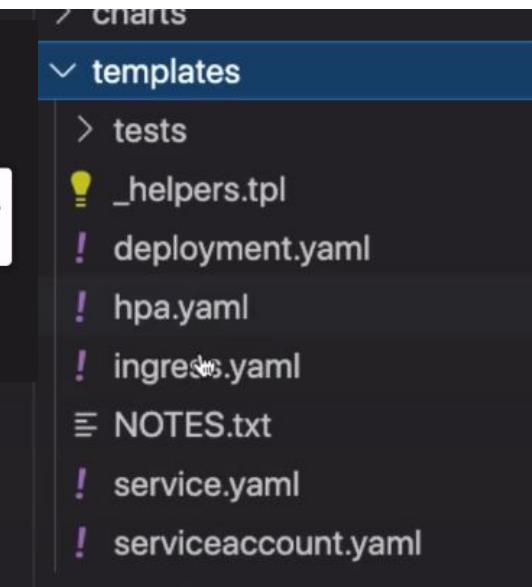
microservice > ! .helmignore

```
1 # Patterns to ignore when building packages.
2 # This supports shell glob matching, relative path
3 # negation (prefixed with !). Only one pattern per
4 .DS_Store
5 # Common VCS dirs
6 .git/
7 .gitignore
8 .bzr/
9 .bzrignore
10 .hg/
11 .hgignore
12 .svn/
13 # Common backup files
14 *.swp
15 *.bak
16 *.tmp
17 *.orig
18 *~
19 # Various IDEs
20 .project
21 .idea/
22 *.tmpproj
23 .vscode/
```



templates folder = where the actual K8s YAML files (template files) are created

create blueprint of yaml file



! service.yaml 5 ×

microservice > templates > ! service.yaml > {} metadata > {} name > [?] "undefined"

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: {{ include "microservice.fullname" . }}
5    labels:
6      {{- include "microservice.labels" . | nindent 4 }}
7  spec:
8    type: {{ .Values.service.type }}
9    ports:
10      - port: {{ .Values.service.port }}
11        targetPort: {{ .Values.service.targetPort }}
12        protocol: {{ .Values.service.protocol }}
13        name: {{ .Values.service.name }}
14    selector:
15      {{- include "microservice.selectorLabels" . | nindent 4 }}
16
```

Helm replaces these placeholders
with the actual values later

EXPLORER

...

! deployment.yaml X

! values.yaml

! service.yaml



✓ OPEN EDITORS

X ! deployment.yaml microserv...

! values.yaml microservice

! service.yaml microservice/t...

✓ ONLINE-SHOP-MICROSERVICES

✓ microservice

> charts

✓ templates

! deployment.yaml

! service.yaml

≡ .helmignore

! Chart.yaml

! values.yaml

! config.yaml

microservice > templates > ! deployment.yaml

1

values.yaml = actual values for
the template files

Create Microservices Helm Chart

Create Basic Template File

all.json

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Values.varName}}
spec:
  selector:
```

"Values" Object

- ▶ A built-in object
- ▶ By default, *Values* is empty
- ▶ Values are passed into template, from 3 sources:

- the *values.yaml* file in the chart

- user-supplied file passed with *-f* flag

- parameter passed with *--set* flag

Built-in Objects

- ▶ Several objects are passed into a template from the template engine
- ▶ Examples: "Release", "Files", "Values ", ...
- ▶ Check out:
helm.sh/docs/chart_template_guide/builtin_objects

Variable Naming Conventions

- ▶ Names should begin with a lowercase letter
- ▶ Separated with camelcase

! deployment.yaml ! values.yaml ! service.yaml

"range"

- ▶ Provides a "for each"-style loop
- ▶ To iterate through or "range over" a list.

```
env:  
{{- range .Values.containerEnvVars}}  
- name: {{.key}}  
  value: {{.value | quote}}  
{{- end}}  
  
image: "{{ .Values.appImage }}"  
ports:  
- containerPort: {{ .Value }}  
env:  
- name: {{ .Values.containerEnvVar.key }}  
  value: {{ .Values.containerEnvVar.value }}
```

EXPLORER

...

! deployment.yaml 9+, u ! values.yaml 7, u ! email-service-values.yaml 7, u >

ONLINE-SHOP-MIRCOSERVICES

microservice

> charts

templates

! deployment.yaml 9+, u

! service.yaml u

.helmignore u

! Chart.yaml u

! values.yaml 7, u

! config.yaml u

demo_microservices_kubernetes... u

! email-service-values.yaml 7, u

```
! email-service-values.yaml > □ serviceType  
all.json  
1 appName: emailservice  
2 appImage: gcr.io/google-samples/emailservice  
3 appVersion: v0.1.2  
4 appReplicas: 2  
5 containerPort: 8080  
6 containerEnvVars:  
7 - name: PORT  
8   value: "8080"  
9 - name: DISABLE_TRACING  
10  value: "1"  
11 - name: DISABLE_PROFILER  
12  value: "1"  
13  
14 servicePort: 5000  
15 serviceType: ClusterIP
```

new Go Run Terminal Window Help

email-service-values.yaml — online-shop-microservices

... config.yaml email-service-values.yaml X deployment.yaml 5

! email-service-values.yaml > [] containerEnvVars > {} 1 > value

1 appName: emailservice
2 appImage: gcr.io/google-samples/microservices-demo/emailservice
3 appVersion: v0.2.3
4 appReplicas: 2
5 containerPort: 8080
6 containerEnvVars:
7 - name: PORT
8 | value: "8080"

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

```
[\\W]$ ls  
config.yaml email-service-values.yaml microservice  
[\\W]$ helm template -f email-service-values.yaml
```

values.yaml

helm template =
render chart templates locally and display the output

```
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/4_kubernetes_Bootcamp/online-shop-mircoservices$ h  
elm template -f email-service-values.yaml microservice  
---  
# Source: microservice/templates/service.yaml  
apiVersion: v1  
kind: Service  
metadata:  
  name: emailservice  
spec:  
  type: ClusterIP  
  selector:  
    app: emailservice  
  ports:  
    - protocol: TCP  
      port: 5000  
      targetPort: 8080  
---  
# Source: microservice/templates/deployment.yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: emailservice  
spec:  
  replicas: 2  
  selector:  
    matchLabels:  
      app: emailservice  
  template:  
    metadata:  
      labels:  
        app: emailservice  
    spec:  
      containers:  
        - name: emailservice  
          image: "gcr.io/google-samples/emailservice:v0.1.2"  
          ports:  
            - containerPort: 8080  
          env:  
            - name: PORT  
              value: "8080"
```

Helm Rendering Process



- ▶ Engine replaces the variables with the actual values
(from the 3 different sources)

```
[\W]$ helm lint -f email-service-values.yaml
          value: "1"
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/
elm lint -f email-service-values.yaml microservice/
==> Linting microservice/
[INFO] Chart.yaml: icon is recommended
1 chart(s) linted, 0 chart(s) failed
(base) gu@gu-GE60-2PC:~/Documents/learn_DevOps/learn_DevOps/
```

"helm lint"

- ▶ examines a chart for possible issues

ERROR = issues that will cause chart to fail installation

WARNING = issues that break with convention or recommendations

```
... config.yaml ! email-service-values.yaml ! values.yaml ! deployment.yaml
! email-service-values.yaml > [ NAME: emailservice
  1 annName: emailservice LAST DEPLOYED: Mon Jun 14 15:01:10 2021
values.yaml PROBLEMS 8 OUTPUT Di NAMESPACE: default
microservice STATUS: deployed
ml micr... 8 REVISION: 1
SERVICES TEST SUITE: None
[ \W ]$ helm install -f ema [ \W ]$ helm ls
[ \W ]$ helm ls
```

"helm install" command

helm install

-f myvalues.yaml

emailservice

microservice

install a chart

override values
from a file

release name

chart name

create yaml file for each service, helm template... helm lint... helm install -f file
nameservice microservice

> OPEN EDITORS

✓ ONLINE-SHOP-MICROSERVICES

> microservice

✓ values

! ad-service-values.yaml

! cart-service-values.yaml

! checkout-service-values.yaml

! currency-service-values.yaml

! email-service-values copy.yaml

! payment-service-values.yaml

! productcatalog-service-values.yaml

! recommendation-service-values.yaml

! shipping-service-values.yaml

! config.yaml

```
values > ! checkout-service-values.yaml > [ ] containerEnvVars > {} 0
      1   appName: checkoutservice
      2   appImage: gcr.io/google-samples/microser
      3   appVersion: v0.2.3
      4   appReplicas: 2
      5   containerPort: 5050
      6   containerEnvVars:
      7     - name: PORT
      8       value: "5050"
      9     - name: PRODUCT_CATALOG_SERVICE_ADDR
     10       value: "productcatalogservice:3550"
     11     - name: SHIPPING_SERVICE_ADDR
     12       value: "shippingservice:50051"
```

put in the values folder for all services

create a redis chart

put microservice and chart together into a chart folder

- helm create redis
- create
 - deployment.yaml
 - service.yaml

replace alll redis to {{.Value.appName}}

create a redis-value

```
deployment.yaml ... deployment-monkey.yaml ... service-video.yaml ...
1 deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] volumeMounts > {} 0 > {} m
12   |   labels:
13   |     app: {{ .Values.appName }}
14   spec:
15     containers:
16       - name: {{ .Values.appName }}
17         image: "{{ .Values.appImage }}:{{ .Values.appVersion }}"
18         ports:
19           - containerPort: {{ .Values.containerPort }}
20         volumeMounts:
21           - name: {{ .Values.volumeName }}
22             mountPath: {{ .Values.volumeName }}
23         volumes:
24           - name: {{ .Values.volumeName }}
25             emptyDir: {}
```

terminal Window Help

redis-values.yaml — online-shop-microservices

... deployment.yaml redis-values.yaml × values.yaml service.yaml

values > redis-values.yaml > # appReplicas

1 appName: redis-cart

ES PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh +

[\W]\$ helm install --dry-run -f values/redis-values.yaml charts/redis

helm install --dry-run =

check generated manifest without installing the chart

Difference of --dry-run and template

- ▶ --dry-run send files to K8s cluster, while template only validates it locally

.yaml
yaml
oy.yaml

yaml
value...
-val...
yaml

Ln 2, Col 15 Spaces: 2 UTF-8 LF YAML

2 ways deploy k8s cluster

- helm command one by one or in script file

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

[\W]$ helm install -f values/redis-values.yaml rediscart charts/redis
```

change mode
./install.sh

```
[\W]$ chmod u+x install.sh
[\W]$ ./install.sh
```

2. method helmfile

What is a Helmfile?

Declarative way for deploying helm charts

helmfile.yaml

```
releases:  
  - name: emailservice  
    chart: ./charts/app  
    values:  
      - ./values/emailservice.yaml  
  - name: paymentservice  
    chart: ./charts/app  
    values:  
      - ./values/paymentservice.yaml  
....
```

Define the desired state!

- ▶ Declare a definition of an entire K8s cluster
- ▶ Change specification depending on application or type of environment

Helmfile config schema (helmfile.json)

```
releases:
- name: rediscart
  chart: charts/redis
  values:
    - values/redis-values.yaml
```

helmfile.yaml > [] releases > {} 0 > [] values > {} 1 > appReplicas

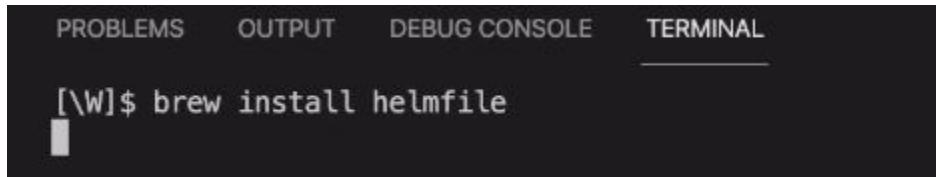
Helmfile config schema (helmfile.json)

```
1 releases:
2   - name: rediscart
3     chart: charts/redis
4     values:
5       - values/redis-values.yaml
6       - appReplicas: "1"
7       - volumeName: "redis-cart-data"
8
9   - name: emailservice
10    chart: charts/microservice
11    values:
12      - values/email-service-values.yaml
```

! helmfile.yaml X install.sh

```
! helmfile.yaml > [ ] releases > {} 2 > [ ] values > 0
values.
5   - values/redis-values.yaml
6
7   - name: emailservice
8     chart: charts/microservice
9     values:
10       - values/email-service-values.yaml
11
12   - name: cartservice
13     chart: charts/microservice
14     values:
15       - values/cart-service-values.yaml
```

install helmfile



```
[\W]$ brew install helmfile
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[\W]$ helmfile sync ←  
Building dependency release=rediscart, chart=charts/redis  
Building dependency release=cartservice, chart=charts/micro  
Building dependency release=emailservice, chart=charts/micro  
Building dependency release=currencyservice, chart=charts/micro  
Building dependency release=paymentservice, chart=charts/micro  
Building dependency release=recommendationservice, chart=charts/micro  
Building dependency release=productcatalogservice, chart=charts/micro  
Building dependency release=shippingservice, chart=charts/micro  
Building dependency release=checkoutservice, chart=charts/micro  
Building dependency release=adservice, chart=charts/microser  
Building dependency release=frontendservice, chart=charts/micro  
Affected releases are:  
  adservice (charts/microservice) UPDATED  
  cartservice (charts/microservice) UPDATED  
  checkoutservice (charts/microservice) UPDATED  
  currencyservice (charts/microservice) UPDATED  
  emailservice (charts/microservice) UPDATED  
  frontendservice (charts/microservice) UPDATED  
  paymentservice (charts/microservice) UPDATED
```

helmfile list
check again
kubectl get pod
check again:
loadbalancer IP address

helmfile destory

host the helm charts in the git repo

- with application code or
- separate git repo just for helm chart

How to feed into git CICD