

# **Certified Kubernetes Administrator**



# Setting the Base

Kubernetes is one of the most popular container orchestration tools in the industry.

It is used extensively in many medium to large-scale organizations.



# What this Course is All About?

This is a certification-specific course aimed at individuals who intend to gain the **Certified Kubernetes Administrator** certification.

This is a perfect course for CKA certification OR if you plan to start your journey in Kubernetes.



# Certification is Beneficial

We will be covering all the domains for the Certified Kubernetes Administrator certification.

1. Cluster Architecture, Installation & Configuration
2. Workloads and Scheduling
3. Servicing and Networking
4. Storage
5. Troubleshooting

# The Exciting Part

We have an **exam preparation section** to help you get prepared for the exam.

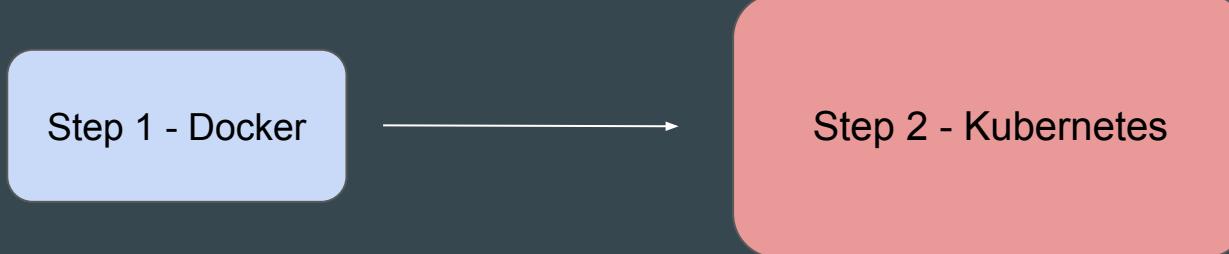
We also have practice tests available as part of the course.



# Point to Note

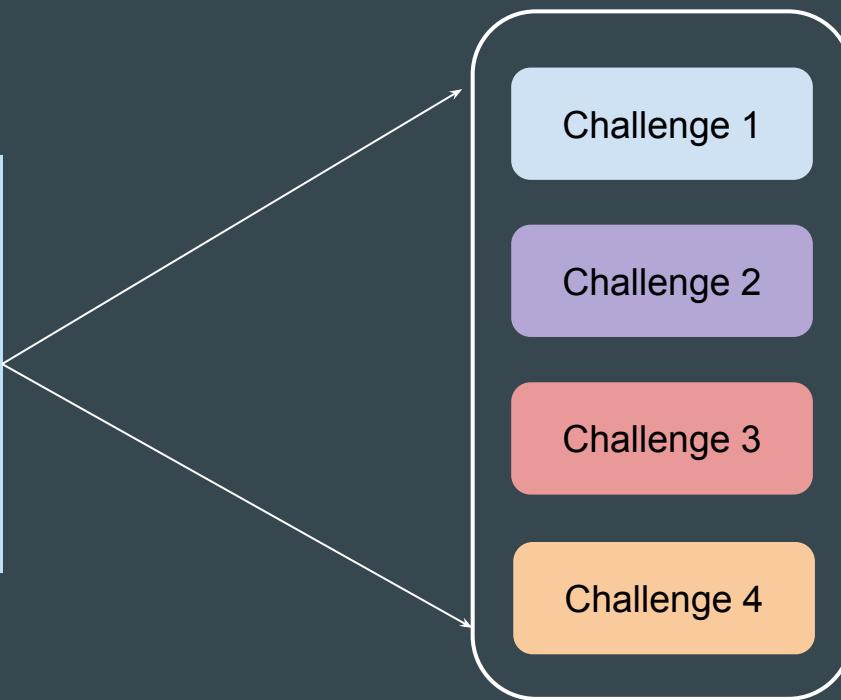
We start our journey on Kubernetes from absolute scratch.

Base Docker knowledge is a **prerequisite** for this Kubernetes.



# Lab Based Exams

CKA exam is a **lab-based exam** where you will have to solve multiple scenarios presented to you.



# About Me

- DevSecOps Engineer - Defensive Security.
- Teaching is one of my passions.
- I have total of 16 courses, and around 400,000+ students now.

Something about me :-



- Certified Kubernetes [Security Specialist, **Administrator**, Application Developer]
- HashiCorp Certified [Terraform Professional [Vault and Consul Associate]
- AWS Certified [Advanced Networking, Security Specialty, DevOps Pro, SA Pro, ...]
- RedHat Certified Architect (RHCA) + 13 more Certifications
- Part time Security Consultant

# **About the Course and Resources**

# 1 - Aim of This Course

The **primary aim** of this course is to learn.

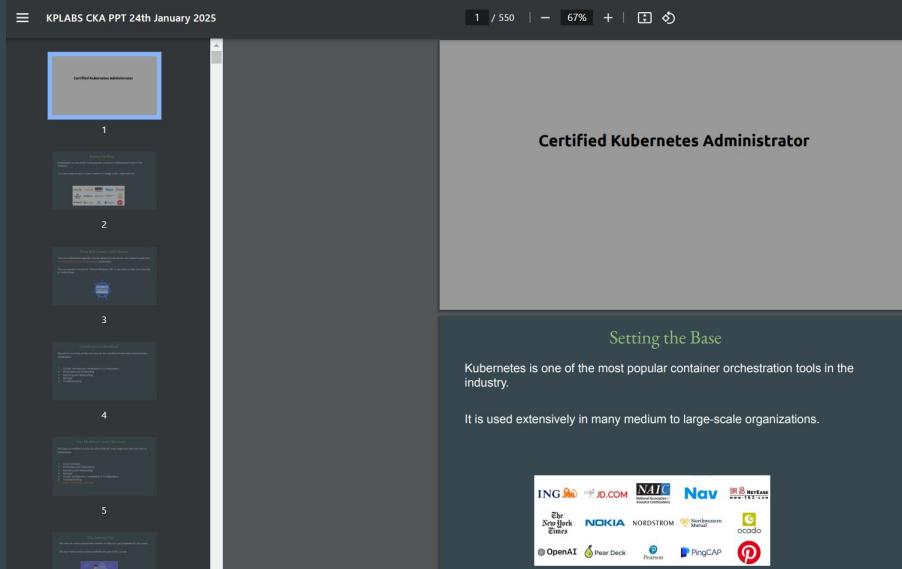
Certification is the byproduct of learning.



## 2 - PPT Slides PDF

ALL the slides that we use in this course is available to download as PDF.

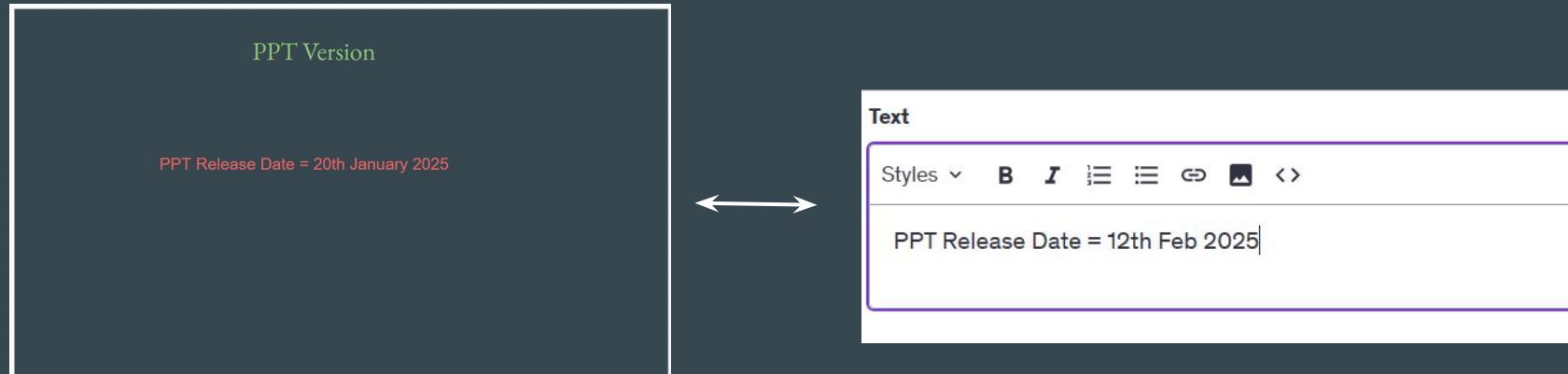
We have around 500+ slides that are available to download.



## 3 - PPT Version

The course is updated regularly and so are the PPTs.

We add the PPT release date in the PPT itself and the lecture from which you download the PPTs.

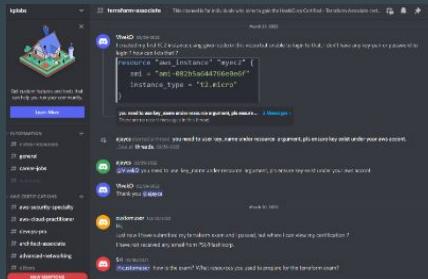


## 4 - Our Community (Optional)

You can **join our Discord community** for any queries / discussions. You can also connect with other students going through the same course in Discord (Optional)

Discord Link: <https://kplabs.in/chat>

Category: #cka



# 5 - Course Resource - GitHub

All the code that we use during practicals have been added to our GitHub page.

Section Name in the Course and GitHub are same for easy finding of code.

📁 Domain 1 - Core Concepts	.
📁 Domain 2 - Workloads & Scheduling	.
📁 Domain 3 - Services and Networking	.
📁 Domain 4 - Security	.
📁 Domain 5 - Storage	.
📁 Domain 6 - Cluster Architecture, Installation & ...	.
📁 Domain 7 - Logging and Monitoring	.
📁 Domain 8 - Troubleshooting	.
📁 practice-tests	Update core-concepts.md

## 6 - Local Structure

We use folder of **kplabs-k8s** as a base for creating and managing K8s files.

**Visual Studio Code** is the default code editor used for this course.

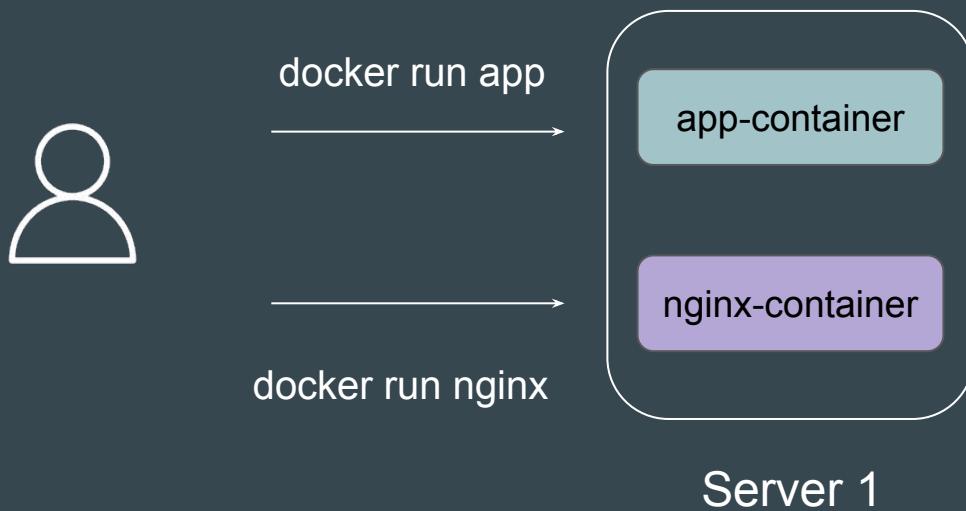


```
C: > kplabs-k8s > ! lb-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: simple-service
spec:
  type: LoadBalancer
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 80
```

# **Overview of Container Orchestration**

# Setting the Base

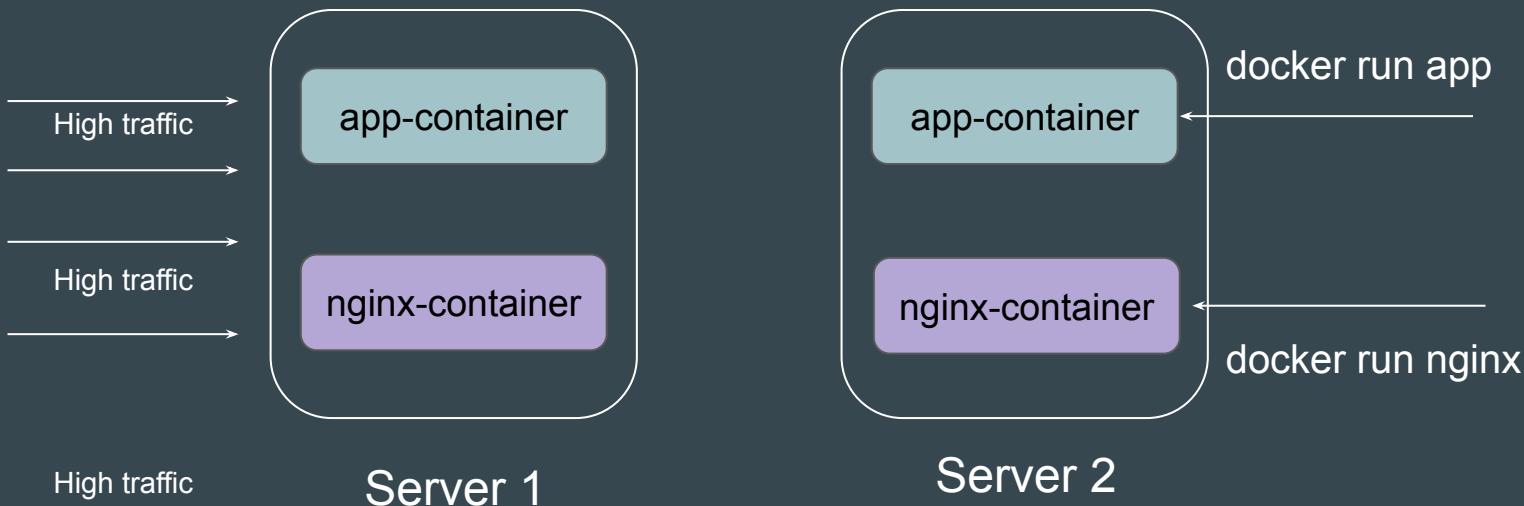
During the initial times, **users relied on manual** container management or basic scripts to handle tasks like deployment, scaling, networking, and monitoring.



# Challenge 1 - Manual Scaling

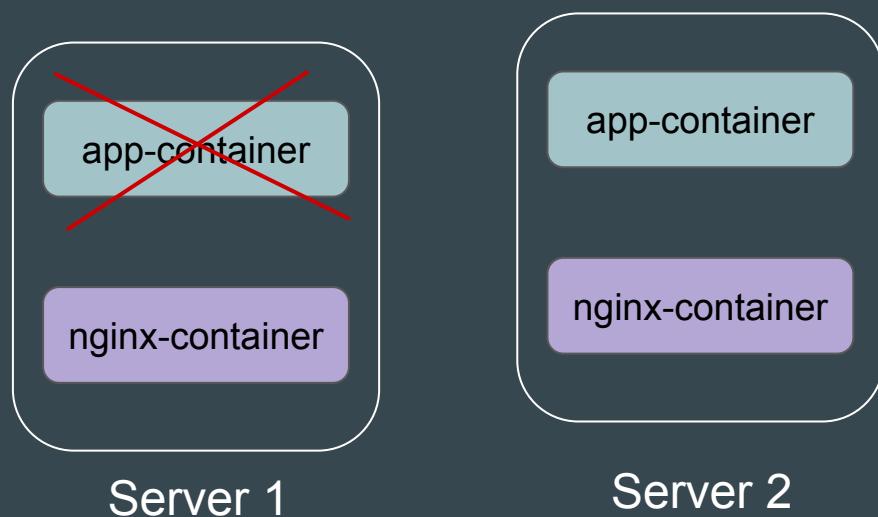
If an application needed to handle higher traffic, developers had to manually start additional containers.

This involved identifying which servers had enough resources, deploying new containers, etc.



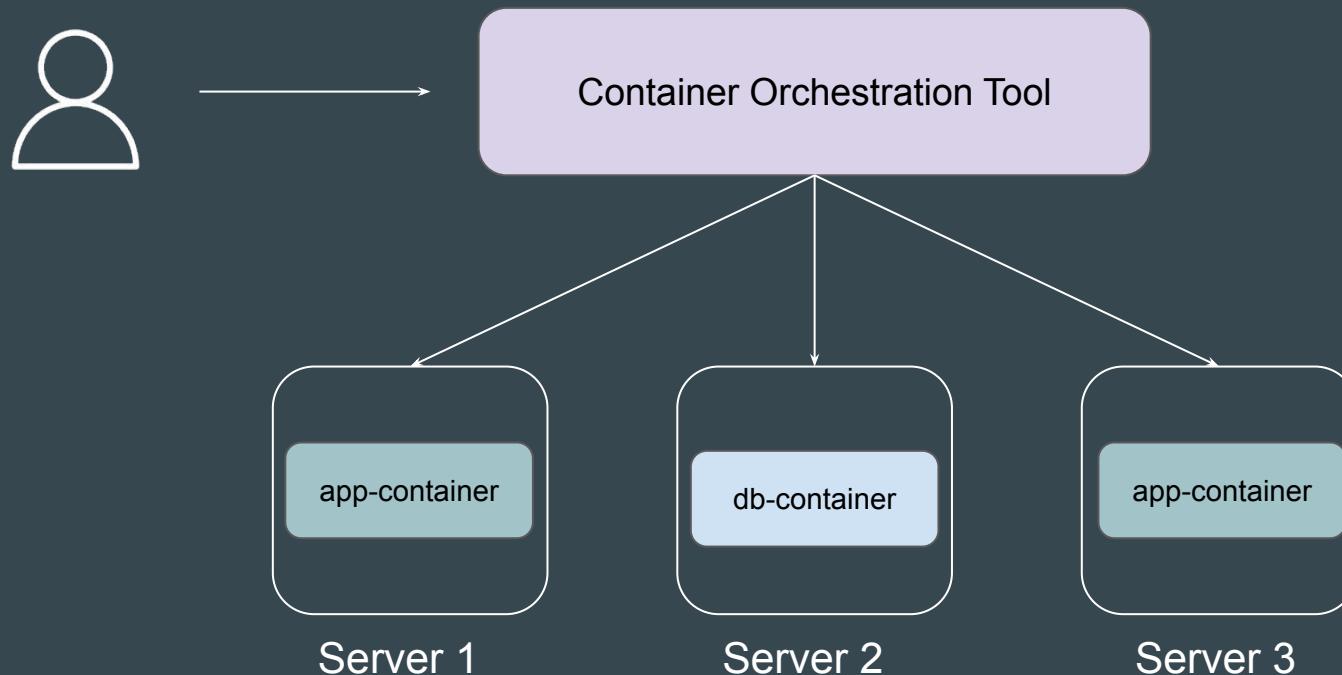
## Challenge 2 - Lack of Fault Tolerance

If a container failed (e.g., due to a crash or resource exhaustion,), it wouldn't automatically restart without manual intervention in most of the cases.



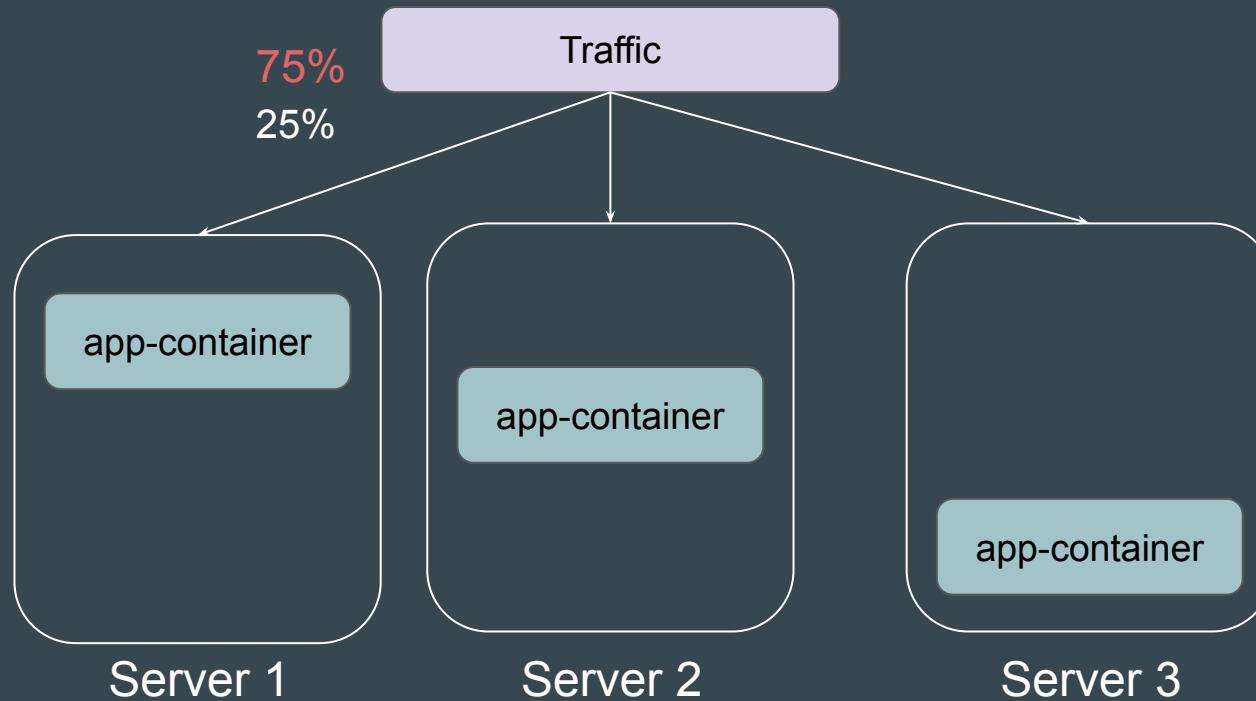
# Introducing Container Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers.



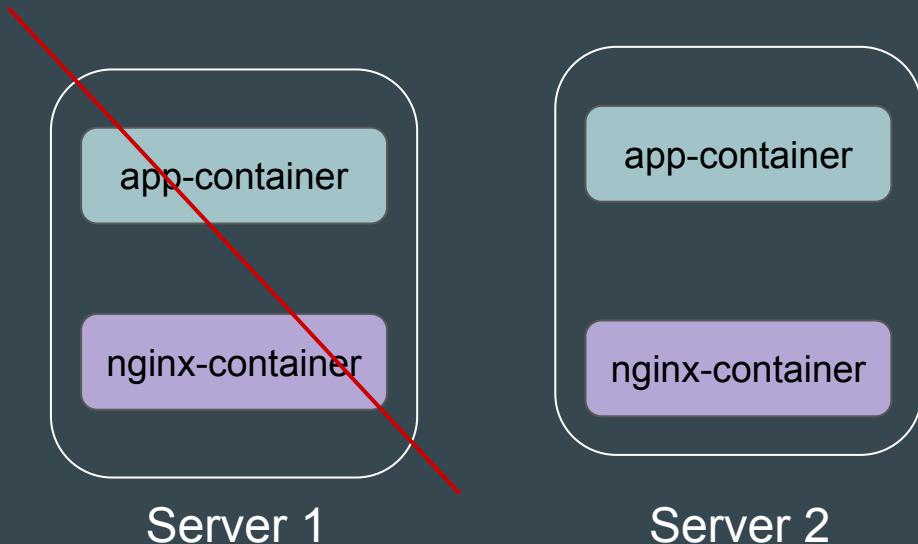
# Solving Challenge - Scaling

Orchestration tools CAN automatically scale containers up or down based on defined policies and real-time traffic.



# Solving Challenge - Fault Tolerance

Failed containers are automatically restarted or replaced to ensure high availability.



# Container Orchestration is LOT More

Container orchestration is the process of automating the networking and management of containers so you can deploy applications at scale

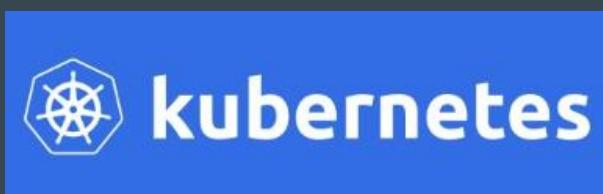
- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Load balancing and traffic routing
- Monitoring container health
- Keeping interactions between containers secure

# Container Orchestration Tools

There are multiple set of Container Orchestration tools available in the Industry.

Based on the organization and requirement, you can choose tools per preference.

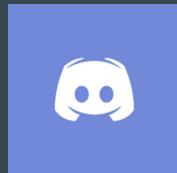
Docker Swarm



Amazon ECS

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



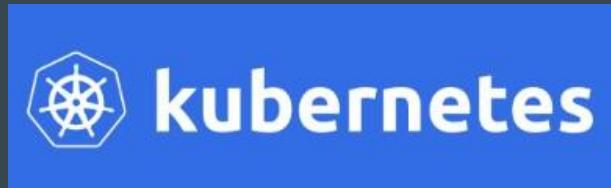
[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Introduction to Kubernetes**

# Setting the Base

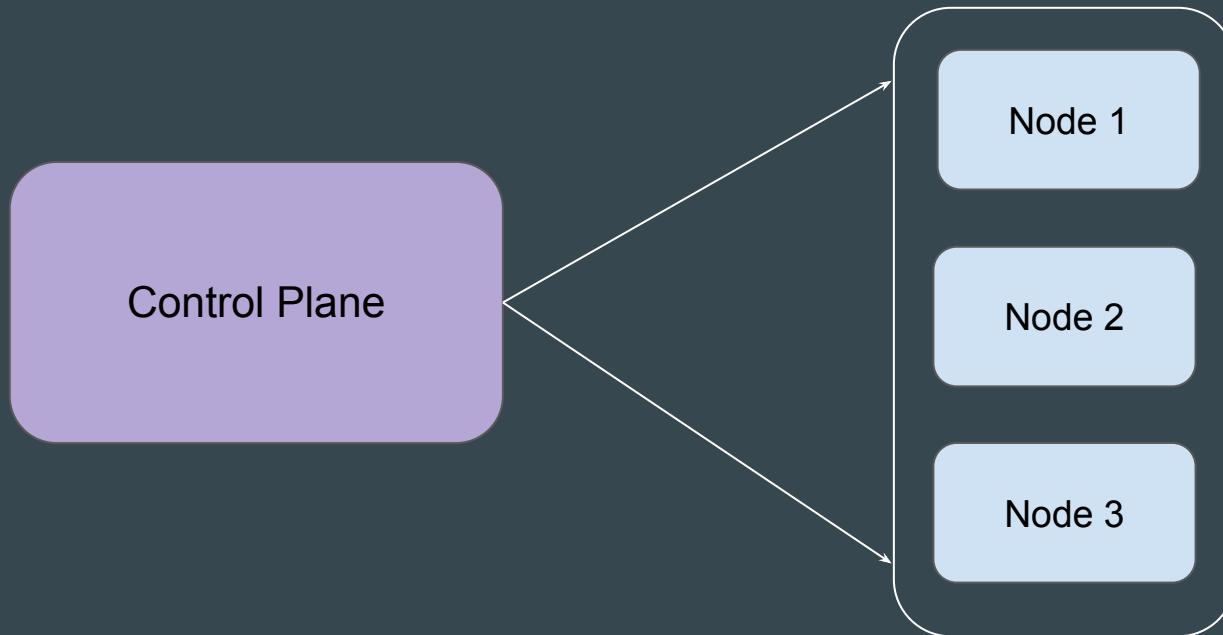
Kubernetes (K8s) is an open-source **container orchestration engine** developed by Google.

It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation.



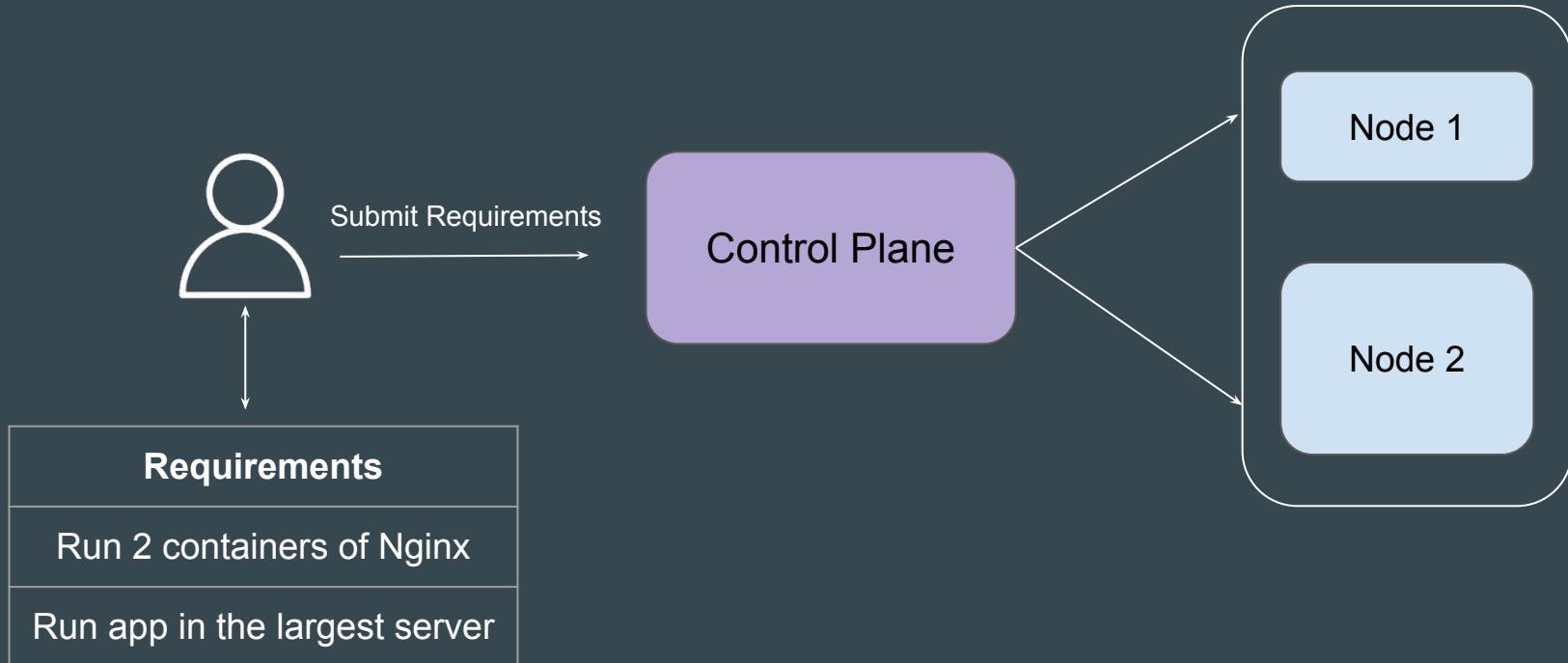
# Architecture of Kubernetes

A Kubernetes cluster consists of a **control plane** + a set of worker machines, called nodes, that run containerized applications



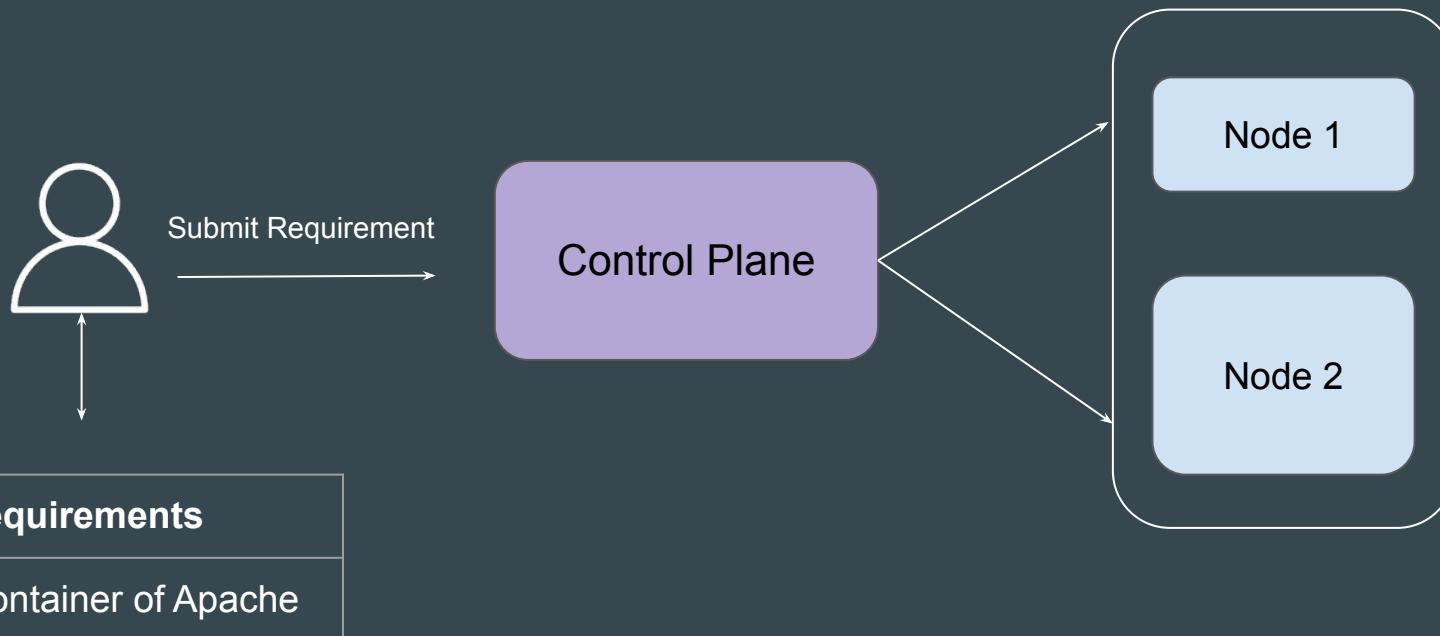
# Basic Workflow

User communicates to Control Plane and provides necessary instructions to run containerized applications.



# Example - Run 1 Container of Apache

If you have instructed Kubernetes to **run 1 container** of Apache, Kubernetes will launch it in one of the worker nodes and will regularly monitor the state of that container to ensure it always runs.



# Kubernetes is Awesome

Kubernetes provides amazing set of features and is designed on the same principles that allow Google to **run billions of containers** a week.

Some of the popular features include:

- Pod Auto-Scaling
- Service discovery and load balancing
- Self-Healing of Containers
- Secret management
- Automated rollouts and rollbacks



# **Installation Options for Kubernetes**

# Analogy - Eating your Favorite Food Dish

You want to **eat your favourite food** dish.

What are the options:

1. Go to store, get ingredients and prepare from scratch.
2. Get ready-made mix, make it hot.
3. Order it from the restaurant.



# Kubernetes Reference

You want to launch a Kubernetes cluster.

What are the options:

1. Go to K8s website, download individual components and integrate them one by one.
2. Use tools like Minikube, Kubeadm to quickly setup K8s cluster for you.
3. Use Managed Kubernetes Service.

# Option 1 - Managed Kubernetes Service

Various providers like AWS, IBM, GCP and others provides **managed** Kubernetes clusters.

Analogy: Order Ready-Made Food from Store.



# Option 2 - K8s Development Tools

Various tools like Minikube, K3d allows you to quickly setup the Kubernetes for local testing.

Analogy: Get ready-made mix, make it hot.

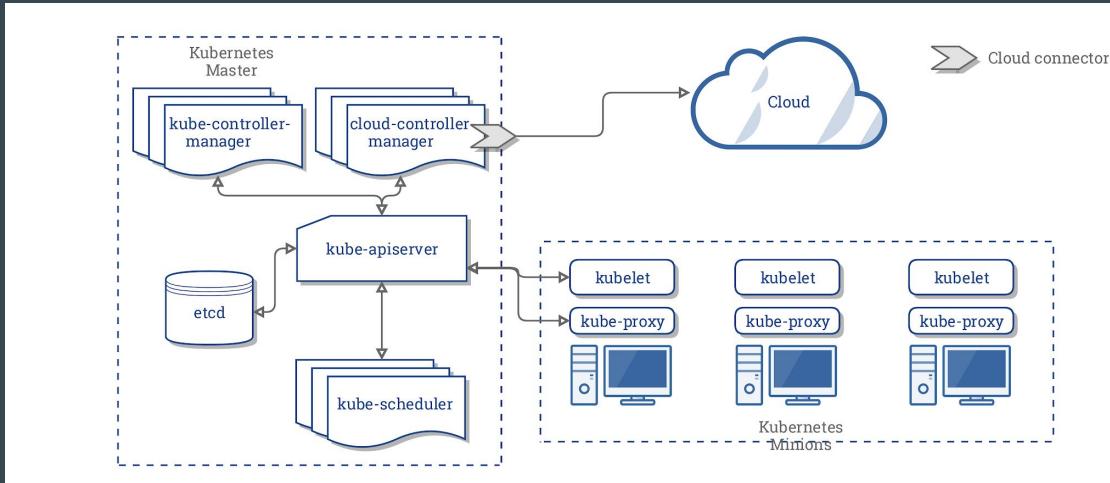


```
~ ➔ time minikube start
😊 minikube v1.13.0 on Darwin 10.15.6
💡 Using the docker driver based on user configuration
👍 Starting control plane node minikube in cluster minikube
🔥 Creating docker container (CPUs=2, Memory=3892MB) ...
🔥 Preparing Kubernetes v1.19.0 on Docker 19.03.8 ...
🔥 Verifying Kubernetes components...
🌟 Enabled addons: default-storageclass, storage-provisioner
💡 kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🎉 Done! kubectl is now configured to use "minikube" by default

Executed in 23.96 secs fish external
    usr time 1.66 secs 237.00 micros 1.66 secs
    sys time 0.78 secs 943.00 micros 0.78 secs
```

# Option 3 - Setup K8s from Scratch

In this approach, you install and configure each Kubernetes component individually.



# **Choosing Right Cloud Provider for Practicals**

# Installation Options for Kubernetes

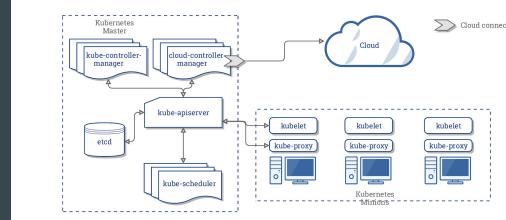
There are **three primary ways** to configure Kubernetes.

1. Using Managed Kubernetes Service from a Provider
2. Use tools like [Minikube](#), [Kubeadm](#) to quickly setup K8s cluster for you.
3. Setup Kubernetes Cluster from Scratch.



```
time minikube start
minikube v1.13.0 on Darwin 10.15.6
Using the docker driver based on user configuration
Starting control plane node minikube in cluster minikube
Creating docker container (CPUs=2, Memory=3892MB)...
Preparing Kubernetes v1.19.0 on Docker 19.03.8 ...
Verifying Kubernetes components...
* Enabled addons: default-storageclass, storage-provisioner
  kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
  Done! kubectl is now configured to use "minikube" by default

Executed in 23.96 secs  fish      external
  usr time  1.66 secs 237.00 microseconds  1.66 secs
  sys time  0.78 secs 943.00 microseconds  0.78 secs
```



# Constraints of Choosing Cloud Provider

1. Easy to Use.
2. Cost Effective.
3. Free Credits to Get Started.

# Option 1 - Amazon EKS

Managed Kubernetes service provided by AWS.

## Amazon Elastic Kubernetes Service

Start, run, and scale Kubernetes without thinking about cluster management

[Get started with Amazon EKS](#)

# Option 2 - Digital Ocean

Managed Kubernetes service provided by Digital Ocean.



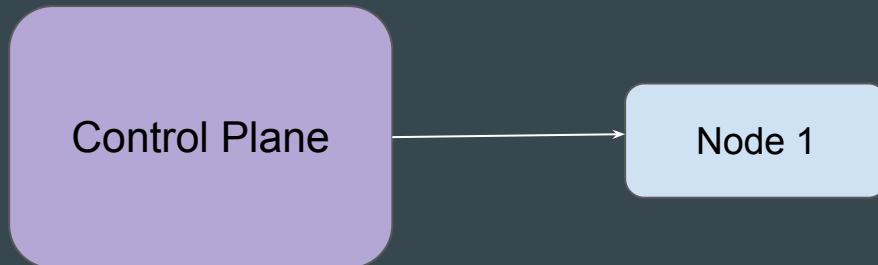
**Kubernetes at scale managed for  
you**

Powerful, cost-effective cloud infrastructure optimized for startups, growing digital businesses, and independent software vendors (ISVs).

# Why Digital Ocean?

Kubernetes Control Plane is completely free.

You will be charged only for the Worker Nodes that you run..



# Initial Free Credits

Digital Ocean provides decent amount of free credits for new users.

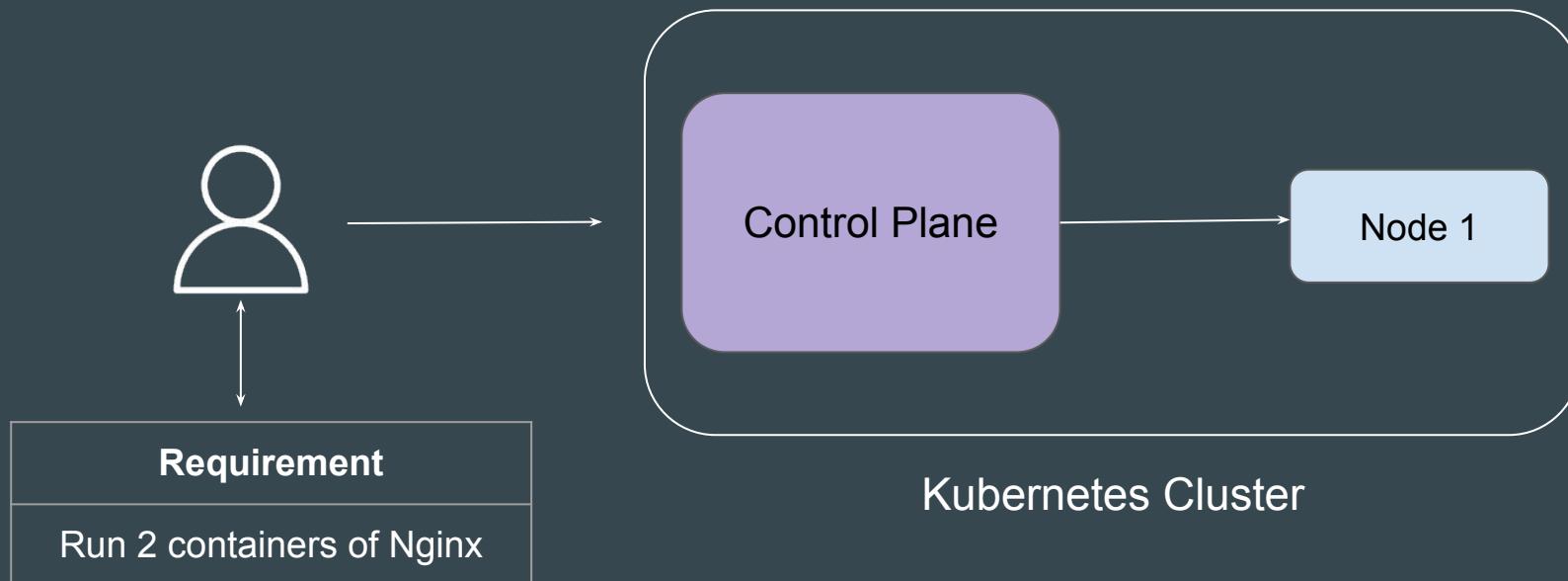
You can literally complete this entire course and ALL practicals for free.



# **Connectivity Options for Kubernetes**

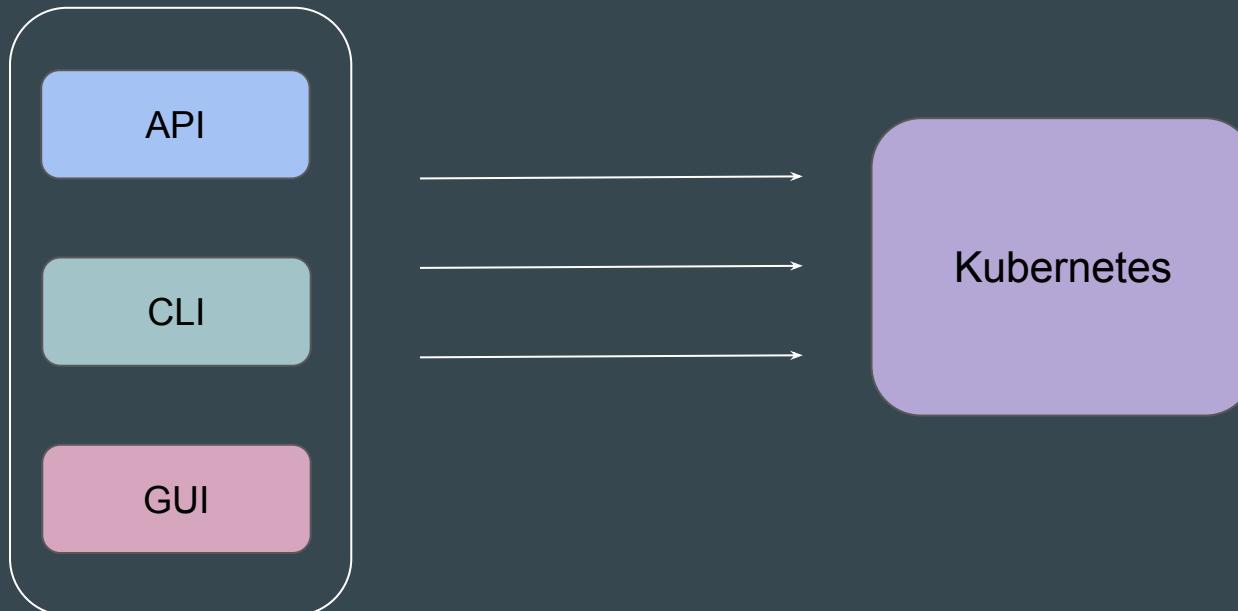
# First Important Step - Connect to Cluster

The **first important step** after launching Kubernetes cluster is to **connect to the Control Plane**.



# Options for Connectivity

There are **three** primary ways to connect to your Kubernetes cluster.



# Option 1 - API

You can use utilities like curl to directly send request to the API.

```
root@kubeadm-master:~# curl http://localhost:8080/api/v1/namespaces/default/pods
{
    "kind": "PodList",
    "apiVersion": "v1",
    "metadata": {
        "resourceVersion": "1346"
    },
    "items": [
        {
            "metadata": {
                "name": "nginx",
                "namespace": "default",
                "uid": "67850778-3e32-4364-9a91-04c2cb9644d8",
                "resourceVersion": "1338",
                "creationTimestamp": "2024-12-26T05:49:11Z",
                "labels": {
                    "run": "nginx"
                },
            }
        }
    ]
}
```

## Option 2 - CLI

To connect to Kubernetes using CLI, you need an important tool named **kubectl**

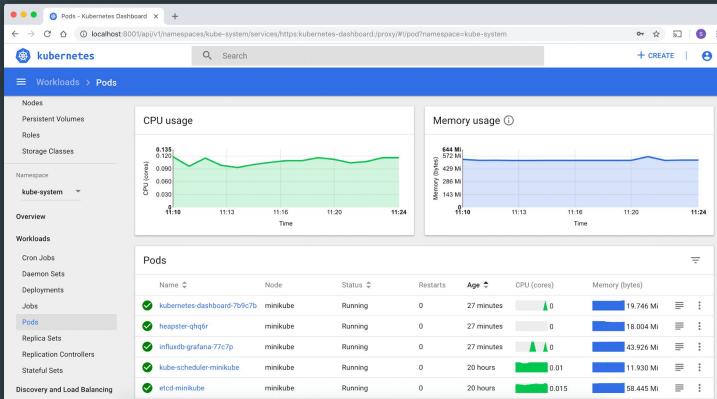


```
root@kubeadm-master:~# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          82s
```

# Option 3 - GUI

Dashboard is a web-based Kubernetes user interface.

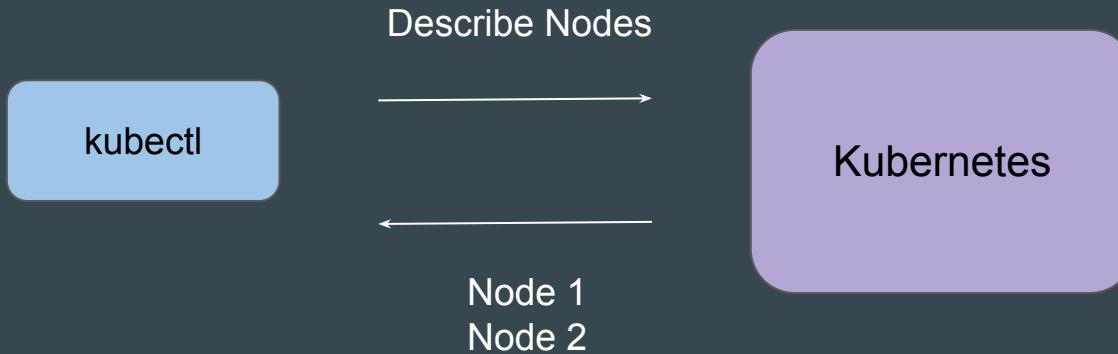
You can use Dashboard to deploy, troubleshoot containerized applications and manage the cluster resources



# **Overview of kubectl**

# Basics of Kubectl

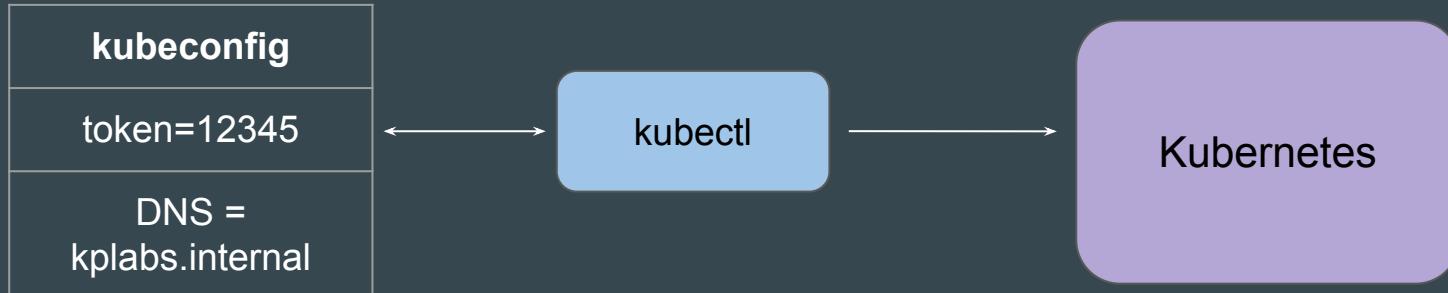
The Kubernetes command-line tool, **kubectl**, allows you to run commands against Kubernetes clusters.



# Important Part - Authentication

Kubectl needs **two important details** while connecting to Kubernetes Cluster

1. DNS / IP Address of Kubernetes Control Plane
2. Authentication Credentials.



# Default Path of Kubeconfig File

Kubectl by default will look for the kubeconfig file in a file named **config** inside **.kube** folder in your **home** directory.

`~/.kube/config`

```
root@kubeadm-master:~# ls -l ~/.kube/config
-rw----- 1 root root 5658 Dec 26 05:46 /root/.kube/config
```

# Referencing to Custom Config File

You can also refer to custom config file using the `--kubeconfig` flag

```
C:\kplabs-k8s>kubectl get nodes --kubeconfig "custom-kubeconfig"
NAME                  STATUS    ROLES      AGE     VERSION
pool-i625o5obd-eob8a  Ready     <none>    3h6m    v1.31.1
```

# **Configuring Kubernetes using Minikube**

# Basics of Minikube

minikube quickly **sets up a local Kubernetes cluster** on macOS, Linux, and Windows.



A terminal window on a dark background showing the output of the command `time minikube start`. The output includes:

- Minikube version: v1.13.0 on Darwin 10.15.6
- Driver used: docker
- Control plane node started: minikube
- Docker container created: (CPUs=2, Memory=3892MB)
- Kubernetes version: v1.19.0 on Docker 19.03.8
- Components verified
- Addons enabled: default-storageclass, storage-provisioner
- Kubectl not found, with a suggestion to use `minikube kubectl -- get pods -A`
- Final message: "Done! kubectl is now configured to use "minikube" by default"

At the bottom, a table shows execution times:

Executed in	23.96 secs	fish	external
usr time	1.66 secs	237.00 micros	1.66 secs
sys time	0.78 secs	943.00 micros	0.78 secs

# **Basics of Kubernetes Pods**

# Setting the Base - Docker Analogy

You have recently **installed Docker** in your system.

What is the first thing that you might do?



`docker run nginx`

---



System with Docker

# Setting the Base - Kubernetes Analogy

You have recently **configured** Kubernetes cluster.

What is the first thing that you might do?



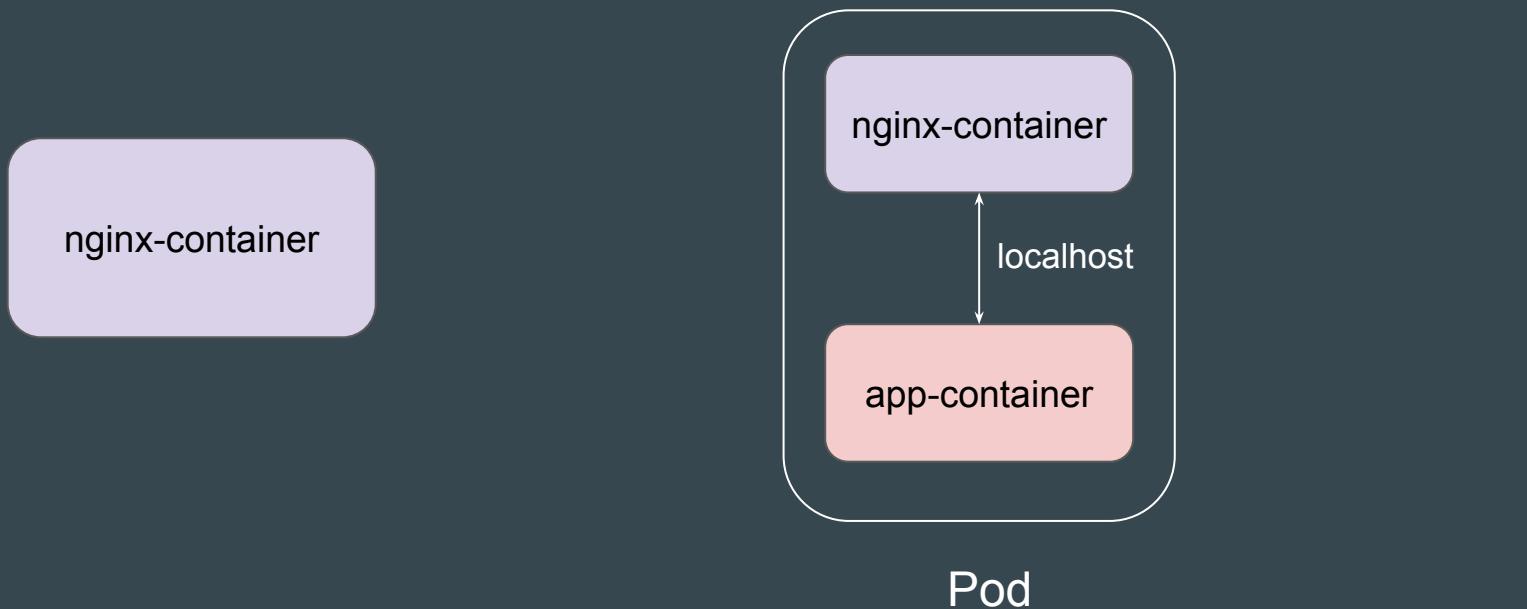
kubectl run nginx --image=nginx



System with Kubernetes

# Difference Between Container and Pods

A Pod can contain **one or more Docker containers** that share the same network namespace and storage volumes



# Comparison Table - Docker vs Kubernetes Commands

Docker Command	Kubernetes Command	Purpose
docker run nginx	kubectl run nginx --image=nginx	Create and run a container/pod
docker ps	kubectl get pods	List running containers/pods
docker logs <container>	kubectl logs <pod>	View logs
docker inspect <container>	kubectl describe pod <pod-name>	Get detailed info
docker exec -it <container> bash	kubectl exec -it <pod-name> -- bash	Execute interactive shell
docker rm <container>	kubectl delete pod <pod-name>	Remove container/pod

# Points to Note - Kubernetes Pod

A Pod always runs on a Node.

A Node is a worker machine in Kubernetes.

Each Node is managed by the Kubernetes Control Plane.

A Node can have multiple pods.

# **Multiple Ways to Create Kubernetes Objects**

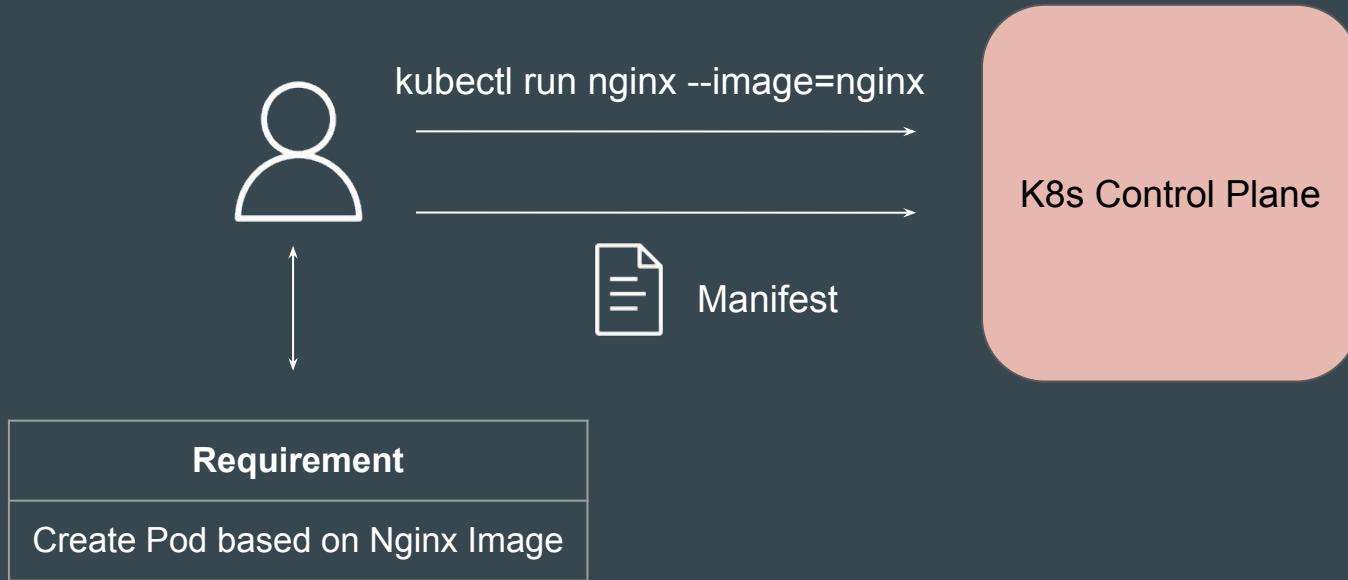
# Setting the Base

We have been using simple approach of using `kubectl` command to create object like Pods in Kubernetes.

```
root@minikube:~# kubectl run nginx --image=nginx  
pod/nginx created
```

# 2 Primary Ways to Create Object

Use **kubectl run** command or supply kubectl with **Manifest file** that contains information of resource to be created.



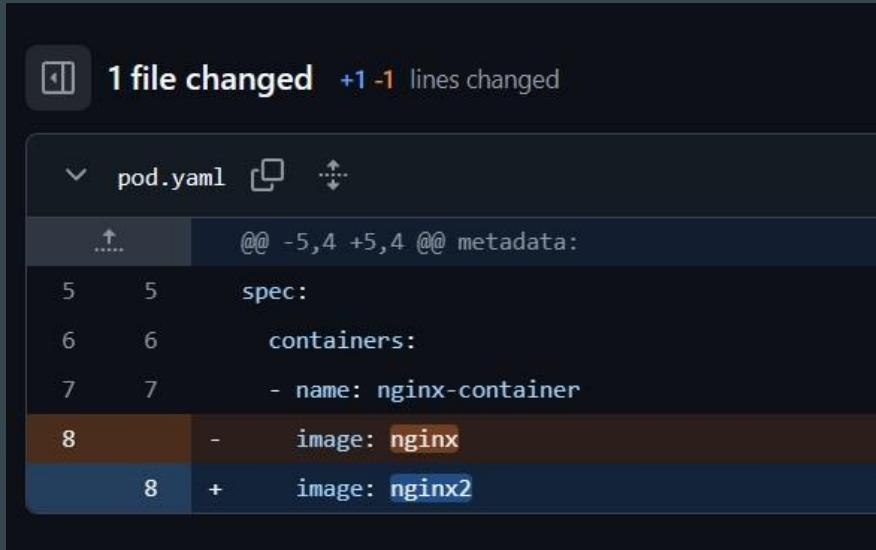
# Manifest File Example

A Kubernetes manifest file is a [YAML or JSON file](#) that defines the desired state of a Kubernetes object.

```
! pod.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: nginx
  spec:
    containers:
      - name: nginx-container
        image: nginx
```

# Benefits of Manifest File - Version Control

Manifest files can be stored in version control systems like Git, allowing you to track changes to your infrastructure over time and easily roll back to previous versions.



A screenshot of a GitHub interface showing a diff between two versions of a file named `pod.yaml`. The title bar indicates "1 file changed +1 -1 lines changed". The diff shows the following changes:

```
@@ -5,4 +5,4 @@ metadata:  
 5   5     spec:  
 6   6       containers:  
 7   7         - name: nginx-container  
 8 -   8           image: nginx  
 8 +   8           image: nginx2
```

# Benefits of Manifest File - Multiple Resources

You can define multiple rest of dependent resource objects that you want to create in a single manifest file.

```
! demo.yaml •
!
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80

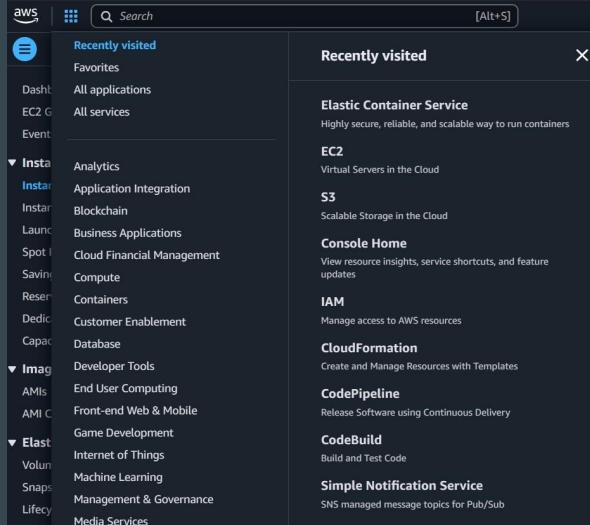
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
```

# **Basics of Kubernetes Resource Types**

# Sample Analogy - Hosting Provider

During the early times, one of the primary offerings of hosting providers was servers / virtual machines.

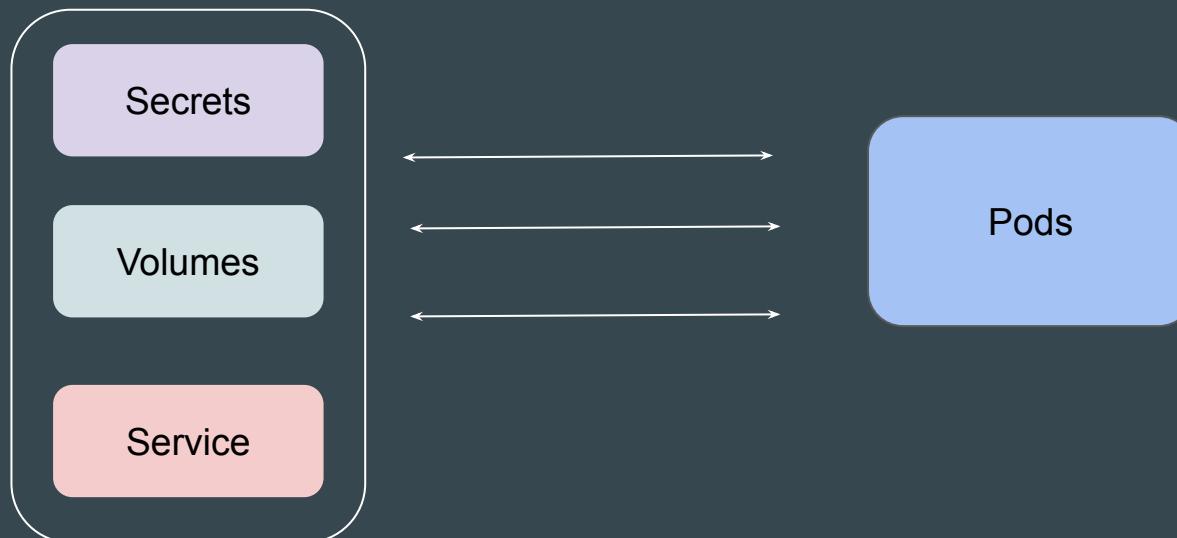
Nowadays, there are 100s of services offered by Cloud providers that aid in the entire lifecycle management of applications and custom use-cases of organizations.



# Understanding from Kubernetes Perspective

Although Pods allow users to host custom applications in containers, it might not always be enough.

Based on use-case, Pod might also require access to other Kubernetes Objects.



# Kubernetes Resource Types

Kubernetes has a broad set of **resource types** that organizations can use based on their requirements.

## Pod

The smallest deployable unit in Kubernetes, representing a single instance of a running process.

## Deployment

Manages multiple replicas of Pods and supports rolling updates.

## Service

Exposes Pods to the network and provides stable load balancing.

## ConfigMap

Stores non-sensitive configuration data for applications.

## Secret

Stores sensitive data like passwords and keys securely.

## PersistentVolume

Represents storage resources for persistent data in the cluster.

## PersistentVolumeClaim

Requests a specific amount of storage from a PersistentVolume.

## Ingress

Manages external HTTP and HTTPS traffic to Services in the cluster.

## Namespace

Provides a way to group and isolate resources within the cluster.

# Reference Screenshot

```
C:\Users\zealv>kubectl api-resources
NAME                      SHORTNAMES
bindings
componentstatuses         cs
configmaps                cm
endpoints                 ep
events                    ev
limitranges               limits
namespaces                ns
nodes                     no
persistentvolumeclaims    pvc
persistentvolumes          pv
pods                      po
podtemplates
replicationcontrollers    rc
resourcequotas            quota
secrets
serviceaccounts           sa
services                  svc
```

# **Basic Structure of a Manifest File**

# Sample - Pod Manifest File

```
! pod.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: nginx
  spec:
    containers:
      - name: nginx-container
        image: nginx
```

# Comparing Manifest Structure with Pod Manifest

! manifest.yaml

```
apiVersion: [API version]
kind: [Resource type]
metadata:
  name: [Resource name]
spec:
  [Resource-specific configuration]
```



! pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx
```

# Basic Structure of Manifest File

! manifest.yaml

```
apiVersion: [API version]
kind: [Resource type]
metadata:
  name: [Resource name]
spec:
  [Resource-specific configuration]
```



Component	Description
apiVersion	Specifies which version of the Kubernetes API to use
kind	Type of resource you are creating.
metadata	Information about resource
spec	Details about resource to be created

# **Generating Manifest File through CLI Command**

# Setting the Base

CLI commands are easy to run, and Manifest file provides a lot of benefits for organizations and team collaboration.

Finding an easier way to generate a manifest file is needed.

Kubectl Command

Generate

```
! pod.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: nginx
  spec:
    containers:
      - name: nginx-container
        image: nginx
```

# Basics of Dry Runs

The `--dry-run=client` option allows you to validate a Kubernetes resource definition without actually applying it to the cluster.

```
C:\kplabs-k8s>kubectl run nginx --image=nginx --dry-run=client  
pod/nginx created (dry run)
```

# Exploring Output of Kubectl command

By default, when you run the basic **kubectl** command to create an object like Pod, in the output, it will just print the confirmation message.

```
C:\>kubectl run nginx --image=nginx  
pod/nginx created
```

# Kubectl with Output of YAML

When kubectl command is used with output of yaml, the command doesn't just create the resource in the cluster; it also **prints the full YAML configuration of the created resource** to your terminal.

```
C:\kplabs-k8s>kubectl run nginx --image=nginx -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-01-01T04:11:42Z"
  labels:
    run: nginx
  name: nginx
  namespace: default
  resourceVersion: "1599760"
  uid: 2f321eaf-15b5-457d-8d58-d644ede8a6cc
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
```

# Combining Dry Run Output of YAML

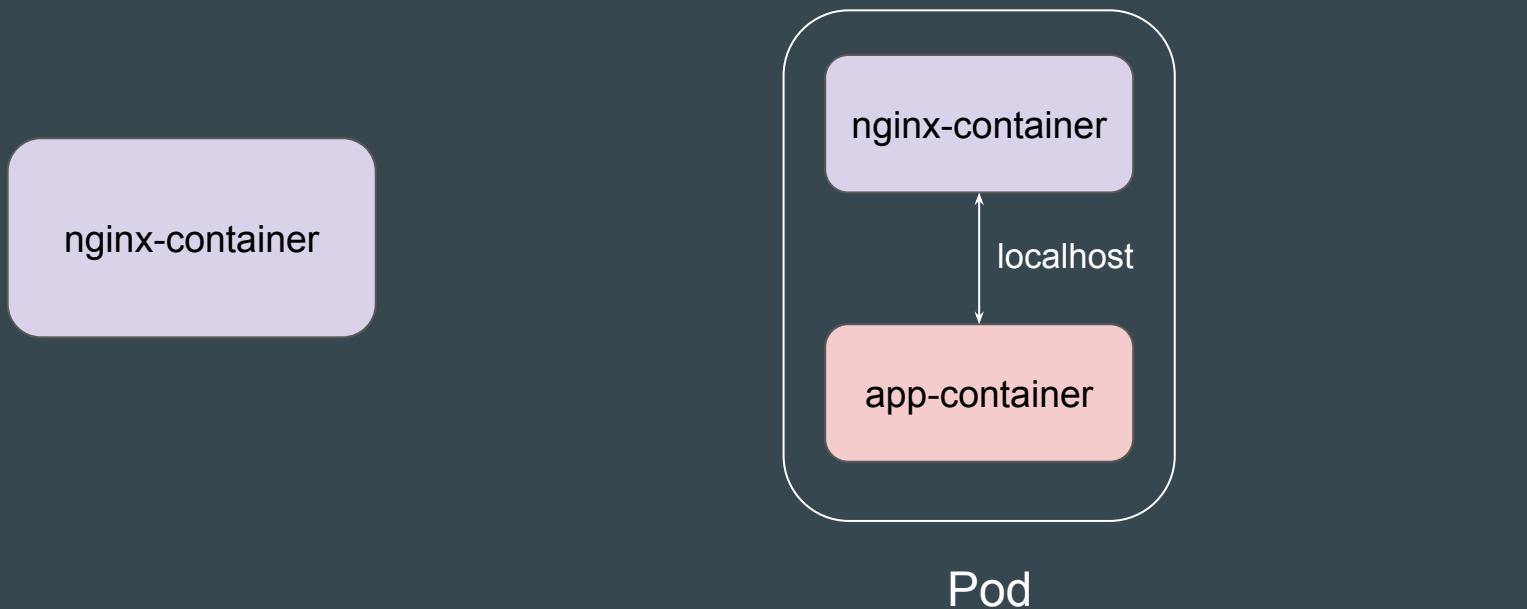
When `--dry-run=client` is combined with `-o yaml`, it will generate manifest file for you associated with the command without actually deploying the object in Kubernetes.

```
C:\>kubectl run nginx --image=nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
    name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

# **Multi-Container Pods**

# Difference Between Container and Pods

A Pod can contain **one or more Docker containers** that share the same network namespace and storage volumes



# Default Option with kubectl Command

The `kubectl run` command allows us to run a **single-container based Pods**.

For Pods with multiple container, you need to use Manifest File based approach.



# Multi-Container Pod Configuration

To create a multi-container based Pods, you can defined additional details in container definition.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx
```



```
! multi-container-pods.yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
    - name: redis-container
      image: redis
```

# Exec into a Container

By default, when you run the `kubectl exec` command, it will connect to the first container.

```
C:\kplabs-k8s>kubectl exec -it multi-container-pod -- bash
Defaulted container "nginx-container" out of: nginx-container, redis-container
root@multi-container-pod:/#
```

# Exec into Other Containers in Pod

To connect with other containers of Pod, you can add **-c flag with <container-name>** as part of the `kubectl exec` command

```
C:\kplabs-k8s>kubectl exec -it multi-container-pod -c redis-container -- bash  
root@multi-container-pod:/data#
```

# **Overview of Commands and Arguments**

# Setting the Base

Whenever a Docker Image is built, it can have a certain ENTRYPPOINT / CMD instructions set that defines what container needs to run when it starts.

🚢 Dockerfile > ...

```
FROM nginx:1.10.1-alpine
COPY index.html /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```



nginx-container-image

# Running Docker Container from Image

Whenever a container starts from Docker Image, it uses the appropriate ENTRYPOINT / CMD Instructions to start the appropriate process inside the container.



## Example 2 - Short Lived CMD Instruction

In the following example, a Docker image is built using a busybox image with CMD instruction that pings google.com three times.



Dockerfile > ...

```
FROM busybox:latest
CMD ["ping", "-c", "3", "google.com"]
```

# What is no long running process in CMD?

Once ping finishes sending the 3 requests and receives the responses (or timeouts), there's nothing else defined in the CMD instruction.

Since there's no additional process keeping the container running, it exits as soon as the initial command finishes.

busybox-container-image



```
root@docker:~/docker-file# docker run busybox:custom
PING google.com (142.250.70.110): 56 data bytes
64 bytes from 142.250.70.110: seq=0 ttl=117 time=15.066 ms
64 bytes from 142.250.70.110: seq=1 ttl=117 time=14.161 ms
64 bytes from 142.250.70.110: seq=2 ttl=117 time=14.175 ms

--- google.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 14.161/14.467/15.066 ms
```

# Entrypoint and CMD in Dockerfile

The ENTRYPOINT is always executed when the container starts.

If you provide a command when running the container, that command is appended as arguments to the ENTRYPOINT.

```
🐳 Dockerfile > ...
  FROM ubuntu
  ENTRYPOINT ["/bin/echo"]
  CMD ["hello", "world"]
```

# Example - Entrypoint vs CMD in Dockerfile

If you run `docker run <image>`, the output will be: hello world

If you run `docker run <image> how are you`, the output will be: how are you

Dockerfile > ...

```
FROM ubuntu
ENTRYPOINT ["/bin/echo"]
CMD ["hello", "world"]
```

```
root@docker:~# docker run demo
hello world
```

```
root@docker:~# docker run demo how are you
how are you
```

# Command and Arguments in Kubernetes

When you create a Pod, you can define a command and arguments for the containers that run in the Pod.

The command field corresponds to ENTRYPPOINT, and the args field corresponds to CMD in some container runtimes.

```
! cmd-args.yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
spec:
  containers:
  - name: demo
    image: busybox
    command: ["/bin/echo"]
    args: ["hello", "world"]
```

# Point to Note

The command argument defined overrides the default ENTRYPPOINT.

The args(arguments) overrides the default CMD.

# **Commands and Arguments - Practical**

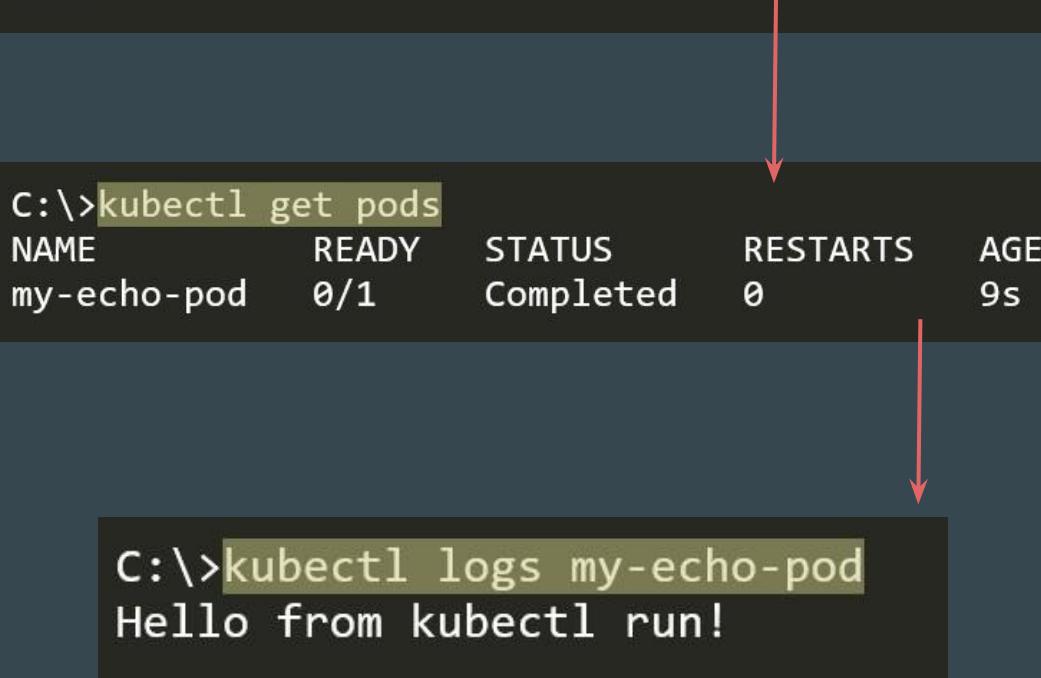
# Defining Commands and Arguments

The following command denotes the basic syntax to define command and arguments with kubectl run command:

```
kubectl run nginx --image=nginx --command -- <command> <args>
```

# Example - Create Pod with Specific Command

```
C:\>kubectl run my-echo-pod --image=busybox:latest --command -- echo "Hello from kubectl run!"  
pod/my-echo-pod created
```



```
C:\>kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-echo-pod	0/1	Completed	0	9s

```
C:\>kubectl logs my-echo-pod  
Hello from kubectl run!
```

# Defining in Manifest File

Use the command and args field to define the necessary commands and arguments.

```
! cmd-args.yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
spec:
  containers:
  - name: demo
    image: busybox
    command: ["/bin/echo"]
    args: ["hello", "world"]
```

# **More Clarity - Commands and Arguments**

# 1 - Defining Commands and Arguments

In Kubernetes, when defining the command (or args) for a container in a Pod specification, there are two primary ways to specify them:

- Array (JSON array notation, square brackets []):
- Multi-Line YAML List (- for each new item)

```
! cmd-args.yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: busybox
      command: ["/bin/sh", "-c", "echo Hello Kubernetes!"]
```

```
! cmd-args.yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: busybox
      command:
        - "/bin/sh"
        - "-c"
        - "echo Hello Kubernetes!"
```

# High-Level Comparison

Feature	Array Notation ([])	Multi-Line YAML List (-)
Syntax Style	JSON-style array with square brackets.	YAML-style list where each item is on a new line, prefixed with -.
Readability	Compact but harder to read for long commands.	More human-readable for long commands.
Preference	Suitable for short commands or when inline.	Preferred for long or complex commands.

# Generic Recommendation

Use array notation ([] ) for shorter commands.

Use multi-line YAML lists (-) for longer, more complex commands or when readability is a priority.

## 2 - Command and Arguments

The separation of command and args in Kubernetes is a design choice that provides flexibility and clarity when working with containerized applications.

However you can add everything in a single command as well.

```
! cmd-args.yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: busybox
      command: ["/bin/sh", "-c", "echo Hello Kubernetes!"]
```

# General Recommendation

`command` is intended to specify the entrypoint or the main executable that will run in the container.

`args` is meant to define the arguments or parameters that are passed to the command.

By keeping them separate, the intent of the configuration becomes clearer

## Points to Note

If a container image defines a default ENTRYPOINT, Kubernetes will use it unless you explicitly override it with command.

# **CLI Documentation for Kubernetes Resources**

# Kubernetes API Documentation

We generally refer to the Kubernetes API Documentation to understand various fields and associated descriptions for a specific resource.

Pod v1 core

[show example](#)

Group	Version	Kind
core	v1	Pod

**⚠ Warning:**

It is recommended that users create Pods only through a Controller, and not directly. See Controllers: [Deployment](#), [Job](#), or [StatefulSet](#).

**ⓘ Appears In:**

- [PodList \[core/v1\]](#)

Field	Description
<code>apiVersion</code> <code>string</code>	APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized values, and may reject unrecognized values. More info: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a>
<code>kind</code> <code>string</code>	Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint. Cannot be updated. In CamelCase. More info: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a>
<code>metadata</code> <code>ObjectMeta</code>	Standard object's metadata. More info: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata</a>

# CLI Documentation

You can also use `kubectl explain` to describe the fields associated with each supported API resource

```
C:\>kubectl explain pods
KIND:     Pod
VERSION:  v1

DESCRIPTION:
  Pod is a collection of containers that can run on a host. This resource is
  created by clients and scheduled onto hosts.

FIELDS:
  apiVersion  <string>
    APIVersion defines the versioned schema of this representation of an object.
    Servers should convert recognized schemas to the latest internal value, and
    may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions

  kind  <string>
    Kind is a string value representing the REST resource this object
    represents. Servers may infer this from the endpoint the client submits
    requests to. Cannot be updated. In CamelCase. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions

  metadata    <ObjectMeta>
    Standard object's metadata. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions

  spec  <PodSpec>
    Specification of the desired behavior of the pod. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions
```

# Similarity to API Doc Hyperlink to See Nested Fields

```
C:\>kubectl explain pods.metadata
KIND:     Pod
VERSION:   v1

FIELD: metadata <ObjectMeta>

DESCRIPTION:
  Standard object's metadata. More info:
  https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
  ObjectMeta is metadata that all persisted resources must have, which
  includes all objects users must create.

FIELDS:
  annotations    <map[string]string>
    Annotations is an unstructured key value map stored with a resource that may
    be set by external tools to store and retrieve arbitrary metadata. They are
    not queryable and should be preserved when modifying objects. More info:
    https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/

  creationTimestamp      <string>
    CreationTimestamp is a timestamp representing the server time when this
    object was created. It is not guaranteed to be set in happens-before order
    across separate operations. Clients may not set this value. It is
    represented in RFC3339 form and is in UTC.

    Populated by the system. Read-only. Null for lists. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#when-apiserver-updates-a-resource

  deletionGracePeriodSeconds    <integer>
```

---

# EXPOSE

Build once, use anywhere

---

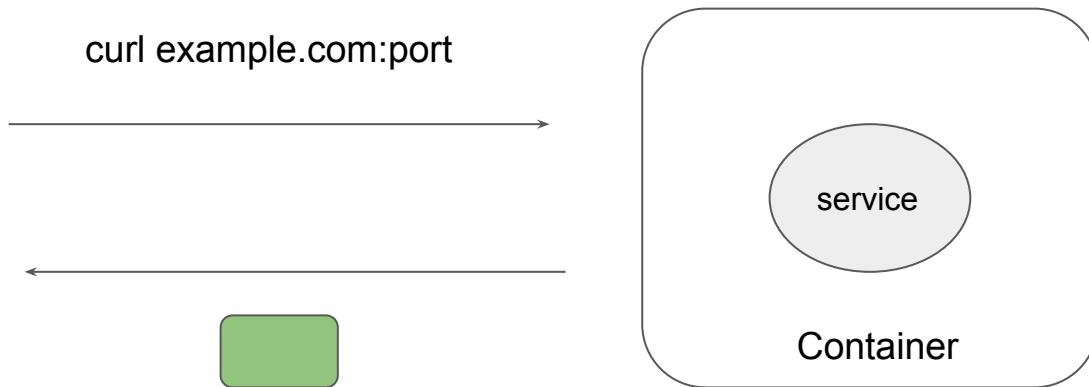
# Overview of EXPOSE instruction

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

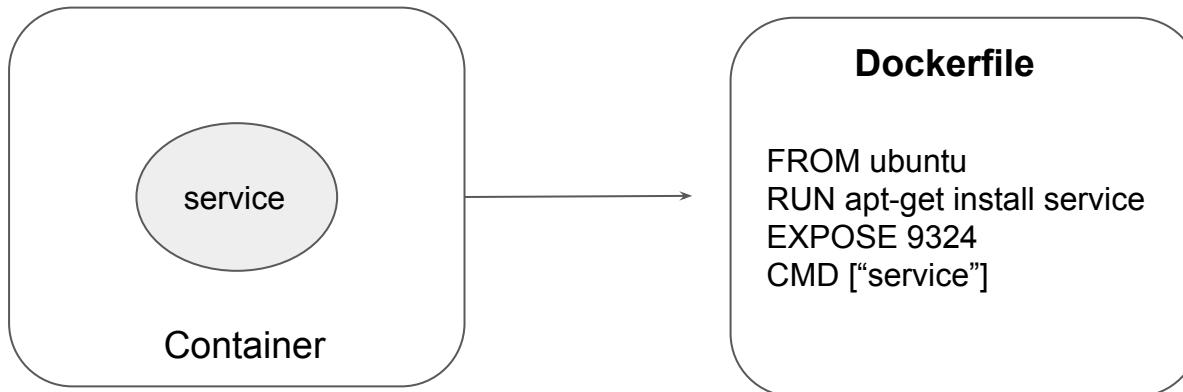
The EXPOSE instruction does not actually publish the port.

It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.

# Understanding the Use-Case



# Understanding the Use-Case



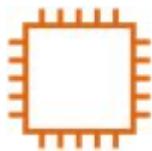
---

# Labels and Selectors

Let's get started

# Overview of Labels

Labels are key/value pairs that are attached to objects, such as pods.



Server



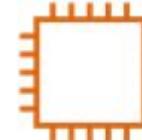
Database



Load Balancer



Load Balancer

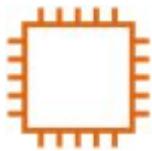


Server



Database

# Adding Labels to Resource



name: kplabs-gateway  
env: production



name: kplabs-db  
env: production



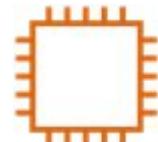
name: kplabs-elb  
env: production



name: kp-elb  
env: dev



name: kp-db  
env: dev



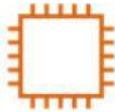
name: kp-gateway  
env: dev

# Overview of Selectors

Selectors allows us to filter objects based on labels.

Example:

1. Show me all the objects which has label where **env: prod**



name: kplabs-gateway  
env: production



name: kplabs-db  
env: production



name: kplabs-elb  
env: production

# Overview of Selectors

Selectors allows us to filter objects based on labels.

Example:

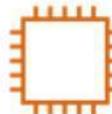
2. Show me all the objects which has label where **env: dev**



name: kp-elb  
env: dev



name: kp-db  
env: dev



name: kp-gateway  
env: dev

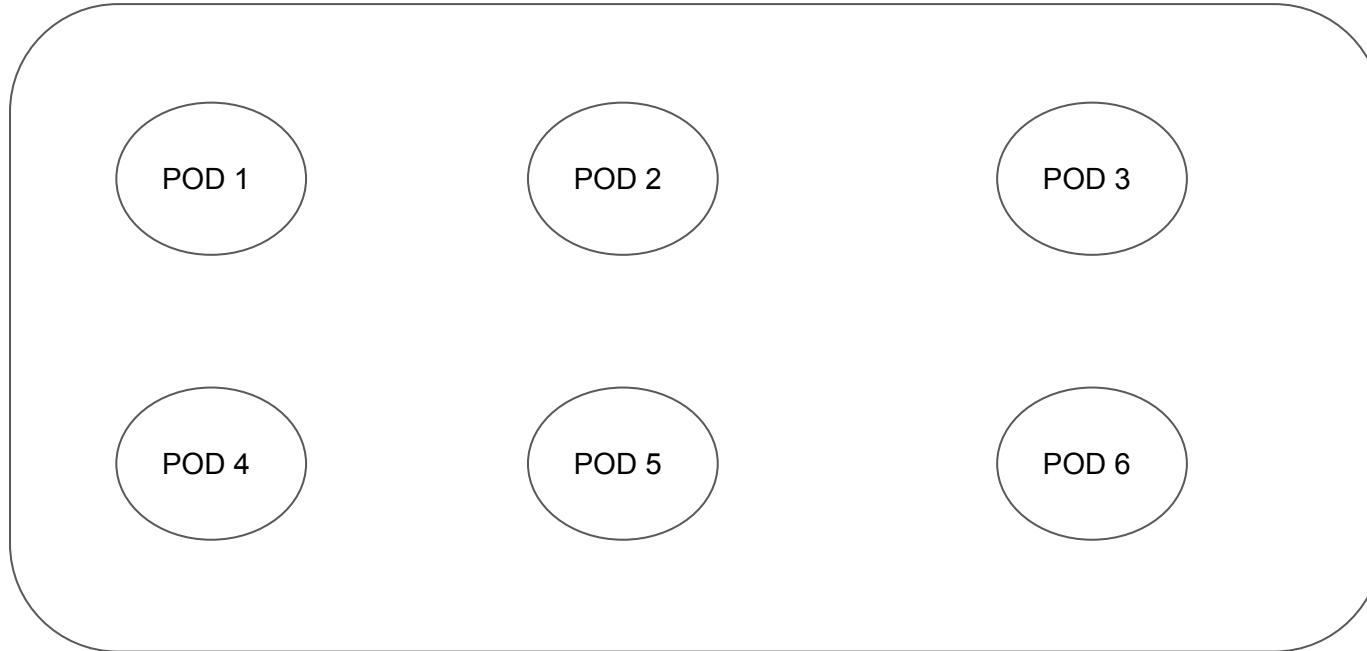
# Kubernetes Perspective

There can be multiple objects within a Kubernetes cluster.

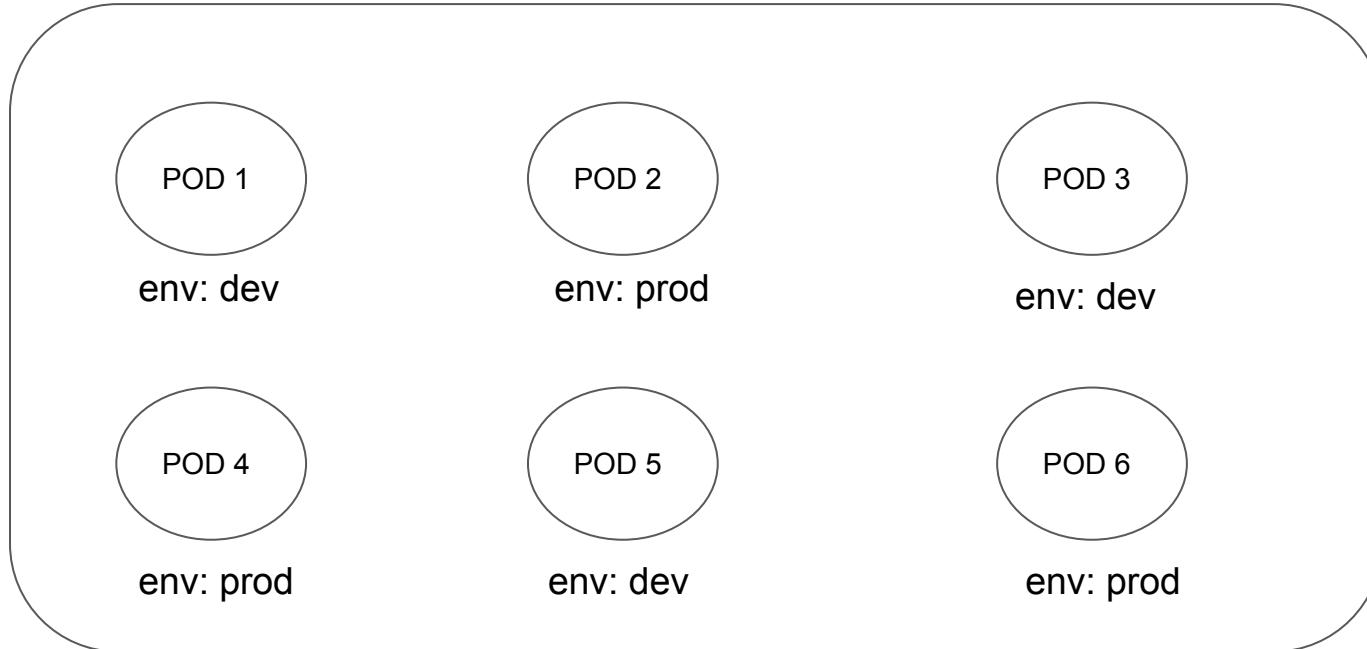
Some of the objects are as follows:

1. Pods
2. Services
3. Secrets
4. Namespaces
5. Deployments
6. Daemonsets

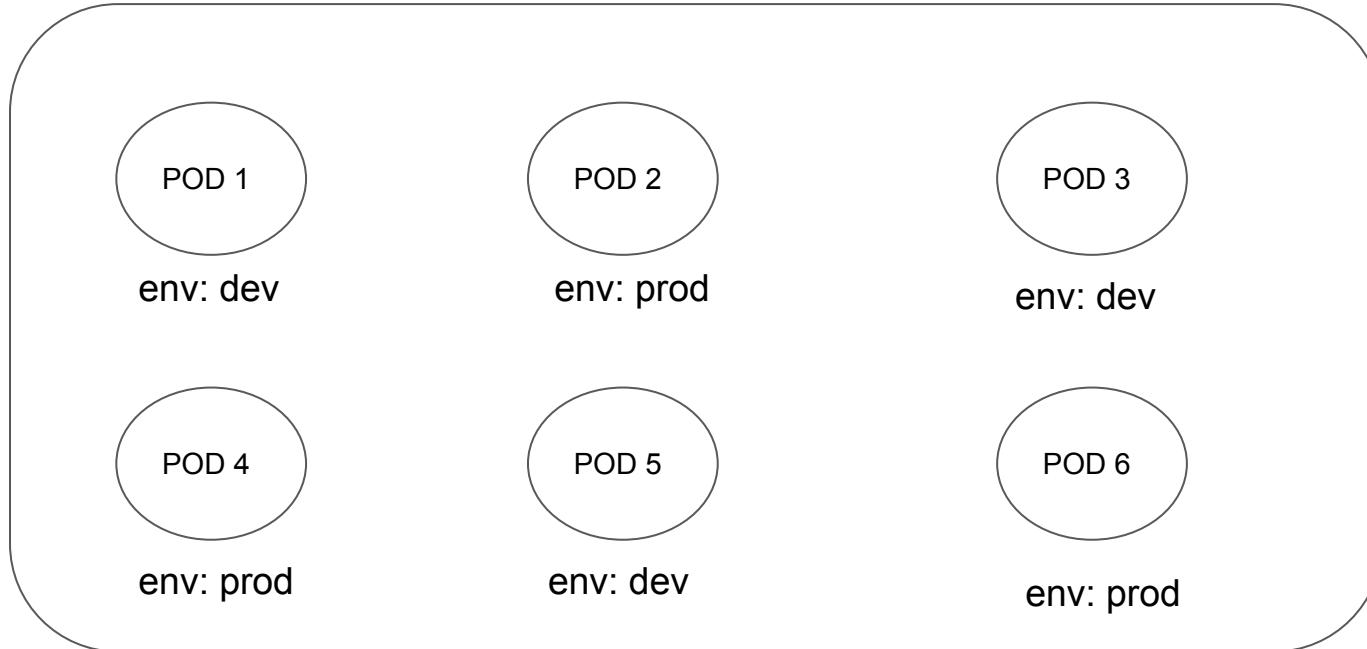
# Kubernetes Perspective



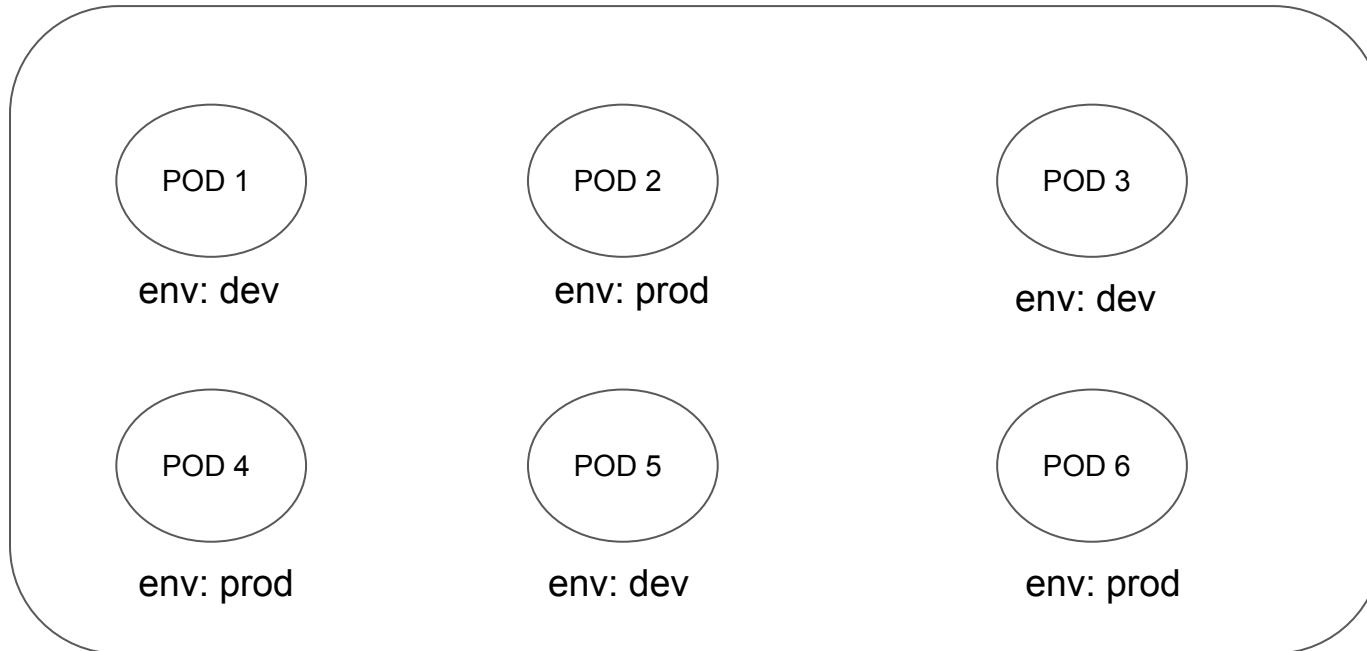
# Assigning Labels to Kubernetes Objects



# Selector: List All Pods from Dev Environment



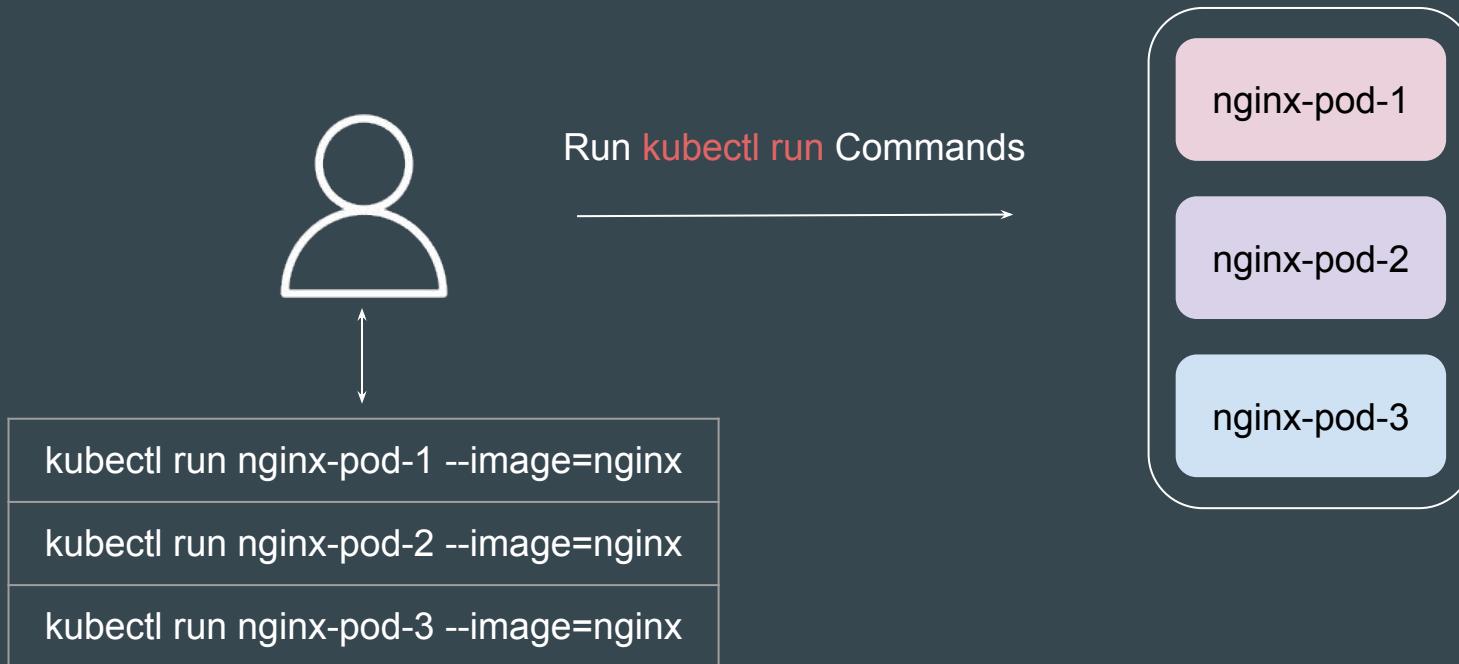
# Selector: List All Pods from Production Environment



# **ReplicaSet**

# Setting the Base

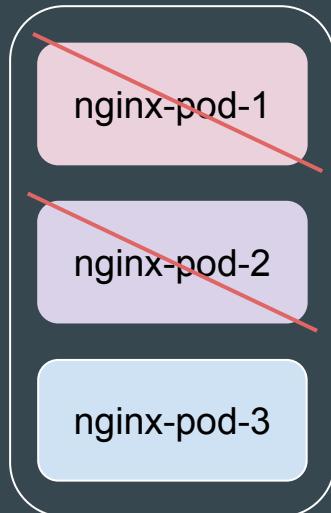
There is a requirement to run 3 Pods based on Nginx image all the time.



# Issue with the Manual Step - Part 1

If a Pod fails (node failure, process crash, etc.), Kubernetes will not automatically recreate it.

You would have to manually detect the failure and recreate the Pod.



## Issue with the Manual Step - Part 2

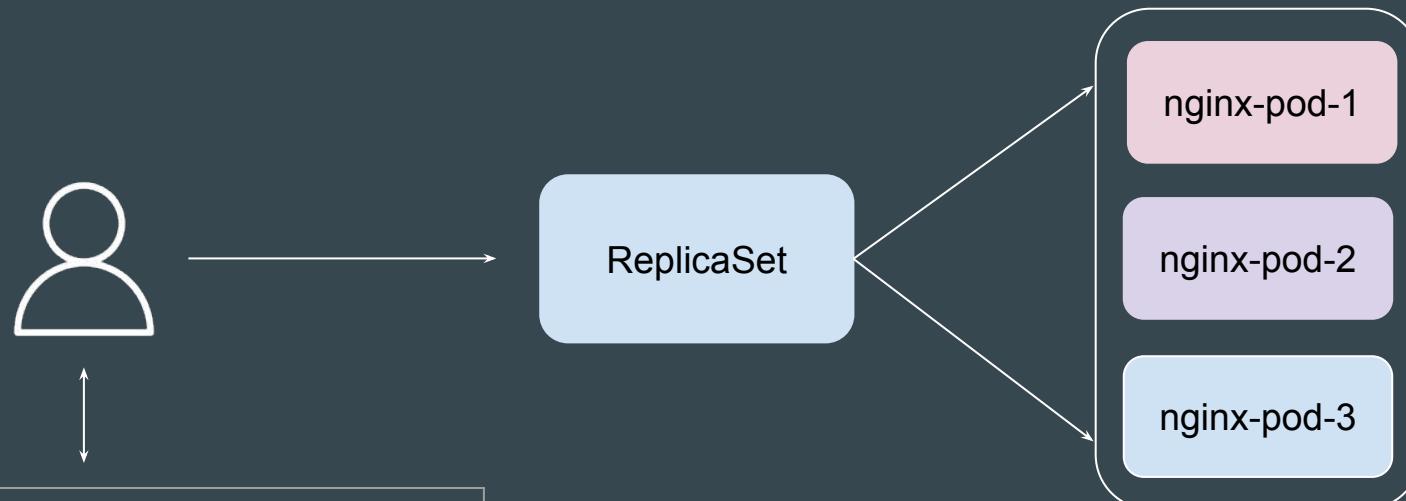
If you need to scale to more or fewer Pods, you would have to manually create or delete Pod definitions and apply the changes.

Example: Requirement is to run 20 Pods based on Nginx Image



# Introducing Kubernetes ReplicaSet

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.



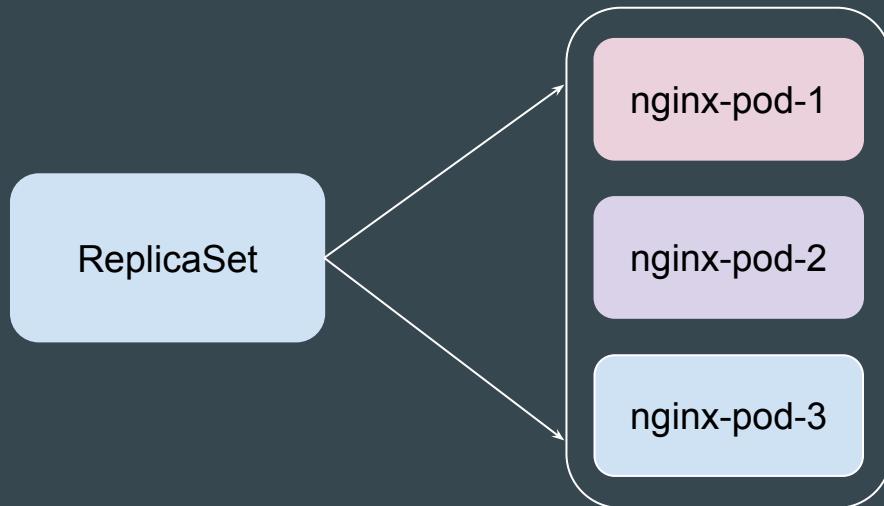
**Requirement**

I need 3 Pods of Nginx Run all the Time

# **Challenges with ReplicaSets**

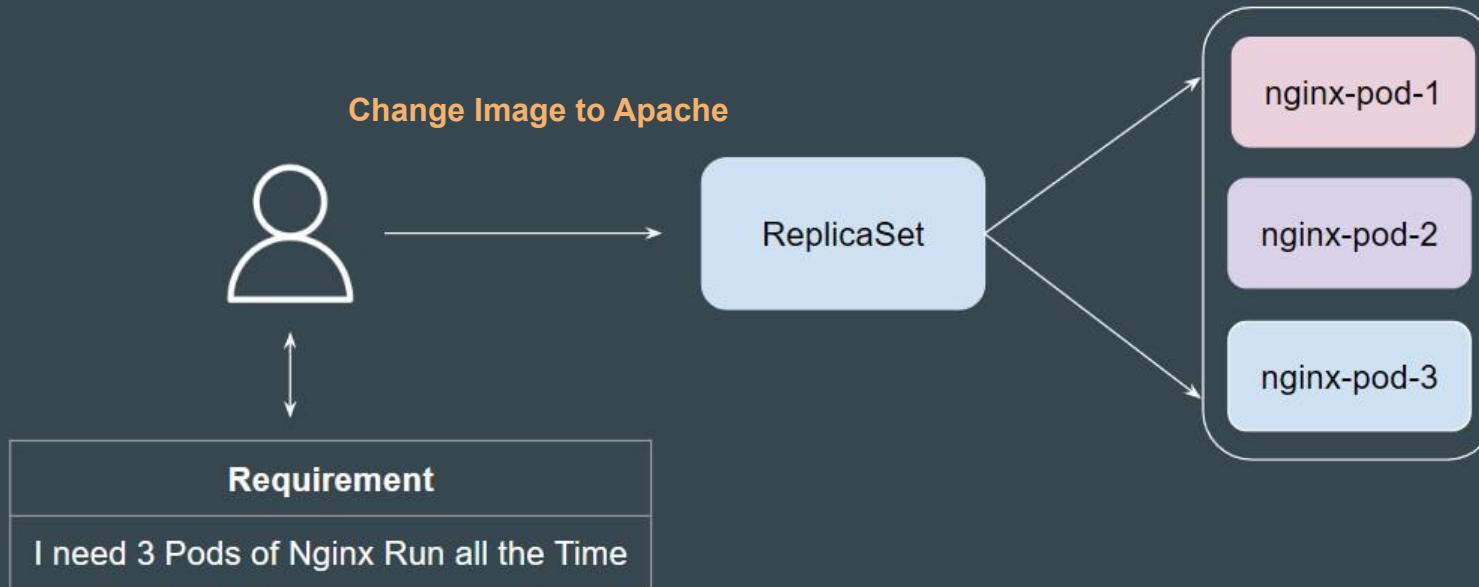
# Setting the Base

ReplicaSets are primarily designed to maintain a specific state of running Pods, not to manage regular updates or changes to their configuration



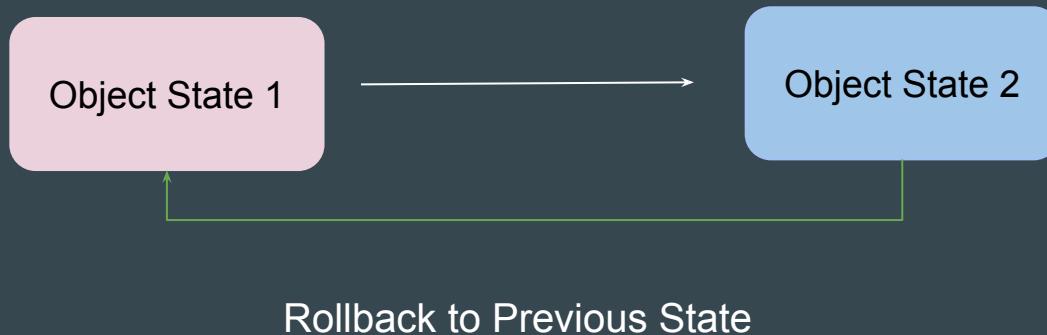
# Challenge 1 - Updating Container Image

When you update the pod template (e.g., **change the container image**) in a ReplicaSet, the existing pods are not updated.



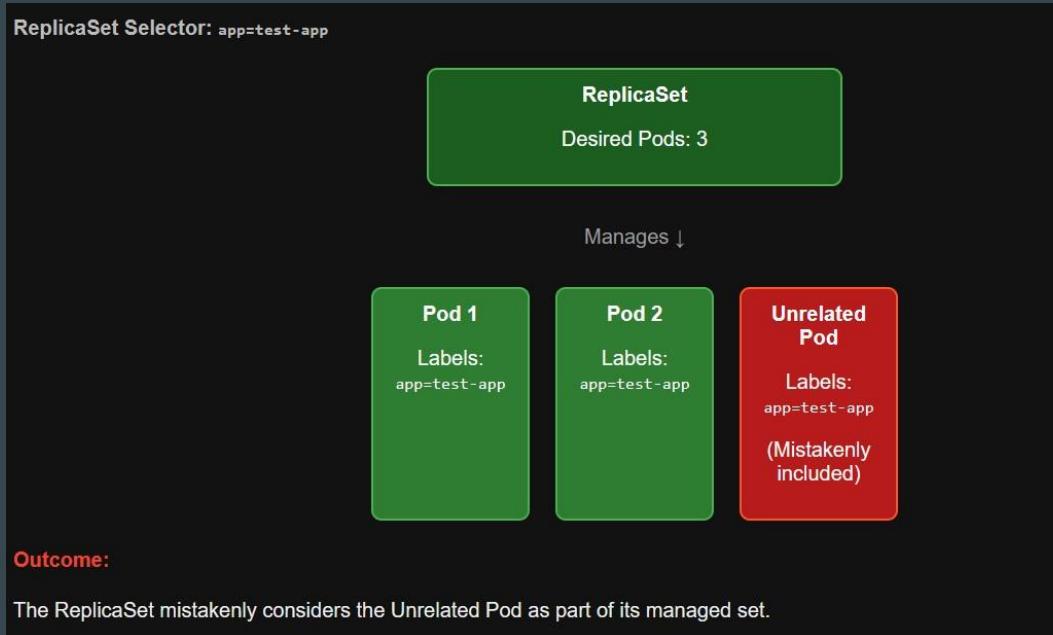
## Challenge 2 - No Built-In Rollback Mechanism

ReplicaSets **lack a rollback mechanism** for reverting to a previous configuration in case of errors during an update.



# Challenge 3 - Label Collision with ReplicaSet Selectors

When a ReplicaSet's selector matches labels of pods that it didn't create, it starts treating those pods as part of its managed set. This can cause unintended consequences.



# **Overview of Deployments**

# Simple Analogy - Camera and Lens

Think of the camera body as the ReplicaSet.

It is the core mechanism that controls and ensures that the right number of photos (or Pods) are being captured



## Simple Analogy - Camera and Lens

Now, imagine attaching a big, powerful lens to the camera body. The lens is the Deployment.

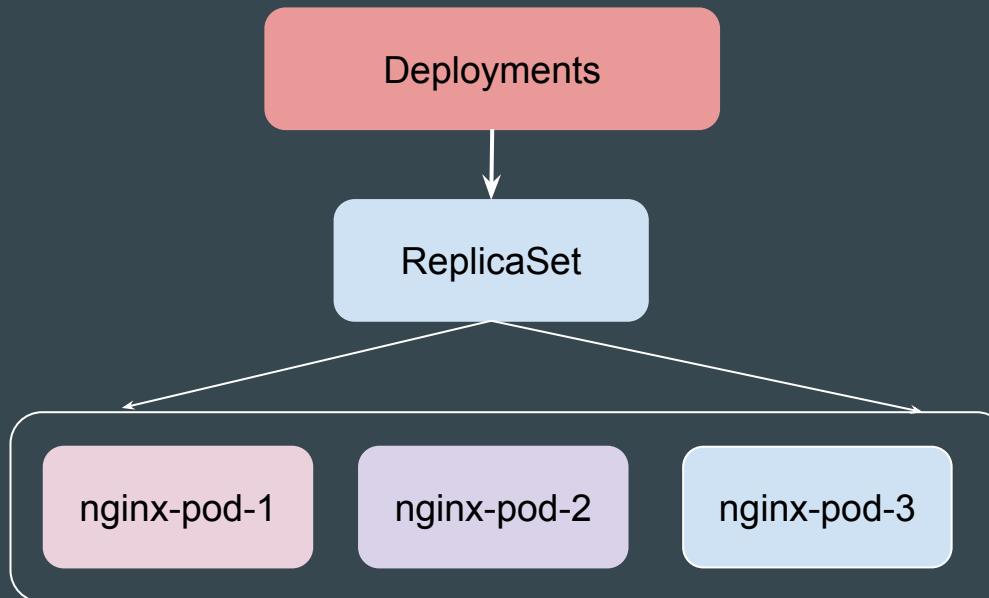
It doesn't just sit on top of the camera; it enhances and extends the camera's functionality, allowing for new perspectives.



# Setting the Base

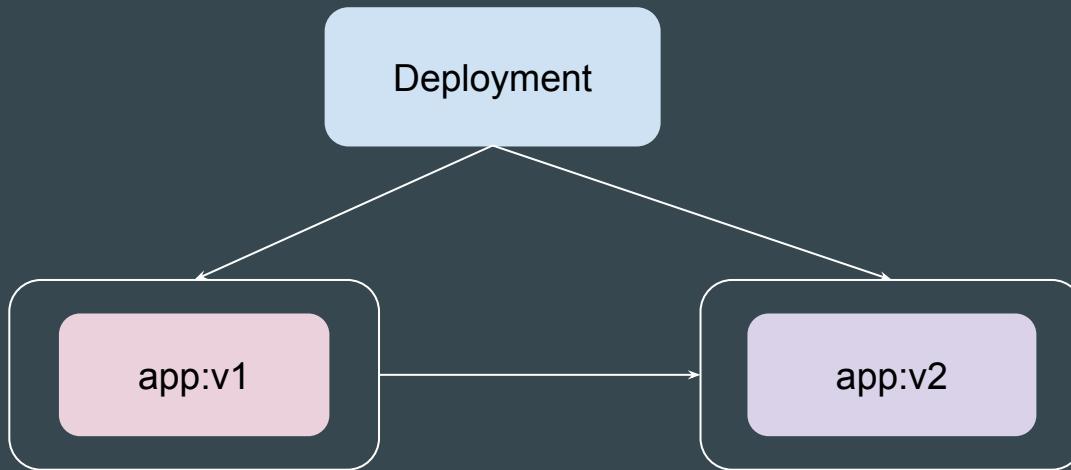
A Deployment is a higher-level abstraction built on top of ReplicaSets.

It not only manages ReplicaSets but also **provides advanced features** like rolling updates, rollbacks, and versioning.



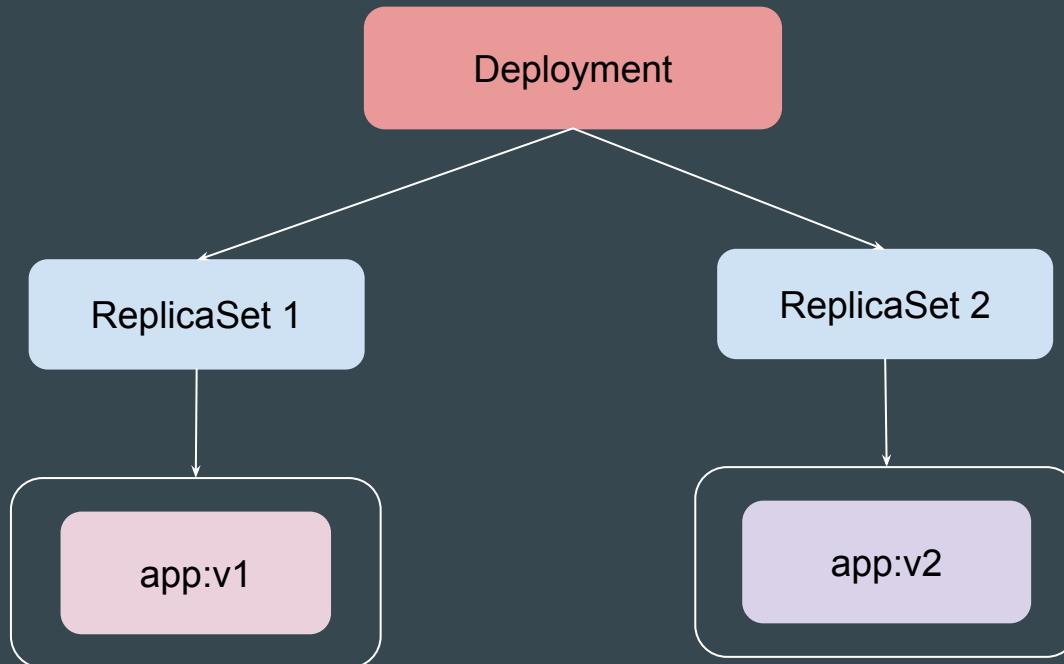
# Use-Case: Update the Container Image

You can easily change the specifications like Container Image and other spec.



# Use-Case: Rolling Update

Deployments will perform update in rollout manner to ensure that your app is not down.



# Rollout History

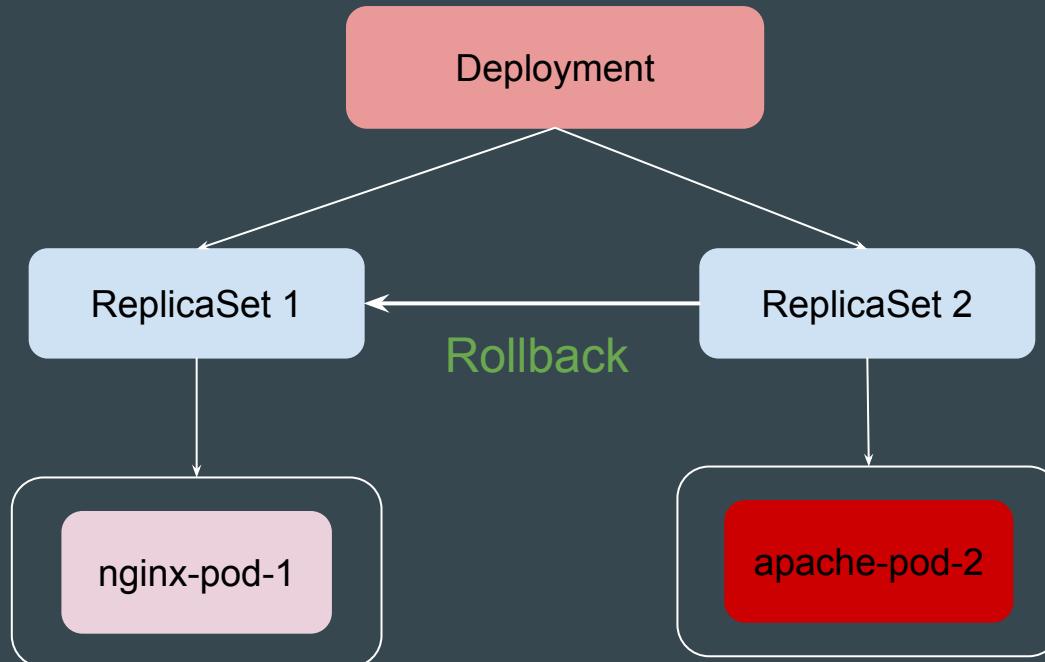
Deployment allows us to inspect the history of your deployments.

It provides a record of changes made to a deployment over time, enabling you to understand the evolution of your application and troubleshoot potential issues.

```
C:\>kubectl rollout history deployment/nginx-deployment  
deployment.apps/nginx-deployment  
REVISION  CHANGE-CAUSE  
1          <none>  
2          <none>
```

# Rolling Back Changes

Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping



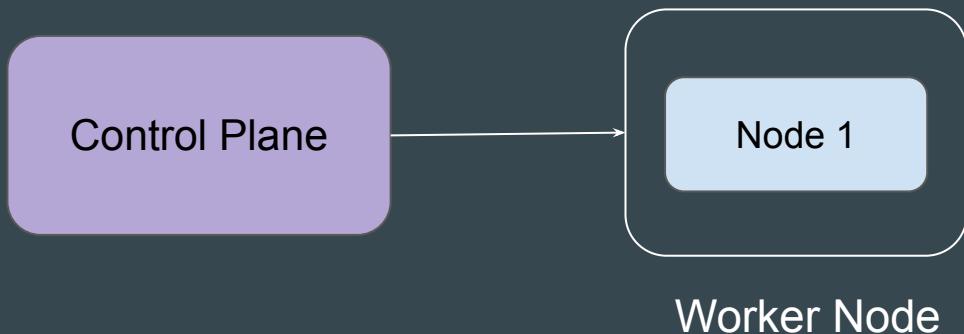
# Key Differences - ReplicaSet and Deployments

Feature	ReplicaSet	Deployment
Abstraction Level	Lower-level	Higher-level
Primary Function	Ensures a specific number of replicas are running.	Manages ReplicaSets and handles updates.
Rolling Updates	Not supported	Supported with configurable strategies.
Rollbacks	Not supported	Supported with version history.
Use Case	Simple, static workloads.	Dynamic workloads with frequent updates.

# **Multiple Worker Nodes for Kubernetes**

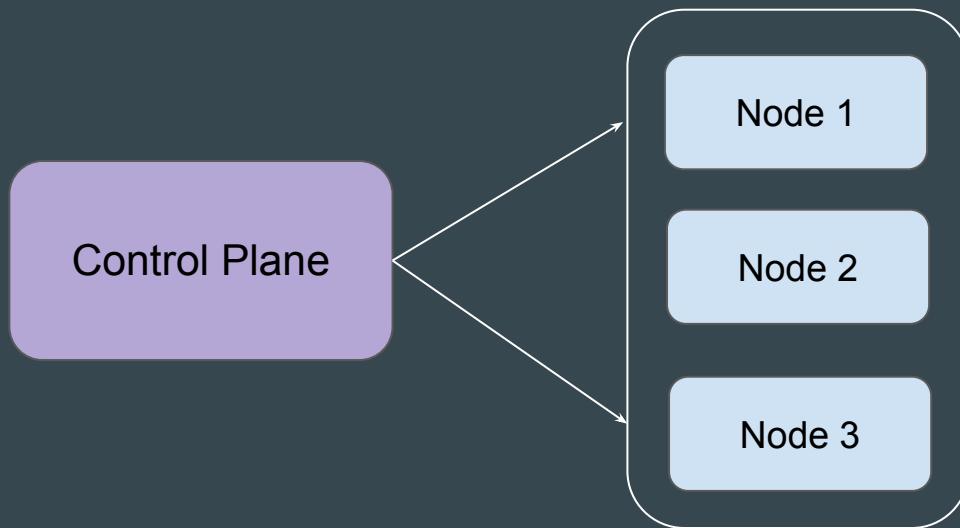
# Understanding the Challenge

A single worker node might not have enough resources (CPU, memory) to handle the load of all the Pods.



# Multiple Worker Nodes are Good

Adding more worker nodes allows you to distribute the workload across the worker nodes.



## Other Benefits of Multiple Worker Nodes

If one worker node fails, the applications running on it will become unavailable.

With multiple worker nodes, Kubernetes can reschedule those applications onto healthy nodes, ensuring continuous service availability.

# Point to Note - Worker Node Sizes

It is not necessary for ALL worker nodes to have same hardware specification.

Sample Example:

Worker Node	Hardware Specification
Worker Node 1	2GB of RAM and 2 core CPU
Worker Node 2	4GB of RAM and 4 core CPU
Worker Node 3	8GB of RAM and 8 core CPU

# How to Check Worker Nodes in K8s Cluster

Use the `kubectl get nodes` command to get list of worker node as part of your cluster.

```
C:\>kubectl get nodes
NAME           STATUS    ROLES      AGE     VERSION
kplabs-k8s-eop82   Ready    <none>    16d    v1.31.1
```

```
C:\>kubectl get nodes
NAME           STATUS    ROLES      AGE     VERSION
kplabs-k8s-eop82   Ready    <none>    17d    v1.31.1
kplabs-k8s-est1m   Ready    <none>    3m13s   v1.31.1
kplabs-k8s-est1q   Ready    <none>    2m33s   v1.31.1
```

## Point to Note

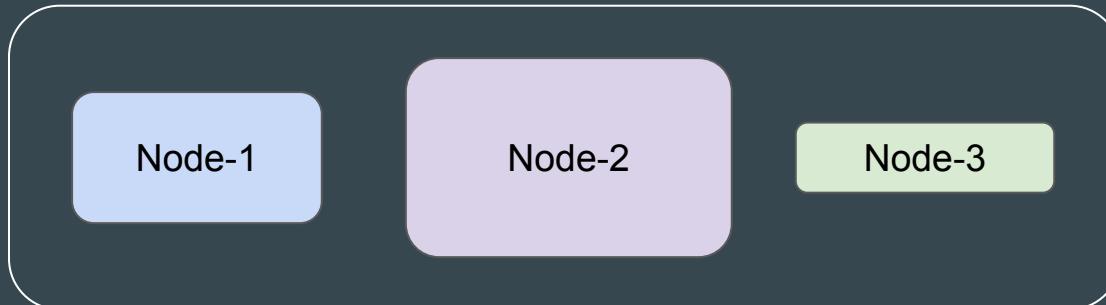
Depending on the type of environment that was used to create the Kubernetes cluster, the steps to add multiple worker nodes will change accordingly.

# **Node Selector**

# Setting the Base

In Kubernetes, different worker nodes can be based on different hardware specifications.

Some applications would require more CPU and Memory, while some applications might require minimum compute resource.



# Understanding the Topic

Node Selector is a mechanism used to **control the placement of pods** onto specific nodes within a cluster.

**Requirement:** AppA Pod requires high memory and CPU. Run it in Big Node.



# Why Use Node Selectors

1. To ensure certain workloads run on specific nodes with special hardware or configurations (e.g., GPUs, high-memory nodes).
2. To segregate workloads for compliance, performance, or isolation purposes.

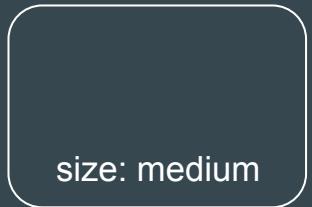
# **Node Selector - Practical**

# How it Works

1. Nodes Are Labeled: Assign appropriate labels to worker node.
  2. Pod Specification: In the pod's specification file, you can define a selector (nodeSelector) to match the desired labels.
- The Kubernetes scheduler places the pod on a node that matches the specified label(s).

# Step 1 - Adding Label to Nodes

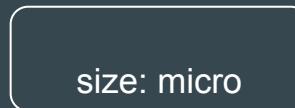
Add appropriate labels to your nodes depending on their hardware type.



Node 1



Node 2



Node 3

## Step 2 - Use NodeSelector Configuration

Create a `nodeSelector` configuration to run pods only on nodes which has a label of `size=Large`

```
! nodeSelector.yaml
apiVersion: v1
kind: Pod
metadata:
  name: memory-intensive-app
spec:
  containers:
  - name: app
    image: nginx
  nodeSelector:
    size: Large
```

# **DaemonSet**

# Setting the Base

A DaemonSet can ensure that all Nodes run a copy of a Pod.

Whenever new nodes are added to the cluster, Pods will be created in them.



Worker Node 1



Worker Node 2



Worker Node 3

# Common Type of Agents for Daemonset

1. Anti-Virus and Malware Scanning Agents
2. Log Collection Agents to collect logs from nodes.
3. Monitoring and Metrics Collection Agents to collect metrics about node

# Rollout History

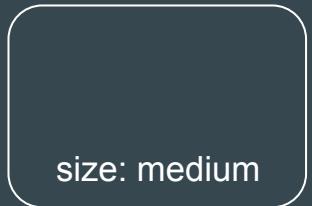
Deployment allows us to inspect the history of your deployments.

It provides a record of changes made to a deployment over time, enabling you to understand the evolution of your application and troubleshoot potential issues.

```
C:\>kubectl rollout history deployment/nginx-deployment  
deployment.apps/nginx-deployment  
REVISION  CHANGE-CAUSE  
1          <none>  
2          <none>
```

# Step 1 - Adding Label to Nodes

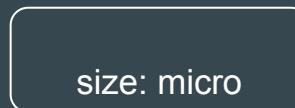
Add appropriate labels to your nodes depending on their hardware type.



Node 1



Node 2



Node 3

## Step 2 - Use NodeSelector Configuration

Create a `nodeSelector` configuration to run pods only on nodes which has a label of `size=Large`

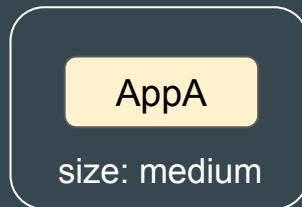
```
! nodeSelector.yaml
apiVersion: v1
kind: Pod
metadata:
  name: memory-intensive-app
spec:
  containers:
  - name: app
    image: nginx
  nodeSelector:
    size: Large
```

# **Node Affinity**

# Setting the Base

Node Selector primarily makes use of a method of **strict equality** where your selector must match the label in a worker node.

**Example:** Run AppA Pod in Node that has Label of size=medium

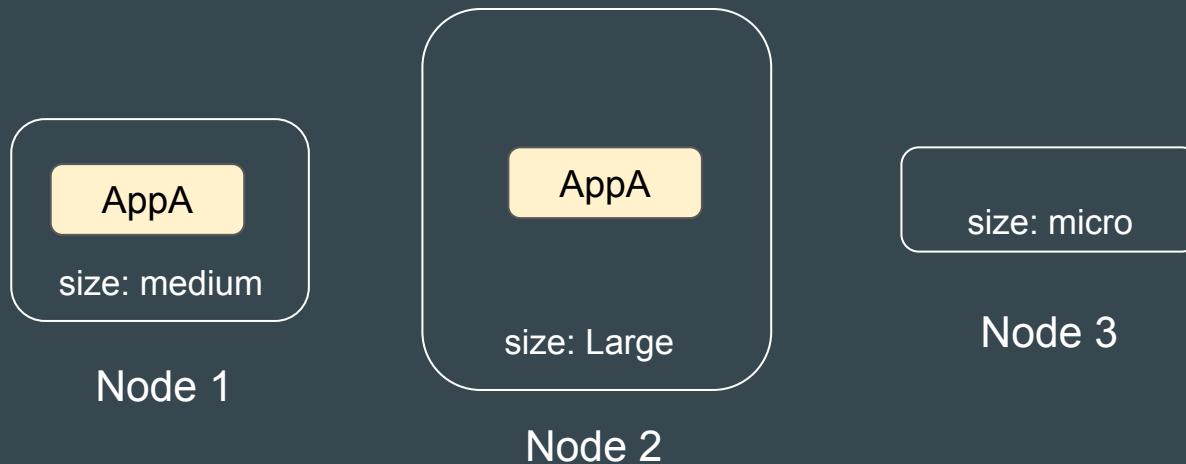


Node 1

# Introduction to the Topic

Node Affinity is similar to the older Node Selector, but it is **more flexible and expressive**.

**Requirement:** Run AppA pod in any sized node except micro.



# Operators in Node Affinity

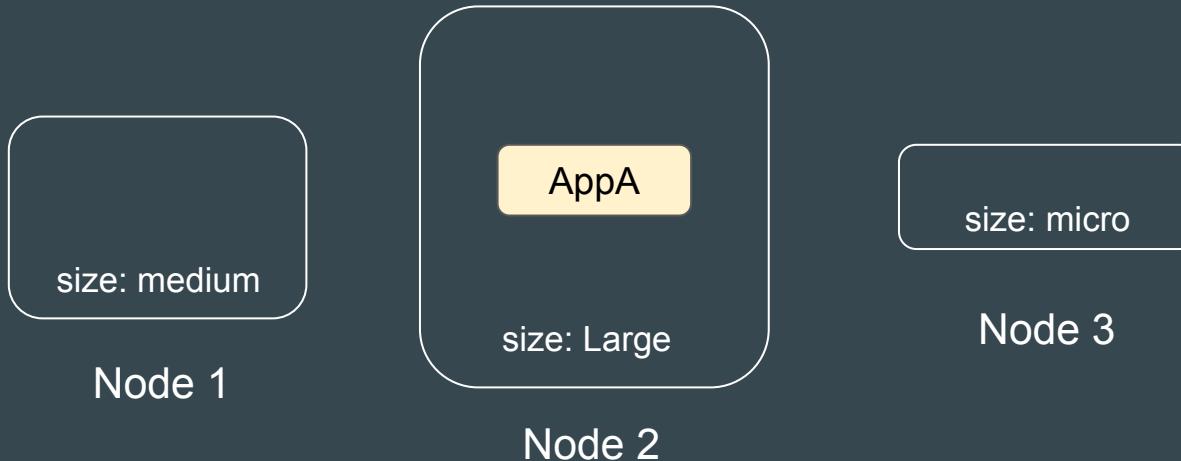
Operator	Description
In	Matches nodes with the specified label values.
NotIn	Excludes nodes with the specified label values.
Exists	Matches nodes that have the specified key, regardless of its value.
DoesNotExist	Excludes nodes that have the specified key.
Gt and Lt	Matches nodes with numeric label values greater than or less than a specified value.

# Hard vs Soft Preferences

There are two approaches that you can use as part of the Affinity.

1. Required (Hard Requirement)
2. Preferred (Soft Requirement)

**Requirement:** Run AppA pod in Extra Large Node (Preferred).



# Hard vs Soft Preferences

When defining Node Affinity, you can specify either hard constraints or soft preferences for node selection. These are represented by the two modes:

Modes	Description
requiredDuringSchedulingIgnoredDuringExecution	<p>Pods must be scheduled on nodes that match the criteria.</p> <p>If no nodes match the rules, the pod won't be scheduled.</p> <p>This is a strict requirement.</p>
preferredDuringSchedulingIgnoredDuringExecution	<p>Pods prefer to be scheduled on nodes that match the criteria.</p> <p>If no nodes match, the pod will still be scheduled on other nodes.</p> <p>This is a recommendation, not a strict rule.</p>

# Reference Code - Required

```
! nodeAffinity-required.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: disktype
            operator: In
            values:
            - ssd
  containers:
  - name: nginx
    image: nginx
```

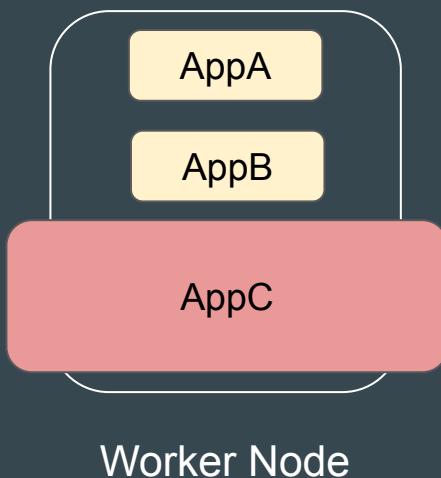
# Reference Code - Preferred

```
! nodeAffinity-preferred.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
  containers:
    - name: nginx
      image: nginx
```

# **Requests and Limits**

# Understanding the Challenge - Part 1

A specific pod may consume more resources than expected, affecting all pods in the node.



# Understanding the Challenge - Part 2

A Pod requires a certain amount of memory and computing resources to run properly.

Example: AppC pod requires 512 MB of RAM and 2 core CPU to run optimally.



# Introduction to the Topic

Requests and Limits are two ways in which we can control the amount of resource that can be assigned to a pod (resource like CPU and Memory)



# Difference Between Request and Limit

Requests	Limits
<p>The minimum amount of CPU or memory a container is guaranteed to get.</p> <p>Kubernetes uses this to schedule the pod on a node that has enough resources.</p>	<p>The maximum amount of CPU or memory a container is allowed to use.</p> <p>If a container exceeds this limit, it may be throttled (CPU) or killed (memory).</p>

# Advantages of Request and Limit

Advantages	Description
Prevents resource starvation	Ensures no single container consumes all resources on a node.
Ensures fair resource allocation	Helps share resources across containers in a cluster.
Avoids over-provisioning	Prevents wasting resources by capping usage.
Stability and performance	Improves stability and performance of your applications.

# Reference Code

! request-limit.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: kplabs-pod
spec:
  containers:
    - name: kplabs-container
      image: nginx
      resources:
        requests:
          memory: "128Mi"
          cpu: "0.5"
        limits:
          memory: "500Mi"
          cpu: "1"
```

## Points to Note - Memory

128 mebibytes is the amount of memory requested by the container.

When the container is scheduled onto a node, Kubernetes ensures that the node has at least this much memory available.

500Mi: This is the memory limit, meaning the container is not allowed to consume more than 500Mi of memory.

If the container exceeds this memory usage, it might be terminated or throttled

# Reference Code

! request-limit.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: kplabs-pod
spec:
  containers:
    - name: kplabs-container
      image: nginx
      resources:
        requests:
          memory: "128Mi"
          cpu: "0.5"
        limits:
          memory: "500Mi"
          cpu: "1"
```

## Points to Note - CPU

0.5: This refers to half a CPU core (or 50% of a single CPU core).

This is the CPU requested by the container. Kubernetes ensures that the container gets at least this much CPU time on a node.

1: This is the CPU limit, meaning the container can use up to 1 full CPU core. If the container tries to use more than 1 CPU core, it will be throttled.

# Explanation for Reference Screenshot

The container is guaranteed a minimum of 128Mi of memory (request) but cannot exceed 500Mi of memory (limit).

The container is guaranteed a minimum of 0.5 CPU cores (request) but cannot exceed 1 CPU core (limit).

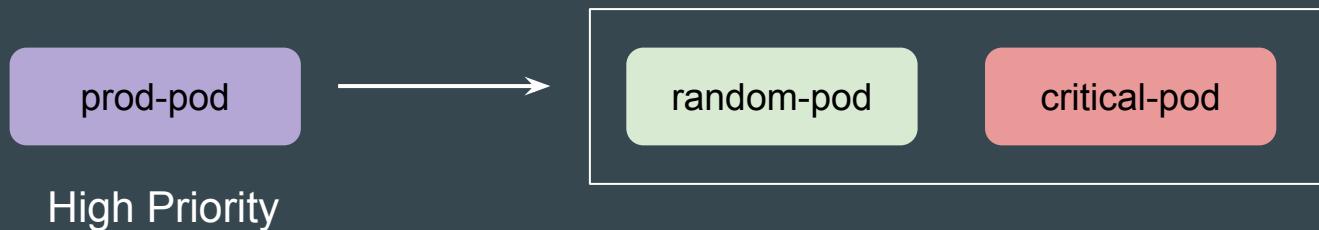
# **PriorityClass**

# Setting the Base

Pods can have **priority**.

Priority indicates the importance of a Pod relative to other Pods.

If a Pod cannot be scheduled, the scheduler tries to preempt (evict) lower priority Pods to make scheduling of the pending Pod possible.



# Advantages of Priority Class

PriorityClass influences two key things:

Influence	Description
Scheduling Order	<p>Pods with higher priority are placed ahead in the scheduling queue. The scheduler tries to schedule higher-priority Pods before lower-priority Pods.</p>
Preemption	<p>If a high-priority Pod cannot be scheduled because of insufficient resources, the scheduler can evict (preempt) lower-priority Pods from a node to make room for the high-priority Pod.</p>

# Creating PriorityClass Object

A PriorityClass object can have any 32-bit integer value smaller than or equal to 1 billion.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

# Setting Priority Class for a Pod

Create a Pod with appropriate priorityClass

```
apiVersion: v1
kind: Pod
metadata:
  name: priority-pod
spec:
  containers:
  - name: nginx
    image: nginx
  priorityClassName: high-priority
```

# Workflow

Worker node has capacity to run only 3 Pods.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
low-priority-deploy	3/3	3	3	2m28s
high-priority-deploy	0/3	0	0	0s
high-priority-deploy	0/3	0	0	0s
high-priority-deploy	0/3	0	0	0s
high-priority-deploy	0/3	3	0	0s
low-priority-deploy	2/3	2	2	2m31s
low-priority-deploy	2/3	3	2	2m31s
low-priority-deploy	1/3	2	1	2m31s
low-priority-deploy	1/3	3	1	2m31s
low-priority-deploy	0/3	2	0	2m31s
low-priority-deploy	0/3	3	0	2m31s
high-priority-deploy	1/3	3	1	5s
high-priority-deploy	2/3	3	2	5s
high-priority-deploy	3/3	3	3	5s

---

# Multi-Container POD Patterns

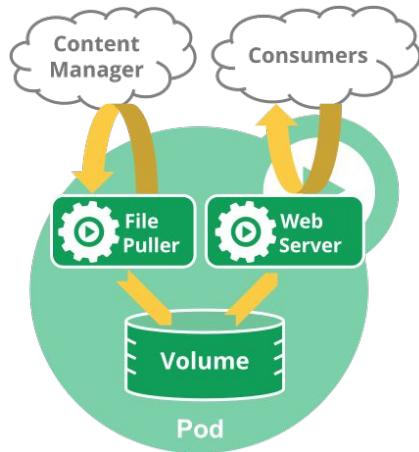
Multi-Container Patterns

---

# Overview of Sidecar Pattern

Sidecar pattern is nothing but running multiple container as part of a pod in a single node.

In below diagram we see that there is a web-server container which servers files from volumes and a separate sidecar container “file puller” which fetches and updates those files.

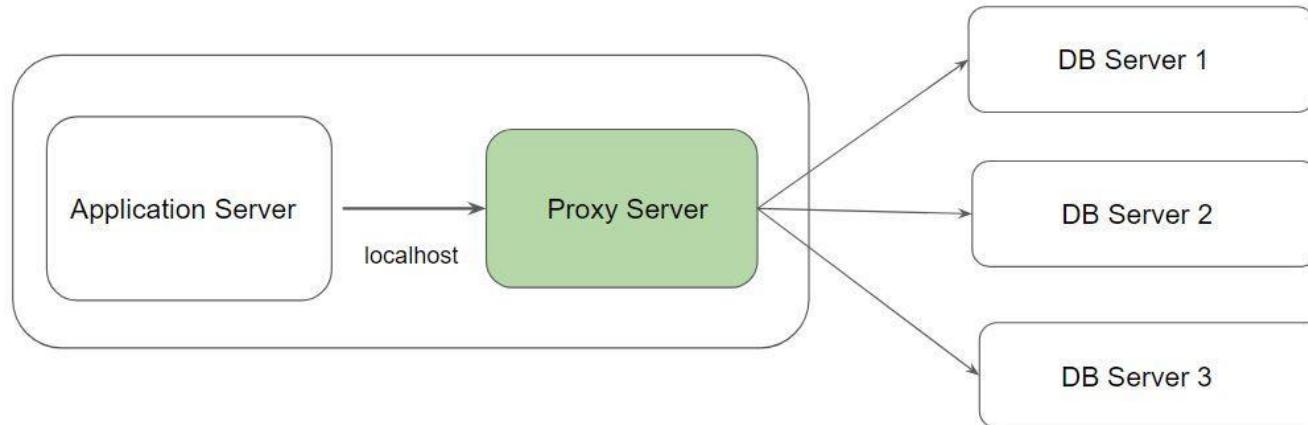


# Overview of Sidecar Pattern



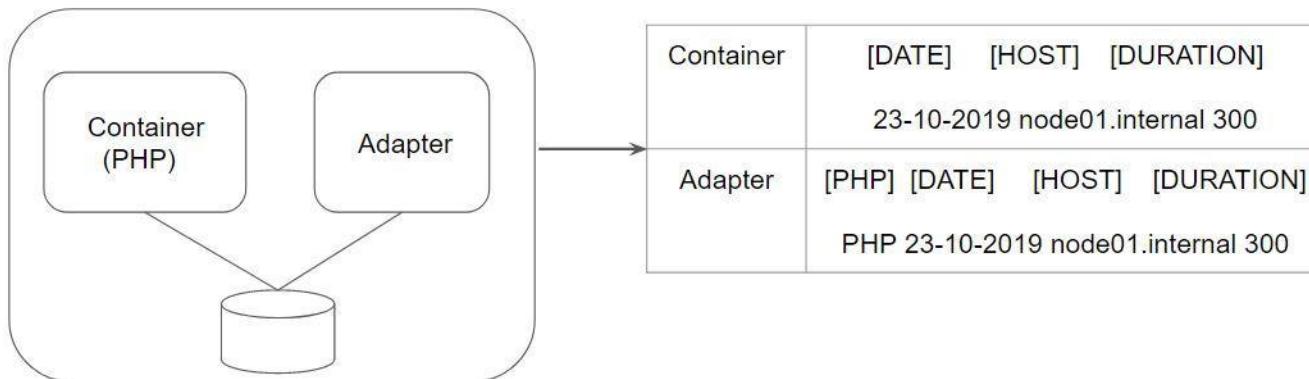
# Ambassador Pattern

Ambassador Pattern is a type of sidecar pattern where the second container is primarily used to proxy the requests.



# Sidecar Container - Adapter Pattern

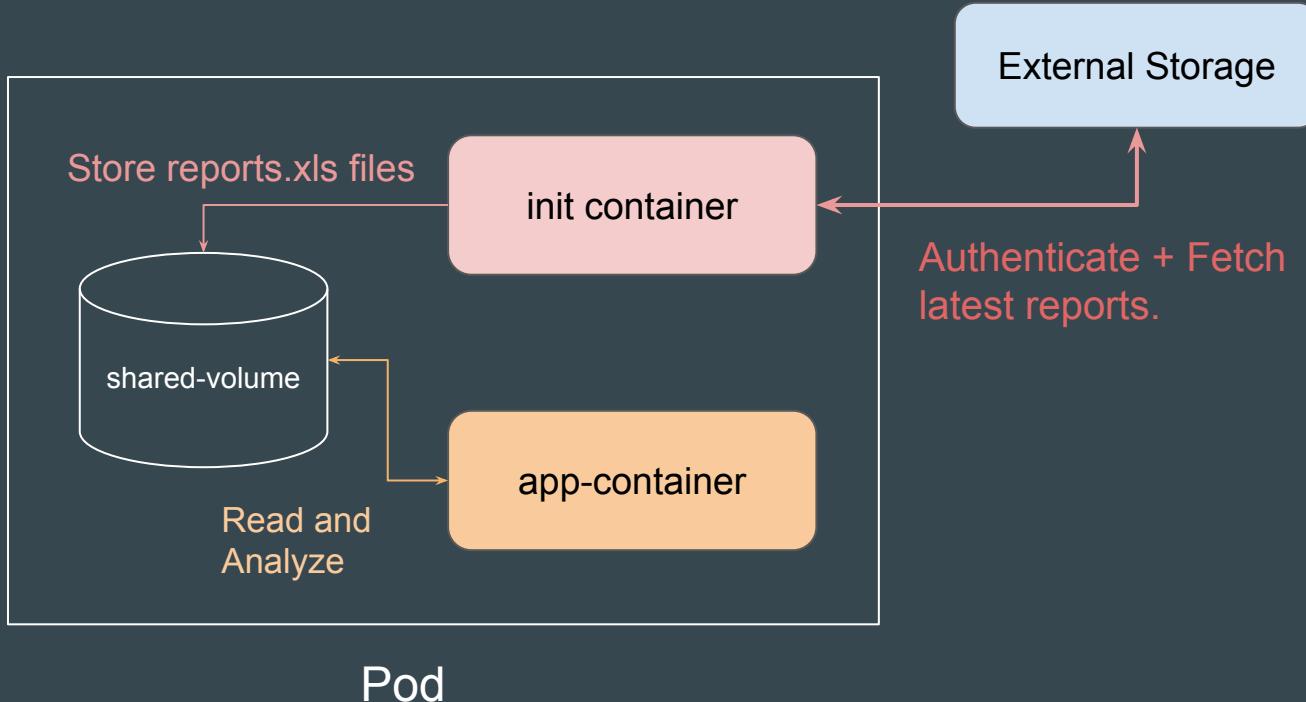
Adapter Pattern is generally used to transform the application output to standardize / normalize it for aggregation.

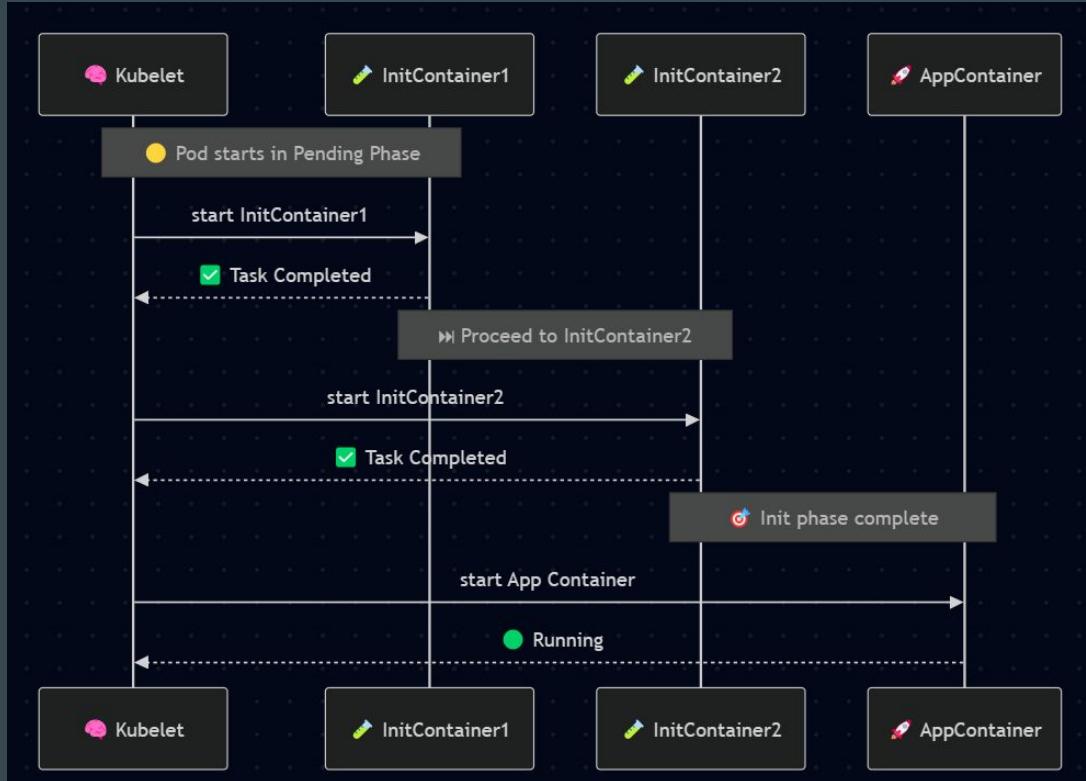


# **Init Containers**

# Setting the Base

Init Containers are specialized containers that run before app containers in a Pod.





## Point to Note

Init containers are exactly like regular containers, except:

1. Init containers always run to completion.
2. Each init container must complete successfully before the next one starts.

If a Pod's init container fails, the kubelet repeatedly restarts that init container until it succeeds (default restart policy of Always)

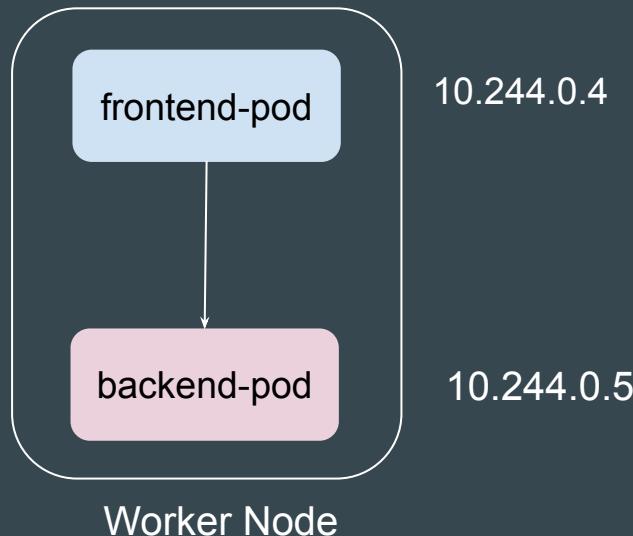
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
  - name: app-container
    image: nginx
  initContainers:
  - name: check-google
    image: alpine/curl
    command: ["ping", "-c", "10", "google.com"]
  - name: check-kubernetes
    image: alpine/curl
    command: ["curl", "kubernetes.io"]
```

# **Overview of Service**

# Setting the Base

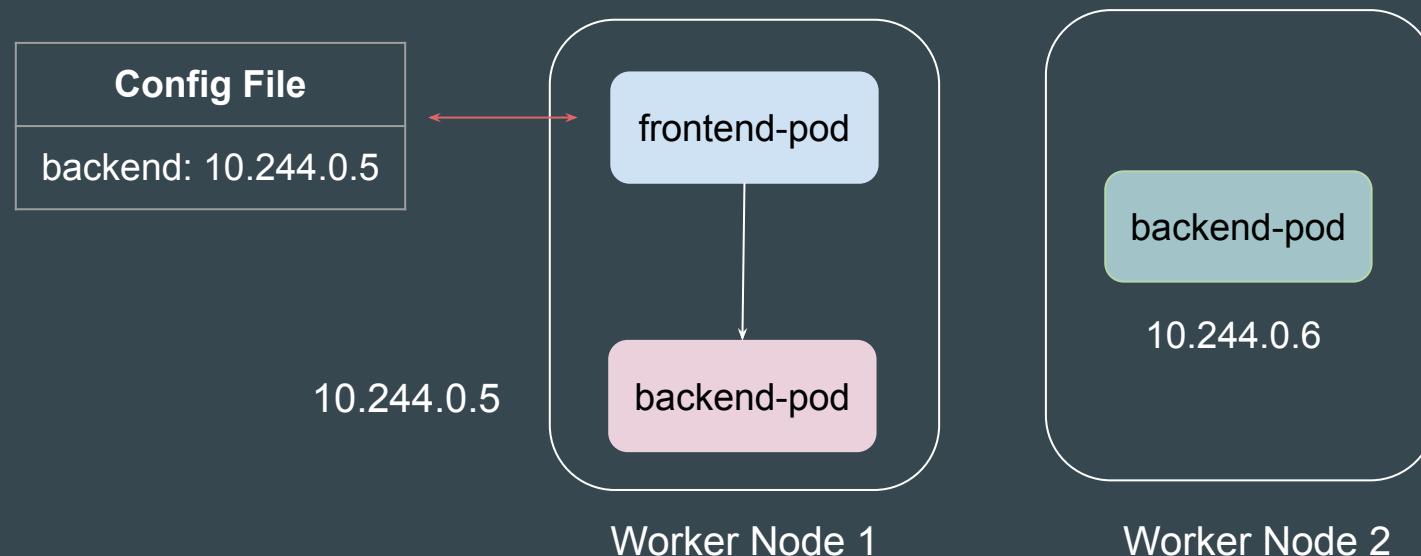
The Pods that get created in the Worker node have a private IP associated with them.

Pods in the same cluster can communicate with each other using Private IPs.



# Use-Case: Frontend to Backend

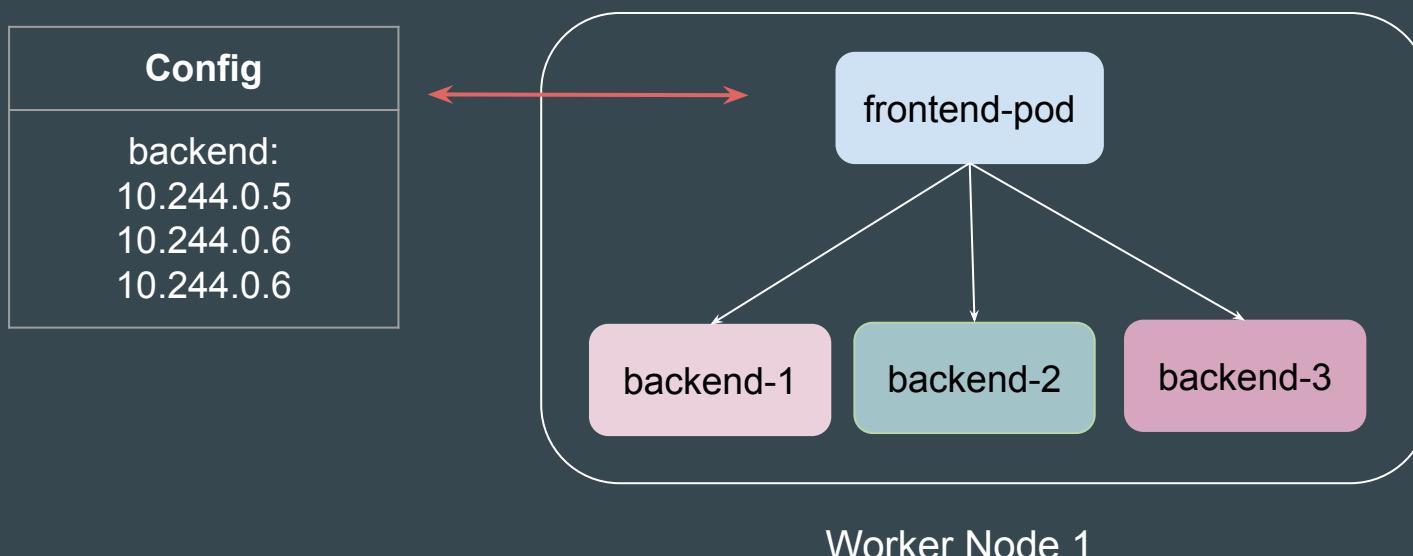
If the IP address of the backend pod is **hardcoded** in the front-end pod, it will create issues since Pods can be ephemeral.



# Use-Case: Frontend to Backend

If backend pods are running as deployment, it is challenging to hardcode the IPs of each backend pod.

You would also like to distribute the traffic across all the backend pods.

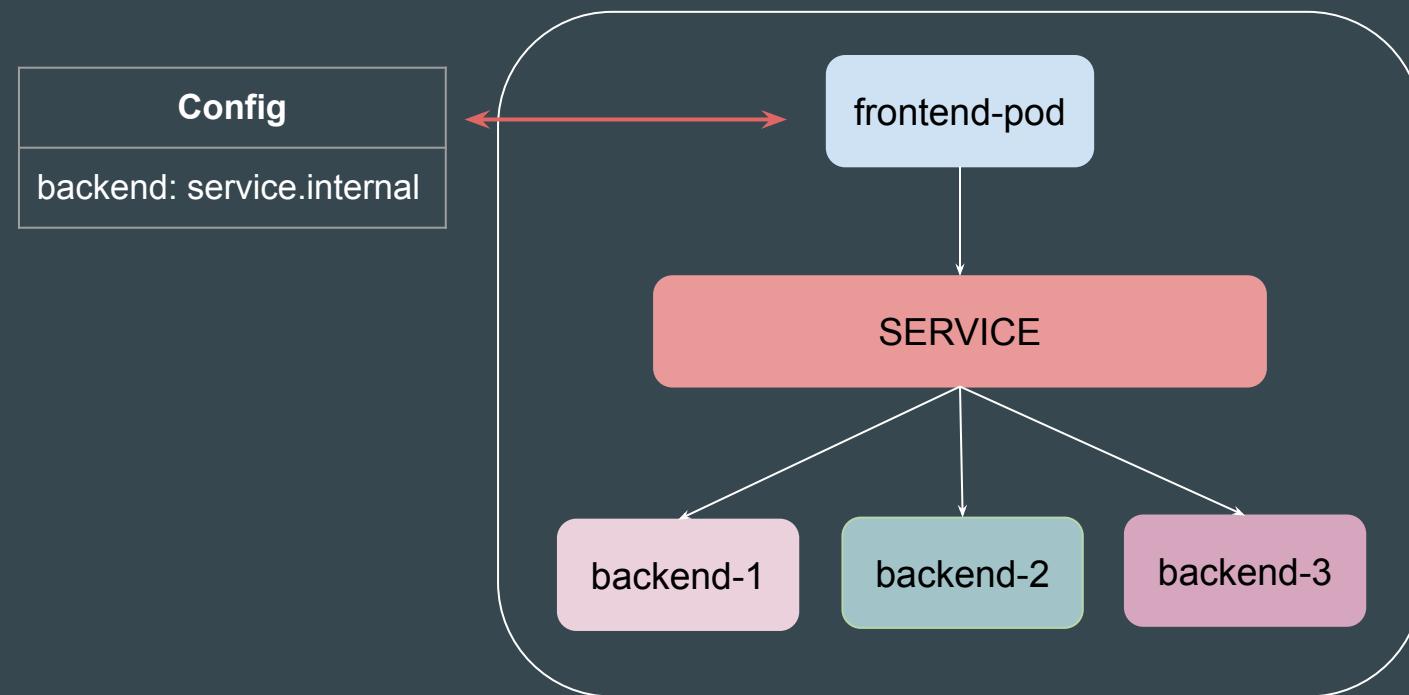


# Reference Screenshot - Distributing Traffic

```
root@frontend-pod:/# curl service.internal  
Backend Pod 2  
root@frontend-pod:/# curl service.internal  
Backend Pod 1  
root@frontend-pod:/# curl service.internal  
Backend Pod 2  
root@frontend-pod:/# curl service.internal  
Backend Pod 1  
root@frontend-pod:/# curl service.internal  
Backend Pod 2  
root@frontend-pod:/# curl service.internal  
Backend Pod 1
```

# Introducing Service

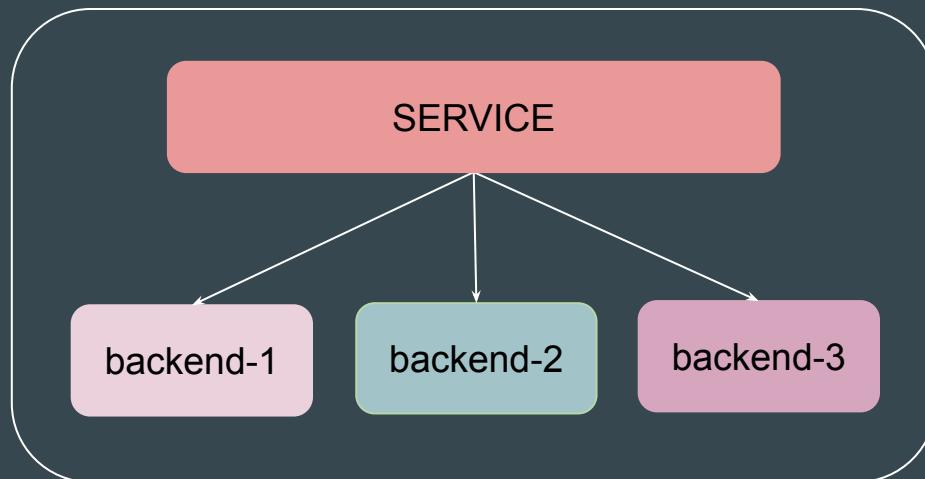
Service acts as a gateway that distributes incoming traffic between its endpoints.



# Service and Deployments

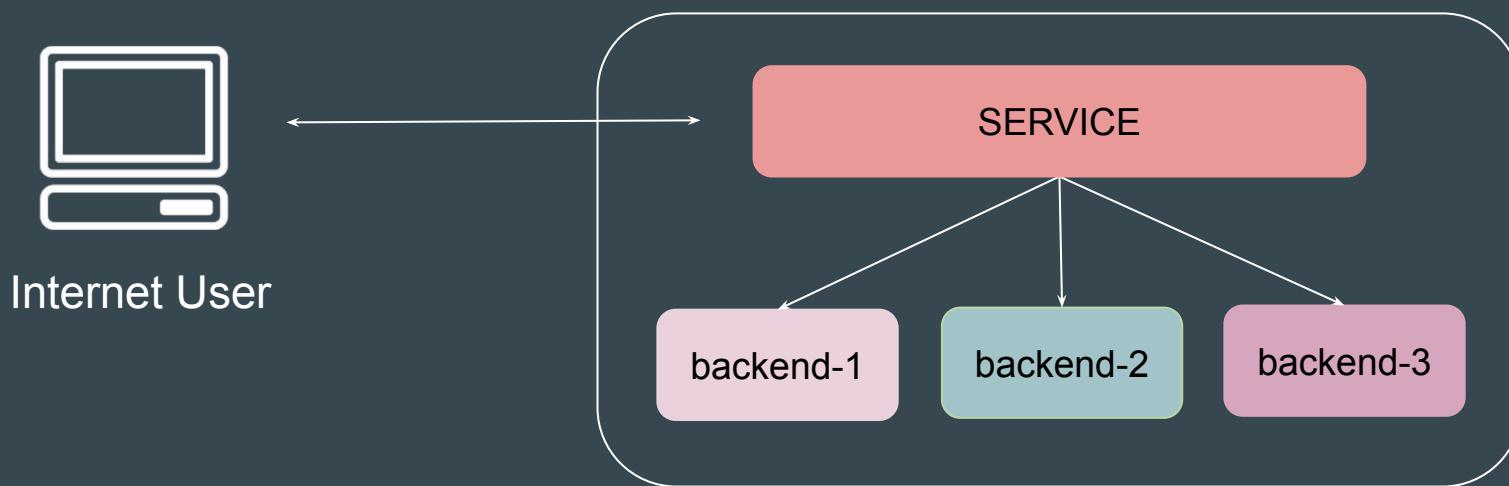
If you use a Deployment to run your app, that Deployment can create and destroy Pods dynamically.

Service can find the latest set of pods and route the traffic accordingly.



# Points to Note

Using Service, users outside of Kubernetes cluster can also connect to the Pods internal to the cluster.



# Benefits of Service

Pods are ephemeral, and their IPs change frequently. Services provide a stable endpoint.

Services distribute traffic across multiple Pod replicas.

Service enables exposing applications to external traffic like from the Internet.

# Types of Services

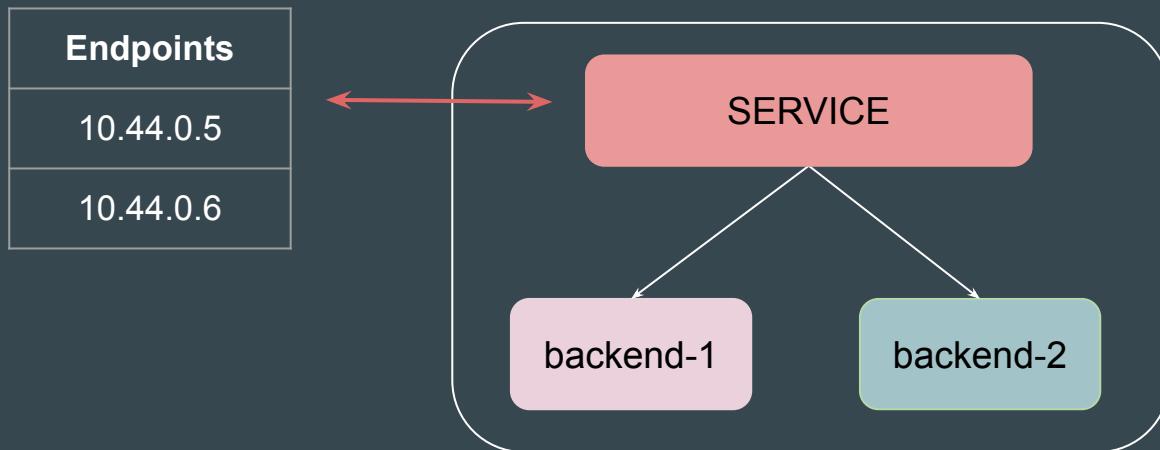
Service Type	Key Features	Use-Cases
ClusterIP	Default Service type. Accessible only within the cluster.	Internal microservices communication
NodePort	Exposes service on a static port (30000-32767) on each Node.	Development testing, demo applications
LoadBalancer	Exposes service externally using cloud provider's load balancer	Production applications requiring external access
ExternalName	Maps service to external DNS name	External service integration

# **Practical - Service and Endpoints**

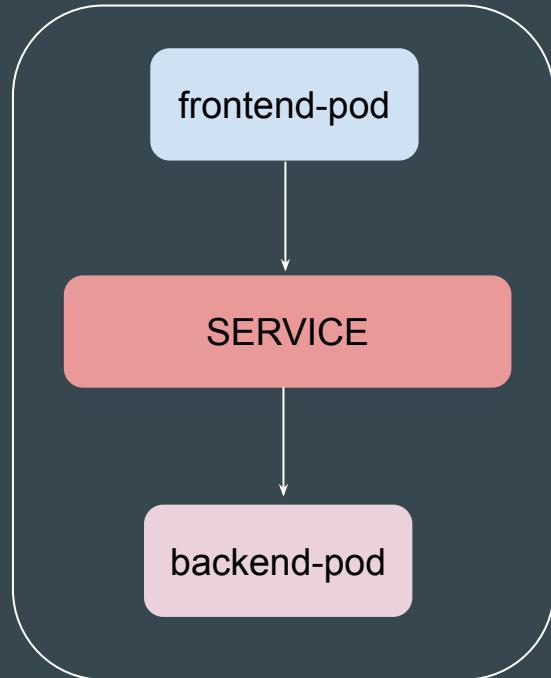
# Setting the Base

**Service** is a gateway that distributes the incoming traffic between its endpoints

**Endpoints** contain the address of underlying Pods to which the service will route the traffic to.



# Architecture of Today's Video



# Step 1 - Create Simple Service

Create a Simple Service in Kubernetes. This service will have its own IP.

```
C:\>kubectl describe service kplabs-service
Name:           kplabs-service
Namespace:      default
Labels:         <none>
Annotations:   <none>
Selector:       <none>
Type:          ClusterIP
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.109.24.231
IPs:           10.109.24.231
Port:          <unset>  8080/TCP
TargetPort:    80/TCP
Endpoints:     <none>
Session Affinity: None
Events:        <none>
```

## Step 2 - Create Endpoint for Service

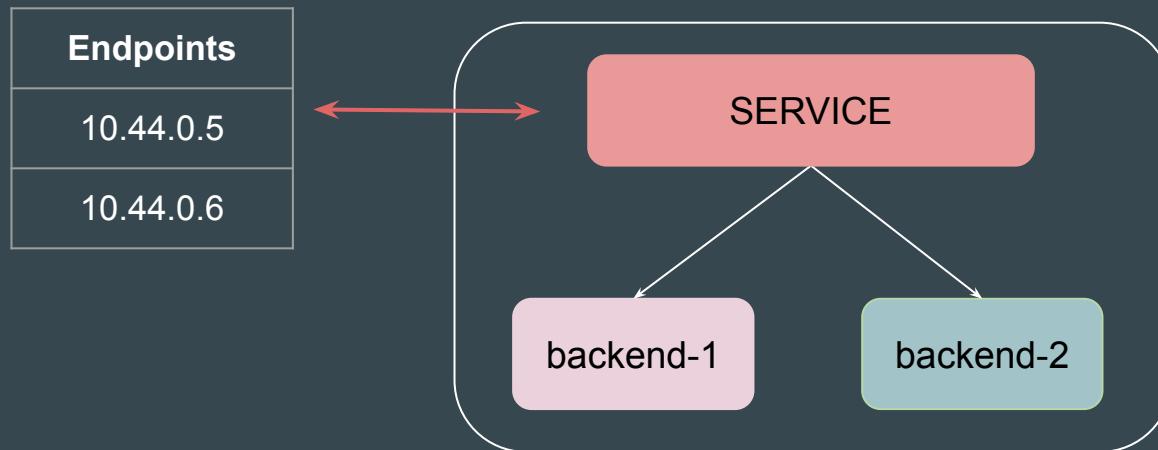
Add appropriate endpoint to the service manually for service to route traffic to.

```
C:\>kubectl describe service kplabs-service
Name:           kplabs-service
Namespace:      default
Labels:         <none>
Annotations:   <none>
Selector:       <none>
Type:          ClusterIP
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.109.12.34
IPs:           10.109.12.34
Port:          <unset>  8080/TCP
TargetPort:    80/TCP
Endpoints:     10.108.0.1:80 ←
Session Affinity: None
Events:        <none>
```

# **Using Selector for Registering Service Endpoints**

# Setting the Base

In a simple manual workflow, you can create a service and manually add endpoints associated with that service.



# Reference Manifest File

! service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: simple-service
spec:
  ports:
    - port: 80
      targetPort: 80
```

! endpoints.yaml

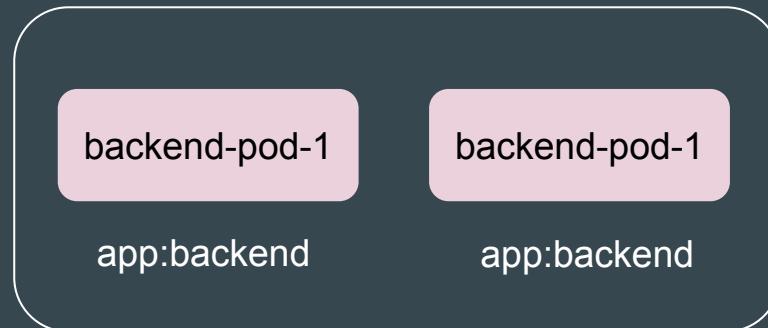
```
apiVersion: v1
kind: Endpoints
metadata:
  name: simple-service
subsets:
  - addresses:
      - ip: 10.108.0.67
    ports:
      - port: 80
```

# Better Approach - Selectors

You can also configure the Service to use **Selectors** to automatically add appropriate Pod IPs within its endpoint.

! service-selector.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: simple-service
spec:
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 80
```



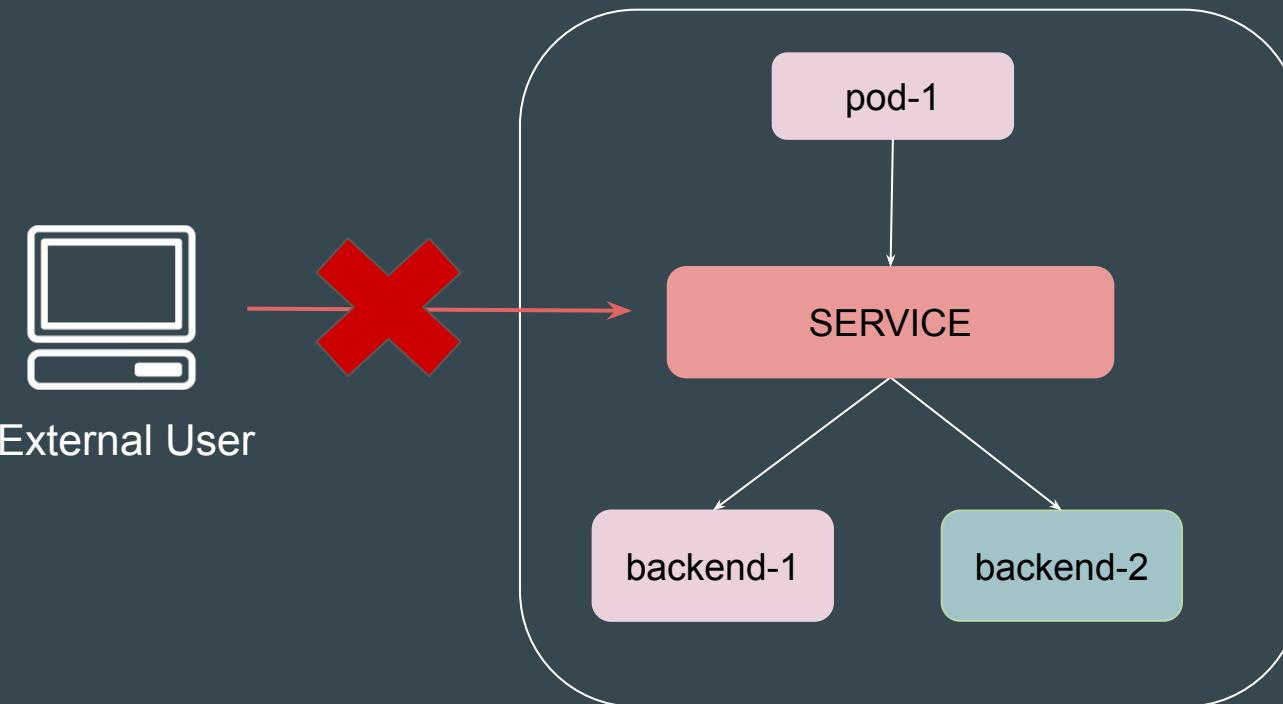
# **Service Type - ClusterIP**

# Types of Services

Service Type	Key Features	Use-Cases
ClusterIP	Default Service type. Accessible only within the cluster.	Internal microservices communication
NodePort	Exposes service on a static port (30000-32767) on each Node.	Development testing, demo applications
LoadBalancer	Exposes service externally using cloud provider's load balancer	Production applications requiring external access
ExternalName	Maps service to external DNS name	External service integration

# Setting the Base

A Kubernetes Service of type **ClusterIP** provides an internal, stable IP address to expose your application **only within the Kubernetes cluster**.



# Point to Note - Part 1

ClusterIP is a **default service type**. If you don't specify a type when creating a Service, Kubernetes defaults to ClusterIP.

```
! service.yaml
apiVersion: v1
kind: Service
metadata:
  name: kplabs-service
spec:
  ports:
  - port: 8080
    targetPort: 80
```



```
C:\>kubectl get service
NAME         TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kplabs-service  ClusterIP  10.109.12.34  <none>        8080/TCP   6m47s
kubernetes     ClusterIP  10.109.0.1    <none>        443/TCP    21d
```

## Point to Note - Part 2

The ClusterIP Service gets a virtual IP address (also called the cluster IP) that remains stable as long as the Service exists.

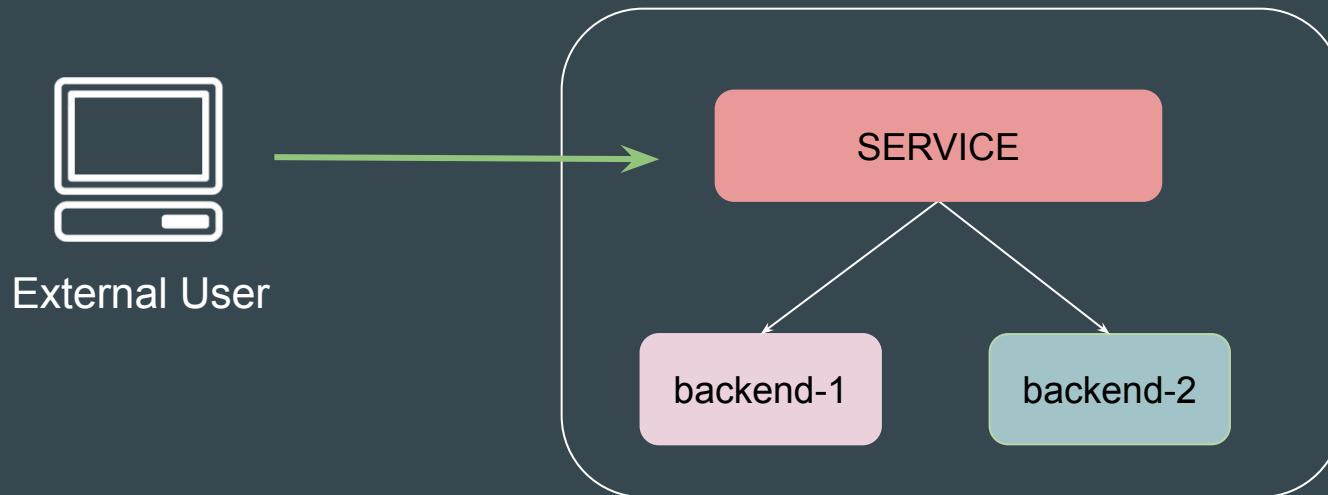
This IP can be used by other Pods inside the cluster to access the Service.

```
C:\>kubectl get service
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
kplabs-service  ClusterIP  10.109.12.34   <none>          8080/TCP      6m47s
kubernetes      ClusterIP  10.109.0.1     <none>          443/TCP       21d
```

# **Service Type - NodePort**

# Setting the Base

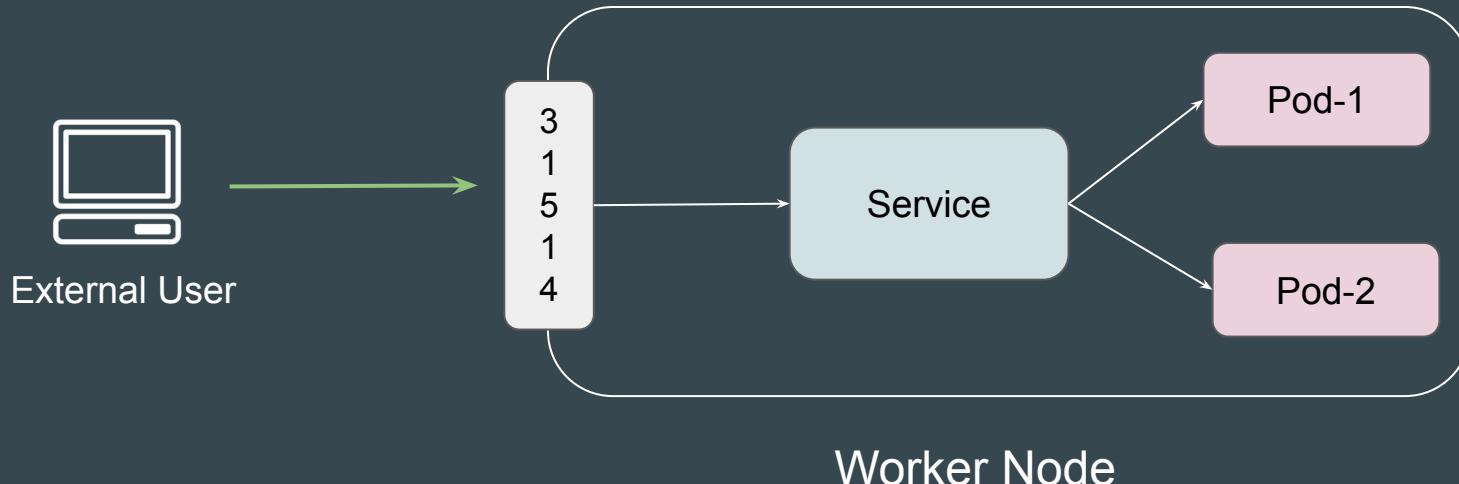
A **NodePort** is one of the service types used to expose your application to the external world.



# How it Works

When you create a service of type NodePort, Kubernetes assigns a port from the NodePort range (default: 30000-32767) on all the nodes in the cluster..

Any traffic sent to <NodeIP>:NodePort is forwarded to the corresponding service



# Reference Screenshot

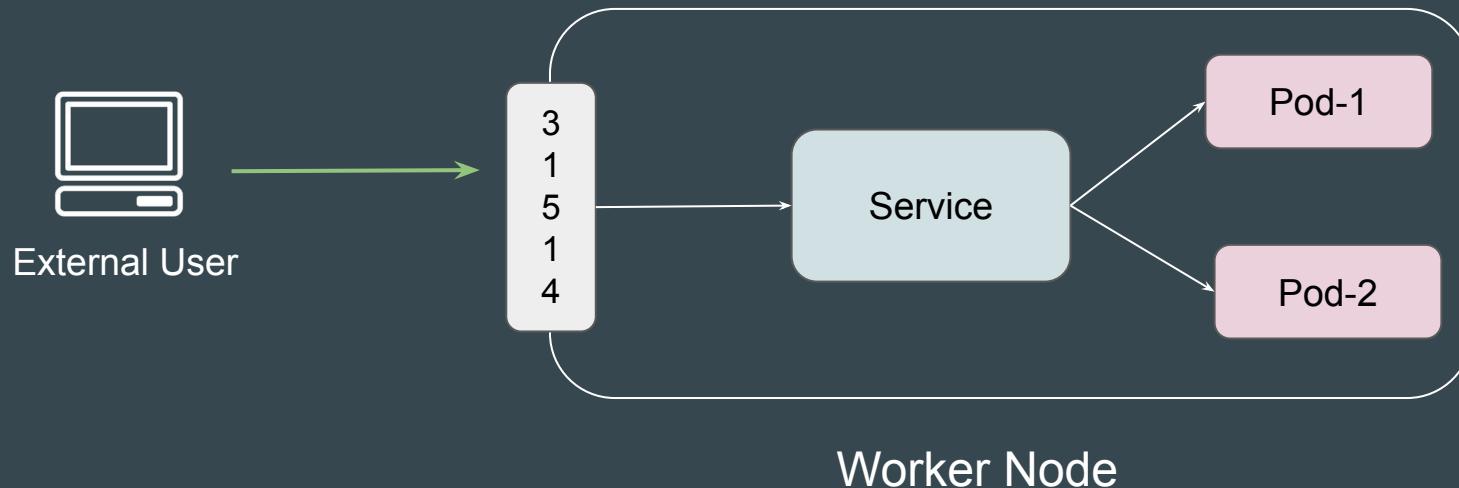
```
C:\>kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.109.0.1	<none>	443/TCP	22d
test-nodeport	NodePort	10.109.4.218	<none>	80:30537/TCP	10s

# **Service Type - LoadBalancer**

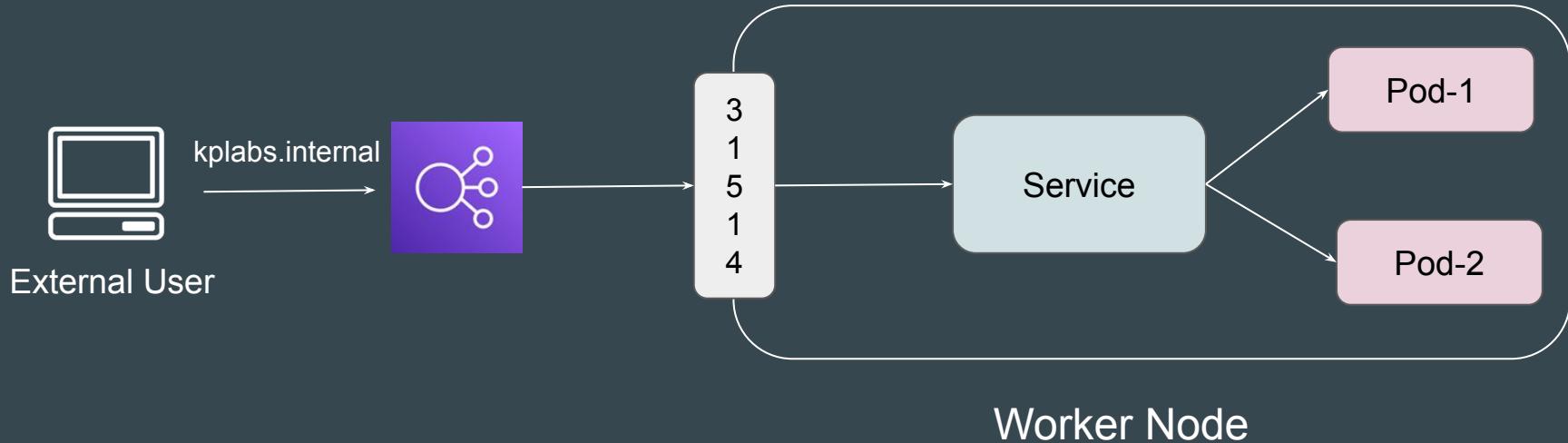
# Challenge with NodePort

To access a NodePort based service, we had to make a request to DNS/IP:NodePort.



# Introducing Load Balancer Service Type

This type creates an External Load Balancer in a Cloud Provider and routes the request received in Load Balancer to underlying NodePort.



# Point to Note

The LoadBalancer service automatically creates a NodePort service behind the scenes.

The external load balancer then directs traffic to these NodePorts, and the traffic is subsequently forwarded to the appropriate pods

C:\>kubectl get service						
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
kplabs-loadbalancer	LoadBalancer	10.109.1.176	146.190.10.2	80:31249/TCP	8m42s	
kubernetes	ClusterIP	10.109.0.1	<none>	443/TCP	22d	

# Point to Note - Supported Provider

This service works only with supported cloud providers. For on-premises or custom setups, additional configuration is required.

## Networking

Domains   Reserved IPs   **Load Balancers**   VPC   Firewalls   PTR records

Kubernetes load balancers must be [added and configured through kubectl](#).

Name	Status	IP Address	Size
 <a href="#">abb6d4491bbe24cbc9f5044229b06527</a> 📍 Regional / ⚙️ External / BLR1 1 Kubernetes node	<span>● Healthy 1/1 Nodes</span>	146.190.10.2	1

# Reference Screenshot - Forwarding Rules

The screenshot shows the CloudBees Kubernetes Load Balancer interface. At the top, there's a navigation bar with a cluster icon, the ID **abb6d4491bbe24cbc9f5044229b06527**, and the location **KPLABS Development / Regional / External / BLR1 / default-blr1 / kplabs-k8s / 146.190.10.2**. Below the navigation bar, there are tabs for **Kubernetes Nodes**, **Graphs**, and **Settings**, with **Settings** being the active tab.

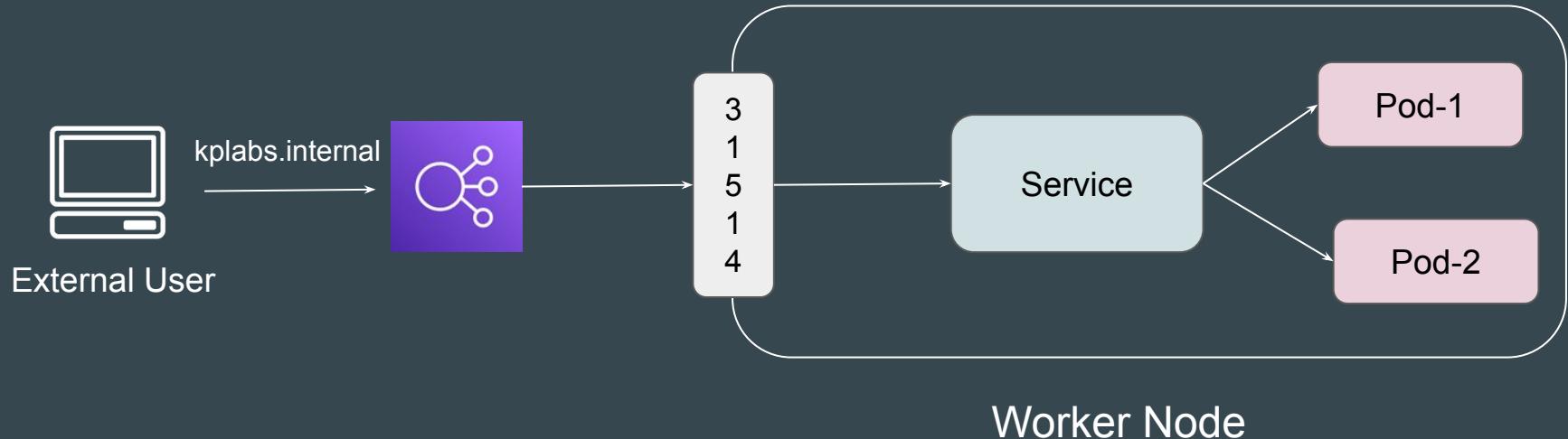
The main content area is titled **Settings**. It contains a note: **Load balancers provisioned by Kubernetes can only be modified through kubectl.** Below this note, it says: **Follow the accompanying how-to guides below to edit settings through kubectl. We also have more guidance around [configuring advanced settings](#) cluster's resource configuration file.**

Below the note, there are sections for **Scaling configuration** (Load Balancer - 1 Node), **Total monthly cost** (\$12 / month), and a **Resize** button. There's also a section for **Forwarding rules** showing a rule: **TCP on port 80 → TCP on port 31249** with an **Edit** button.

# Ingress

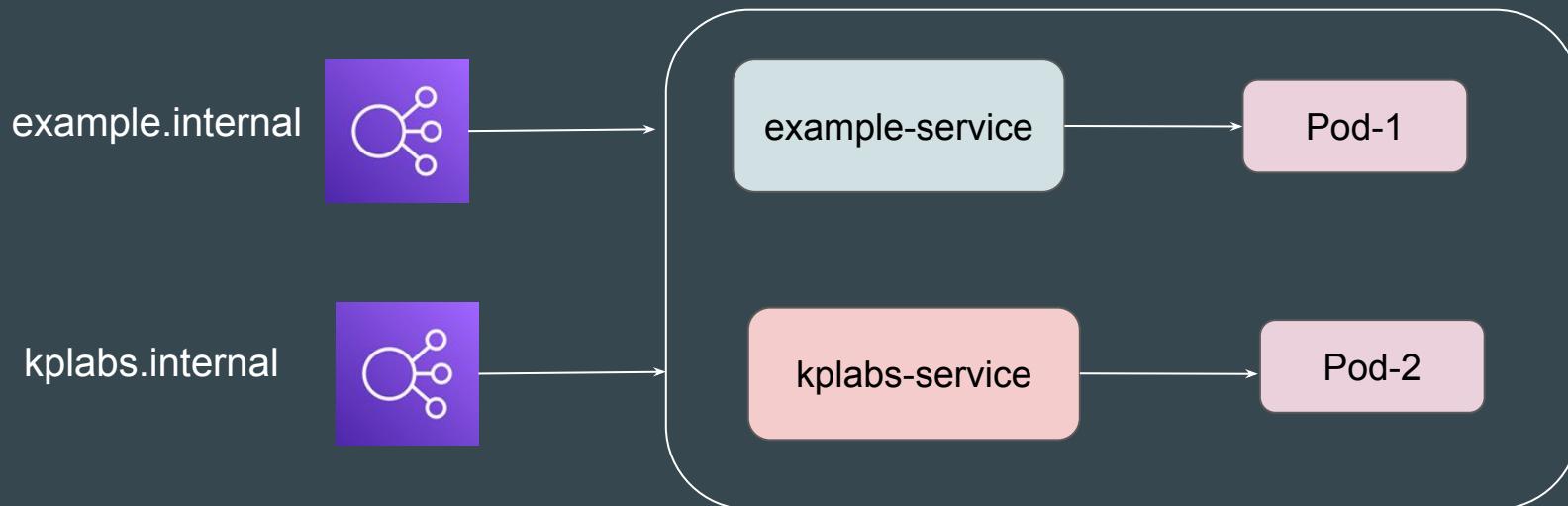
# Challenge with Basic Configuration

When we use a LoadBalancer Service Type, the Load balancer forwards traffic to a NodePort associated with a single service.



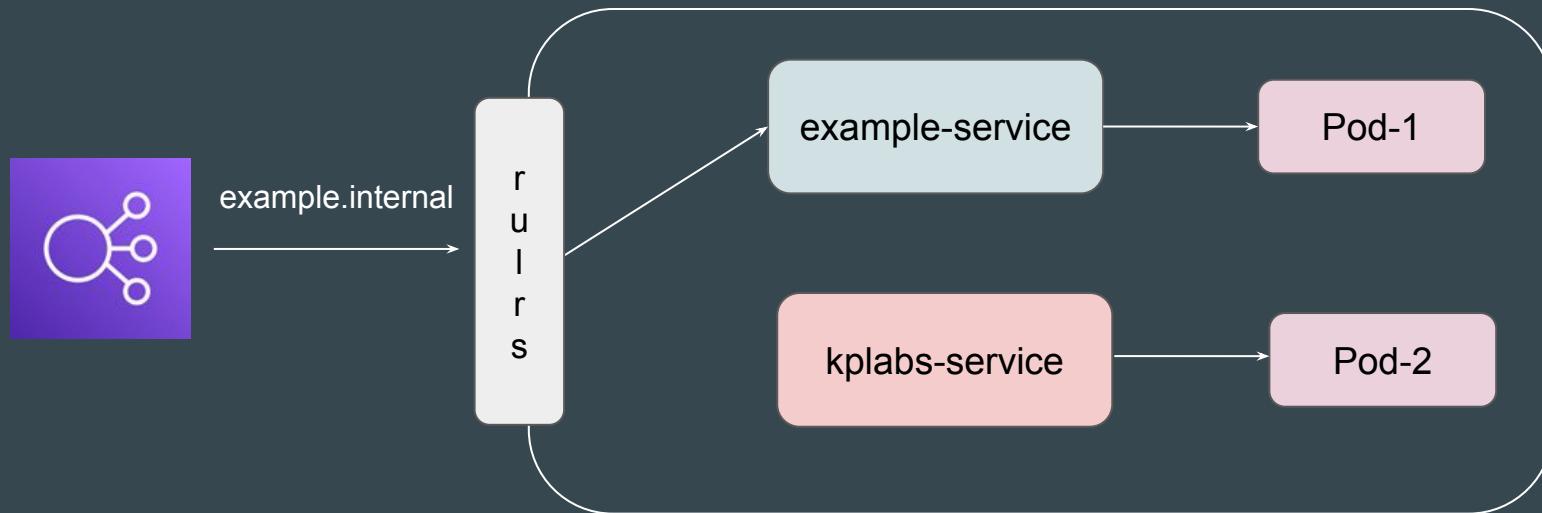
# Multiple Service Scenario

In a scenario where you have multiple services for different websites, you might have to create multiple sets of load balancers for each service. This is expensive.



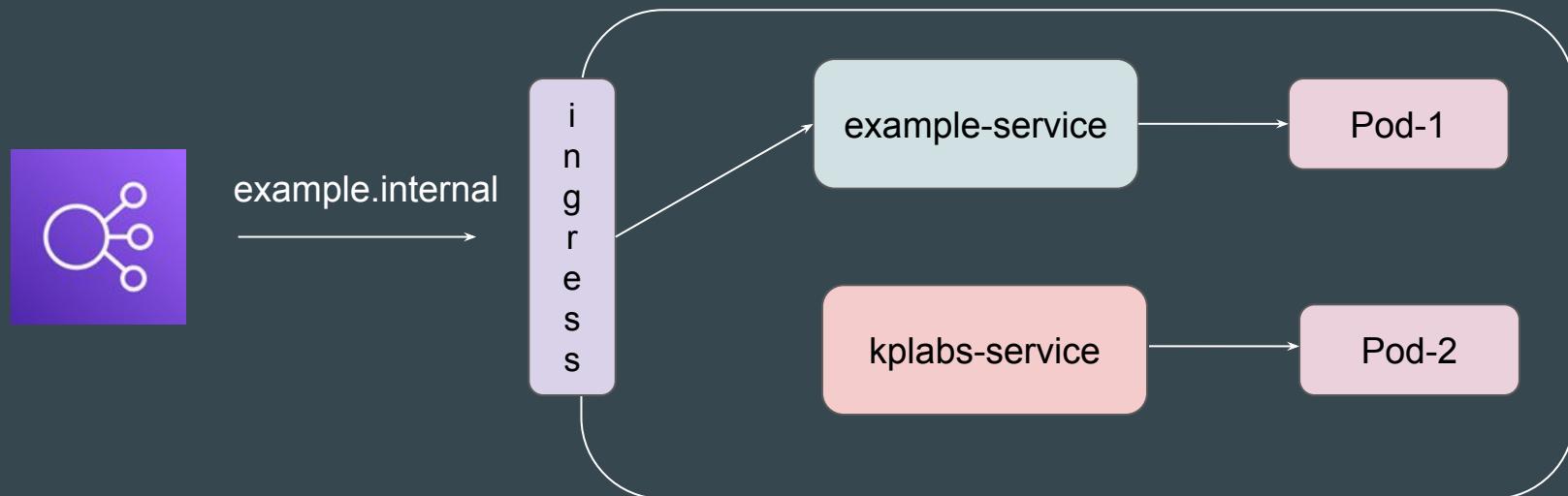
# Ideal Approach

In an ideal approach, you want a single load balancer to handle requests for multiple services and a logic that can route traffic accordingly.



# Introducing Ingress

Ingress acts as an entry point that routes traffic to specific services based on rules you define.



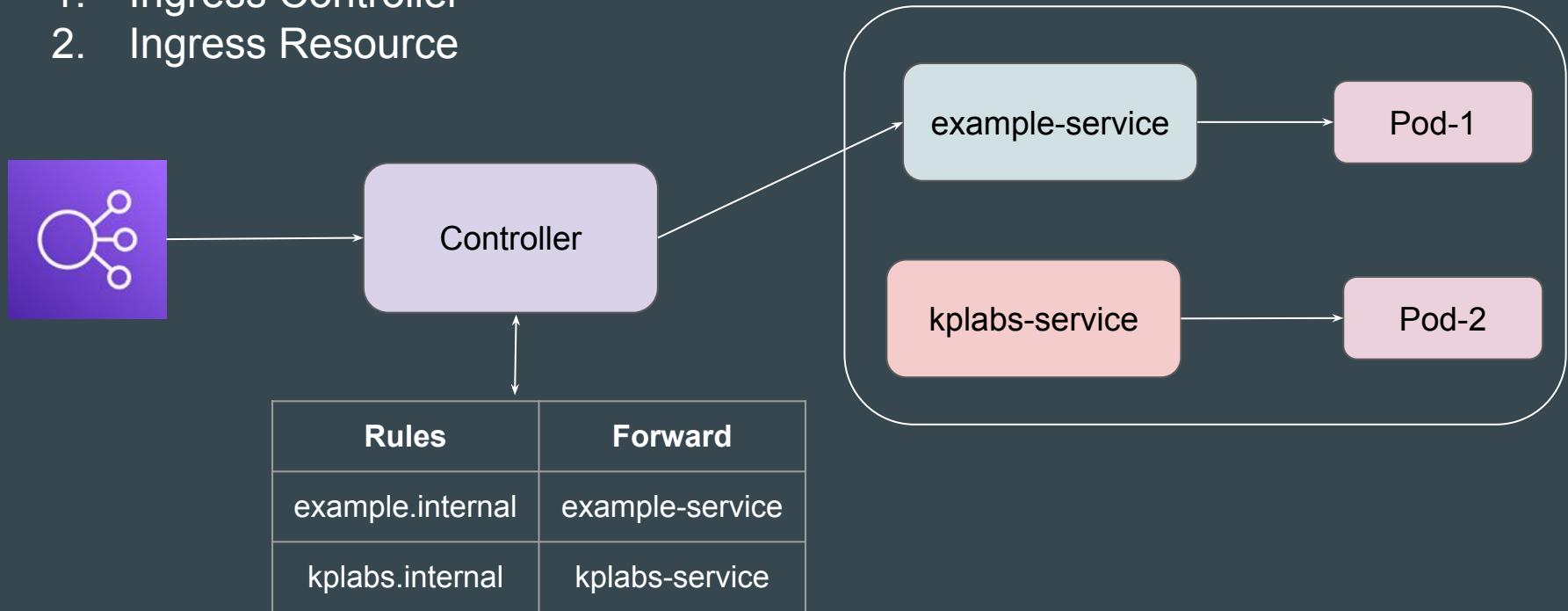
# Reference Diagram



# Components of Ingress

There are two sub-components of Ingress:

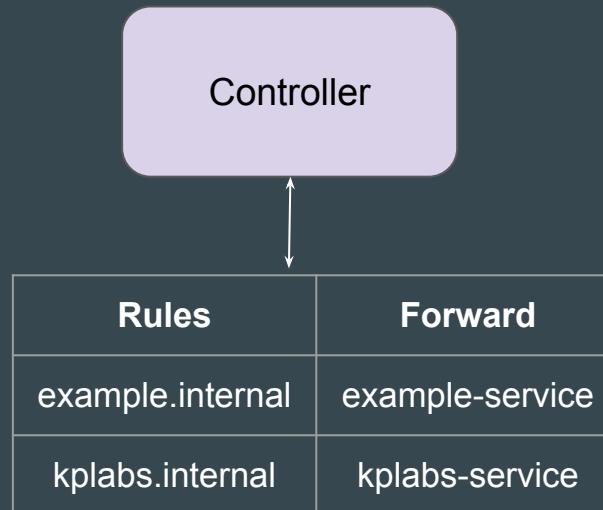
1. Ingress Controller
2. Ingress Resource



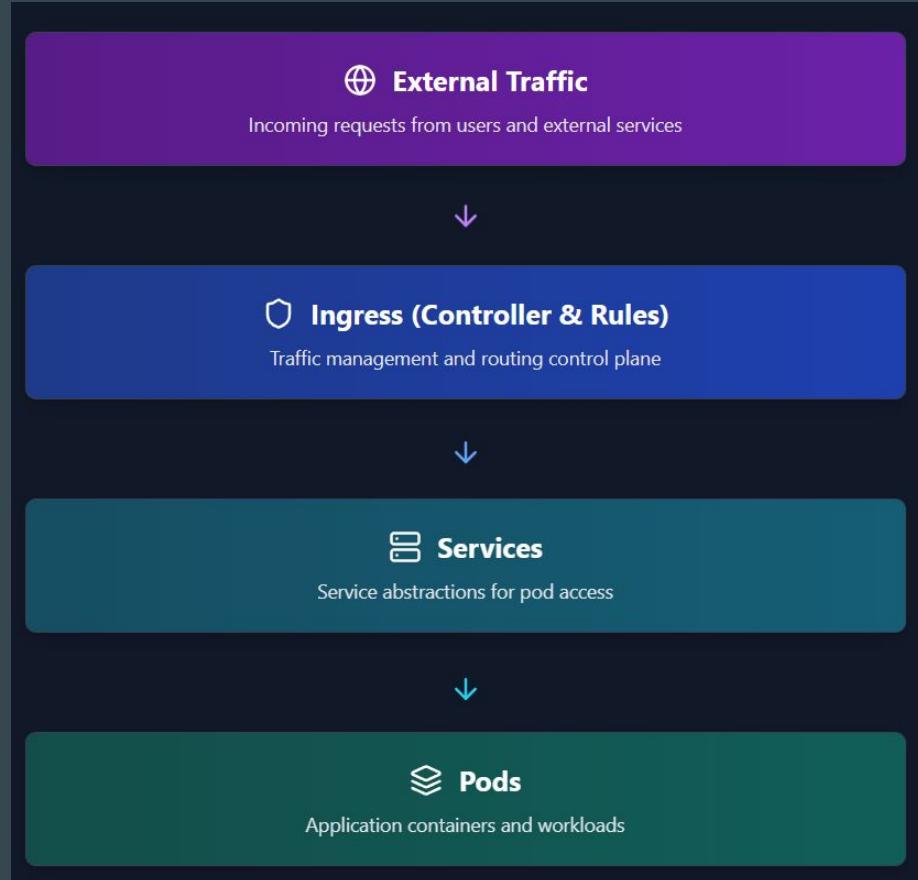
# Components of Ingress

An **Ingress Controller** is a component that implements the rules defined in Ingress resources.

Ingress Controller is a running application within your cluster.



# Reference Workflow Diagram



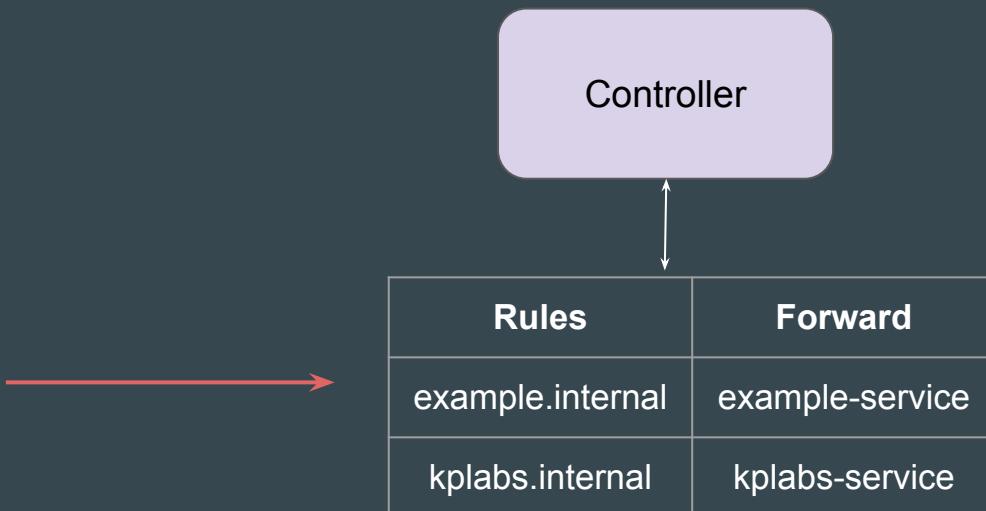
# Key Difference Summarized

Ingress	Ingress Controllers
API object (rules, configuration)	Application (implements the rules)
Defines routing rules	Enforces routing rules, manages traffic flow

# **Ingress Rules - Practical**

# Scope of Today's Video

In today's video, our focus is going to be primarily on how to write appropriate Ingress Rules



# Creating Ingress Rules

Use the `kubectl create ingress` command

## Examples:

```
# Create a single ingress called 'simple' that directs requests to foo.com/bar to svc
# svc1:8080 with a TLS secret "my-cert"
kubectl create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

# Create a catch all ingress of "/path" pointing to service svc:port and Ingress Class as "otheringress"
kubectl create ingress catch-all --class=otheringress --rule="/path=svc:port"

# Create an ingress with two annotations: ingress.annotation1 and ingress.annotations2
kubectl create ingress annotated --class=default --rule="foo.com/bar=svc:port" \
--annotation ingress.annotation1=foo \
--annotation ingress.annotation2=bla

# Create an ingress with the same host and multiple paths
kubectl create ingress multipath --class=default \
--rule="foo.com/=svc:port" \
--rule="foo.com/admin/=svcadmin:portadmin"

# Create an ingress with multiple hosts and the pathType as Prefix
kubectl create ingress ingress1 --class=default \
--rule="foo.com/path*=svc:8080" \
--rule="bar.com/admin*=svc2:http"
```

# Example 1 - Ingress with 1 Rule

Create an ingress rule that routes all traffic for the domain of kplabs.internal to be routed to kplabs-service on port 80

```
C:\>kubectl describe ingress demo-ingress
Name:           demo-ingress
Labels:         <none>
Namespace:      default
Address:
Ingress Class: <none>
Default backend: <default>
Rules:
  Host          Path  Backends
  ----          ----  -----
  kplabs.internal
                  /    kplabs-service:80 (10.108.0.113:80)
Annotations:    <none>
```

## Example 2 - Ingress with 2 Rules

Create Ingress with two rules where traffic for kplabs.internal domain be routed to kplabs-service, and traffic for example.internal domain be routed to example-service.

```
C:\>kubectl describe ingress demo-ingress
Name:           demo-ingress
Labels:         <none>
Namespace:      default
Address:
Ingress Class: <none>
Default backend: <default>
Rules:
  Host          Path  Backends
  ----          ----  -----
  kplabs.internal   /    kplabs-service:80 (10.108.0.113:80)
  example.internal /    example-service:80 (10.108.0.93:80)
Annotations:    <none>
```

# Reference Manifest File

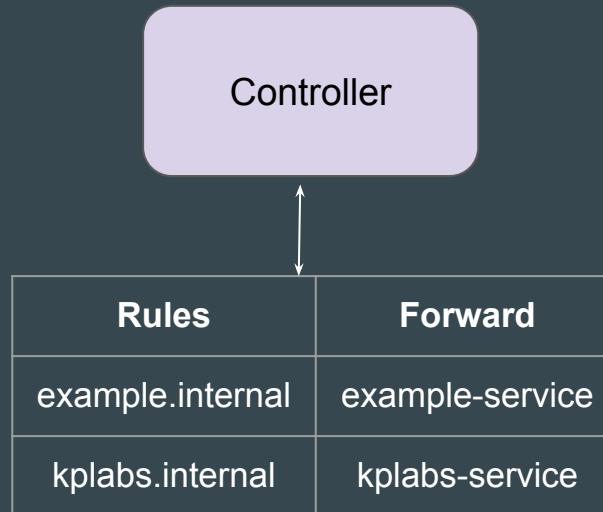
```
! ingress-kplabs.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-ingress
spec:
  rules:
  - host: kplabs.internal
    http:
      paths:
      - backend:
          service:
            name: kplabs-service
            port:
              number: 80
        path: /
        pathType: Prefix
```

# **Ingress Controller - Practical**

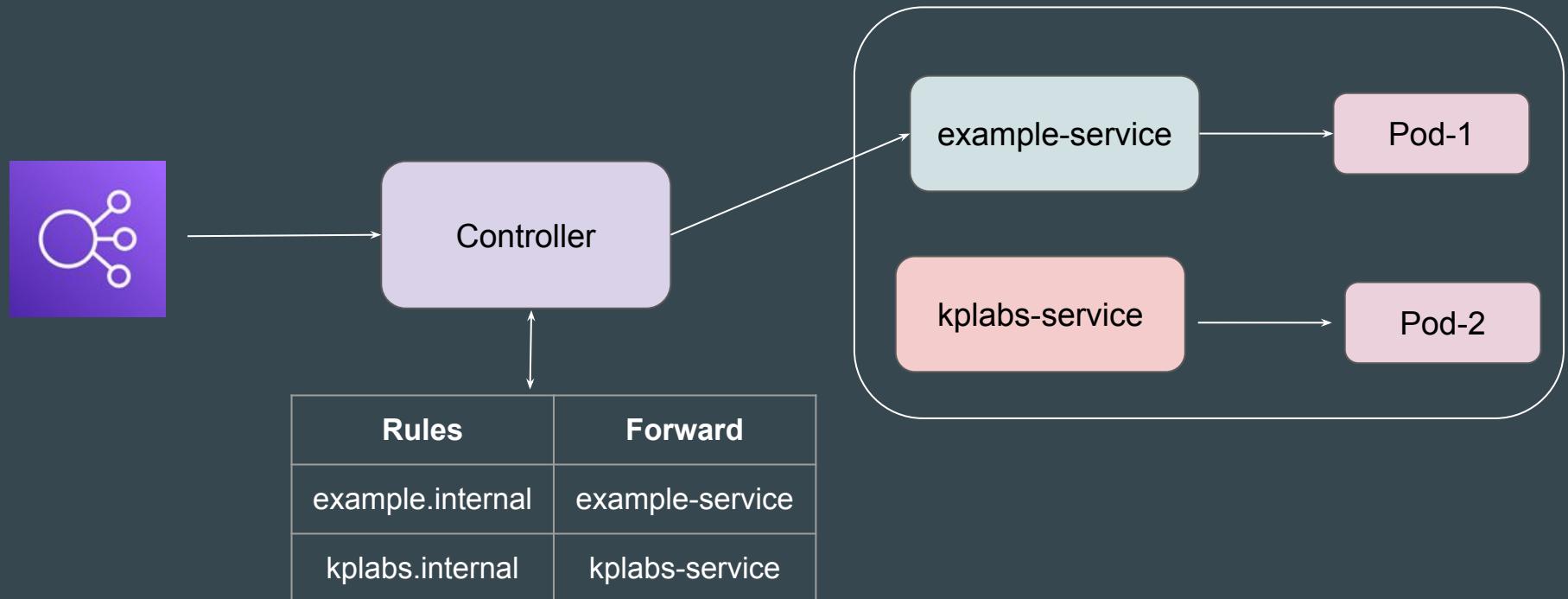
# Setting the Base

We have learnt on how to create ingress rules.

The main controller that can implement the rules is still missing.



# Final Practical Architecture



# Controllers Maintained by Kubernetes

Kubernetes as a project supports and maintains AWS, GCE, and nginx ingress controllers

There are various other 3rd party controllers also available.

The screenshot shows the GitHub repository page for the Ingress NGINX Controller. The page has a dark theme with white text. At the top, there's a navigation bar with links like 'Home', 'Code', 'Issues', 'Pull requests', 'Commits', 'Releases', and 'Wiki'. Below the navigation, there's a header with the repository name 'Ingress NGINX Controller' and a 'fork' button. A summary box displays metrics: 1.2k stars, 18k forks, 1.2k open issues, 1.2k pull requests, 1.2k commits, and 1.2k releases. Below the summary, there's a 'About' section with a 'Readme' link. The main content area has sections for 'Overview', 'Get started', and 'FAQ'. The 'Overview' section contains a brief description of the project and a link to the Kubernetes documentation. The 'Get started' section includes a link to the 'Getting Started' document and a note about not using it in multi-tenant production environments. The 'FAQ' section has a link to the 'FAQ' page.

Ingress NGINX Controller

openssf best practices in progress 96% go report A+ license Apache-2.0 18k Stars 18k contributions welcome

## Overview

ingress-nginx is an Ingress controller for Kubernetes using [NGINX](#) as a reverse proxy and load balancer.

[Learn more about Ingress on the Kubernetes documentation site.](#)

## Get started

See the [Getting Started](#) document.

Do not use in multi-tenant Kubernetes production installations. This project assumes that users that can create Ingress objects are administrators of the cluster. See the [FAQ](#) for more.

---

# Overview of Helm

Package Manager for Kubernetes

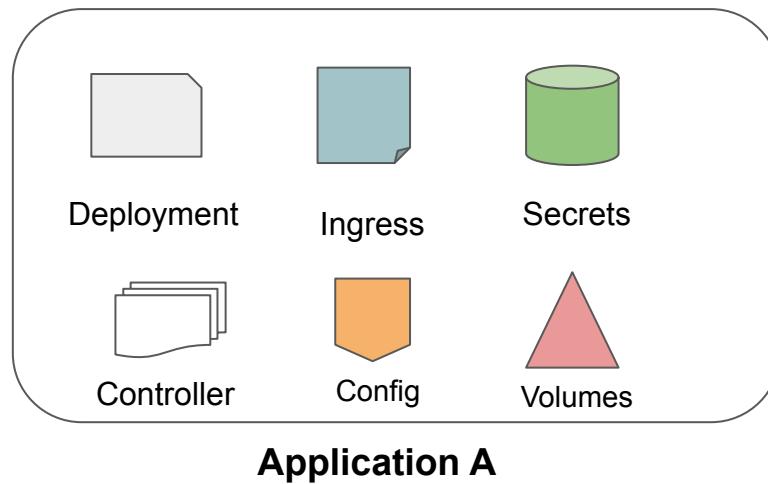
---

# Understanding Helm

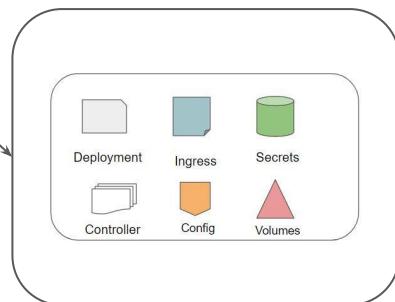
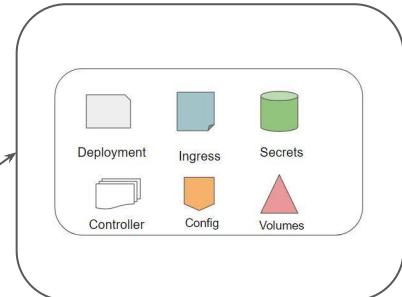
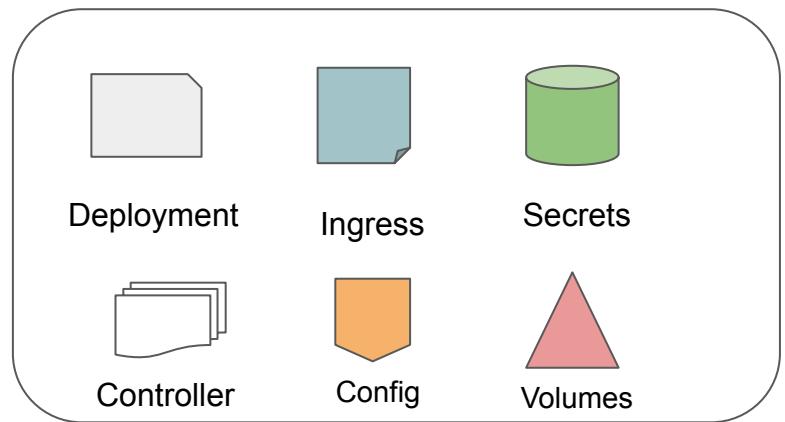
Helm is one of the package manager for Kubernetes.

Kubernetes application can contain lot of lot of objects like:

Deployments, Secrets, LoadBalancers, Volumes, services, ingress and others.

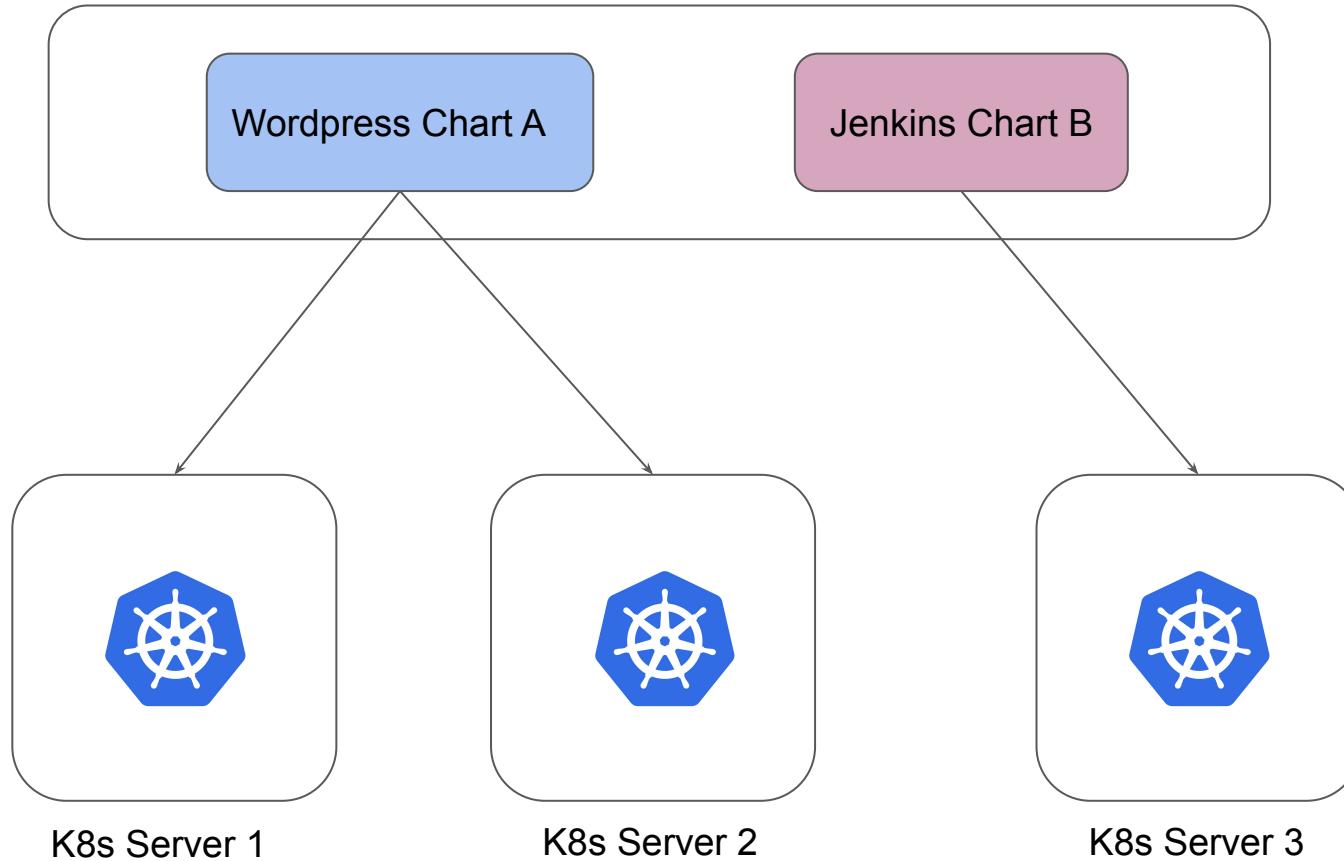


# Deploying Helm Charts



# Repository for Helm Charts

Public Repo



# Revising Helm Concepts

A Chart is a Helm package.

It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster.

A Repository is the place where charts can be collected and shared

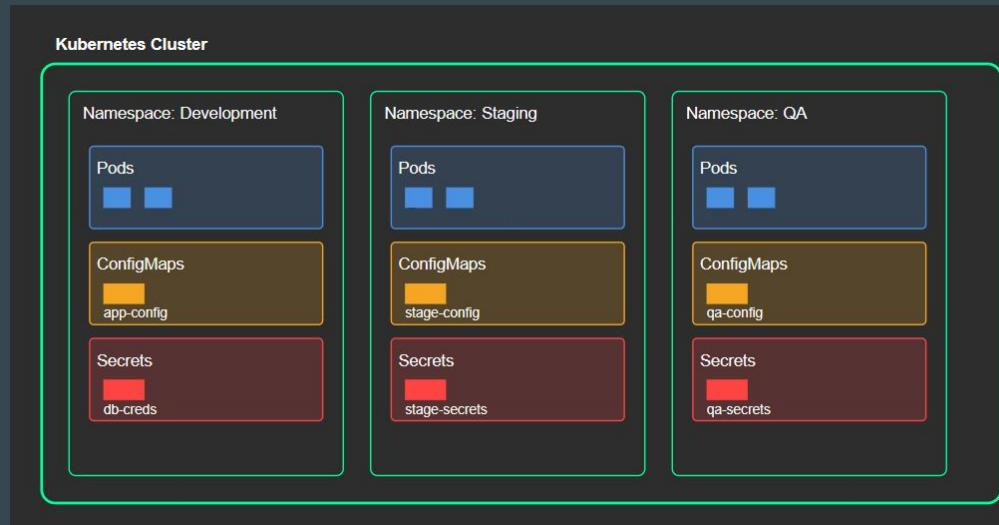
# **Helm Practical**

<b>Helm Command</b>	<b>Description</b>
helm search <repo/hub> <keyword>	Provides the ability to search for Helm charts in the various places they can be stored including the Artifact Hub and repositories you have added.
helm repo add <name> <url>	Adds a Helm chart repository (e.g. nginx)
helm install <release> <chart>	Installs a chart as a release into Kubernetes
helm uninstall <release>	Uninstalls (deletes) a Helm release
helm list	Lists all installed Helm releases
helm template <release> <chart>	Renders Kubernetes manifests locally without installing

# **Namespaces**

# Setting the Base

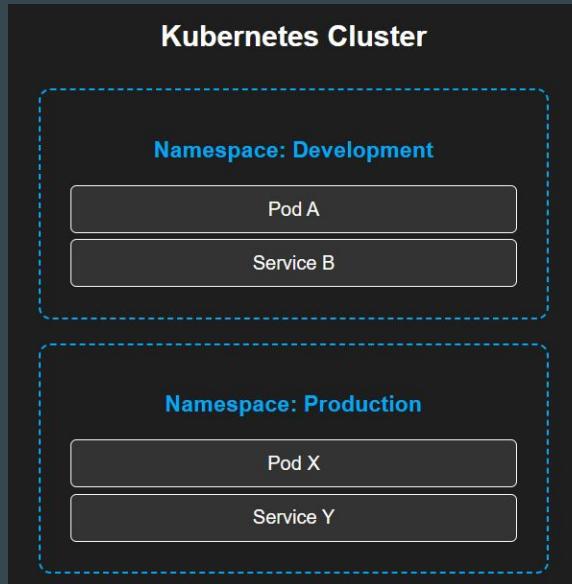
In Kubernetes, namespaces provide a mechanism for **isolating groups of resources** within a single cluster



# Benefits of Namespace

Multiple teams or projects can share the same cluster without interfering with each other.

Separate different environments like, development, QA, staging.



# Default Namespaces in Kubernetes

Kubernetes comes with the following default namespaces:

Namespace	Description
default	Used for resources with no specific namespace.
kube-system	The namespace for objects created by the Kubernetes system
kube-public	A namespace visible to all users, often used for publicly accessible data.
kube-node-lease	Used for node heartbeat leases

# Reference Screenshot

```
C:\>kubectl get namespace
NAME          STATUS   AGE
default       Active   24d
kube-node-lease Active   24d
kube-public   Active   24d
kube-system   Active   24d
```

# Creating a new Namespace

Use the `kubectl create namespace` command to create a new namespace.

```
C:\>kubectl create namespace my-namespace  
namespace/my-namespace created
```

# Creating Resource in Specific Namespace

Use the `-n <namespace>` option to **create and fetch** resource in a specific namespace.

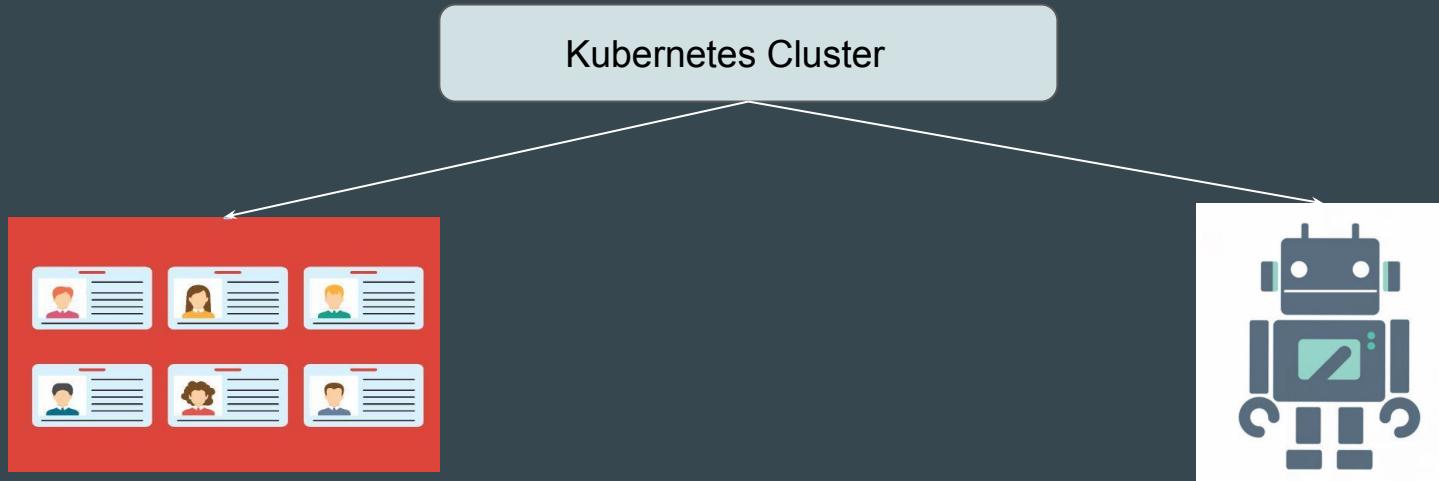
```
C:\>kubectl run test-pod --image=nginx -n my-namespace  
pod/test-pod created
```

# **Service Accounts**

# Understanding the basics

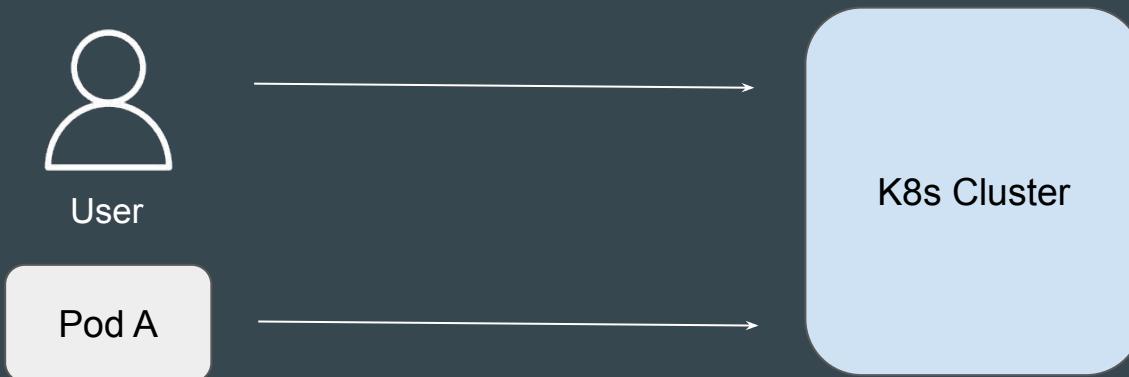
Kubernetes Clusters have two categories of accounts:

- User Accounts (For Humans).
- Service Accounts (For Applications)



# Importance of Credentials

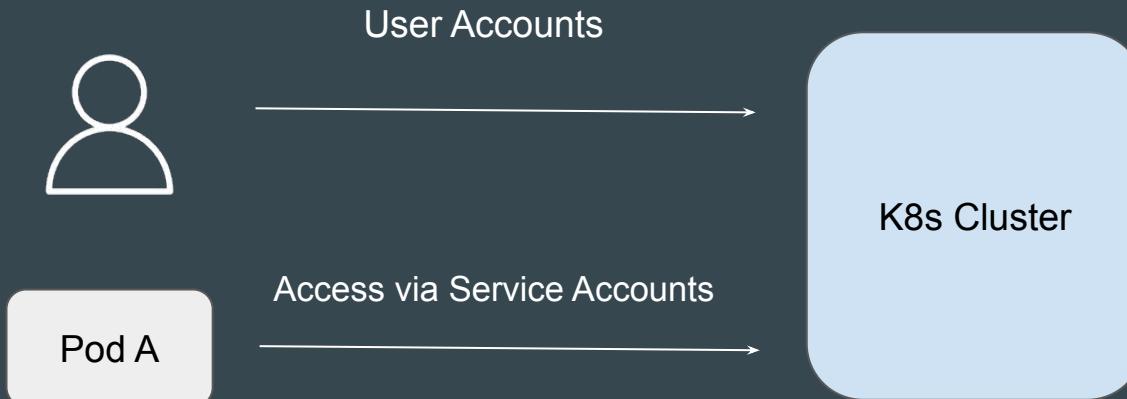
To connect to Kubernetes cluster, an entity needs to have some kind of authentication credentials.



# Different Type of Credentials

Humans will use **User Accounts** to connect to Cluster.

Pods / Applications will use **Service Accounts** to connect to Cluster.



# Service Accounts in K8s Cluster

Various different components of Kubernetes uses service accounts to communicate with the cluster

```
C:\Users\zealv>kubectl get serviceaccounts --all-namespaces
NAMESPACE      NAME          SECRETS   AGE
default        default       0         5m57s
kube-node-lease default       0         5m57s
kube-public    default       0         5m57s
kube-system   attachdetach-controller 0         5m57s
kube-system   certificate-controller 0         6m1s
kube-system   cilium        0         4m22s
kube-system   cilium-operator 0         4m22s
kube-system   cloud-controller-manager 0         5m54s
kube-system   cluster-autoscaler 0         5m10s
kube-system   clusterrole-aggregation-controller 0         5m57s
kube-system   coredns        0         102s
```

# Service Accounts and Pods

Let's assume that a Service Account is associated with Pod A.

Pod A can use the token associated with the service account to perform some actions on Kubernetes cluster.



# **Service Accounts - Points to Note**

# Default Service Account

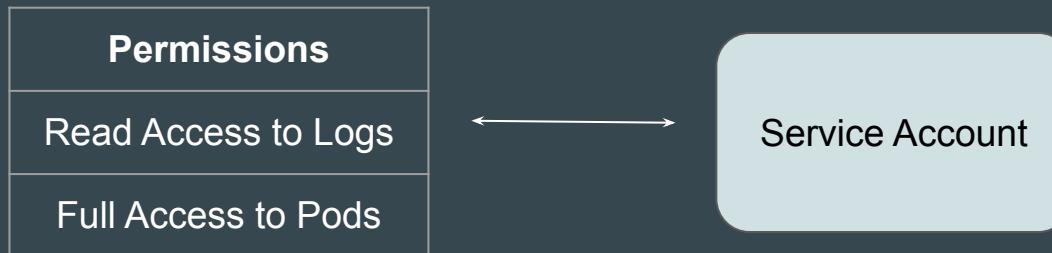
When you create a cluster, Kubernetes automatically creates a ServiceAccount object named **default** for every namespace in your cluster.

```
C:\Users\zealv>kubectl get serviceaccount
NAME      SECRETS   AGE
default    0          23m
```

# Service Account and Permissions

Each Service Account in Kubernetes can be associated with certain permissions.

When Pod uses the service account, it can inherit the permissions.



## Point to Note

The default service accounts in each namespace get no permissions by default other than the default API discovery permissions that Kubernetes grants to all authenticated principals if role-based access control (RBAC) is enabled

# Assigning Pods to Service Accounts

If you deploy a Pod in a namespace, and you don't manually assign a ServiceAccount to the Pod, Kubernetes **assigns the default ServiceAccount** for that namespace to the Pod



# Accessing Service Account Token

Service Account Token gets mounted inside the Pod inside the /var/run directory and can easily be accessed using cat command.

```
root@nginx:/# cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzIiNlIsImtpZCI6IiJLZWlNejJxdTh3NlY2SjhRZHvkSEZfMWxKazFoZmttb3JLNnNTNzTdxMifQ.eyJhdWQiolsic3lzdGVt0mtvbm5lY3Rpdm
l0eS1zZXJ2ZXIiXSwiZXhwIjoxNzI3NzU50TY2LCJpYXQiOjE2OTYyMjM5NjYsImlzcyI6Imh0dHBz0i8va3ViZXJuZXRlcyc5kZWZhdWx0LnN2Yy5jbHVzdGVyLmx
vY2FsIiwiia3ViZXJuZXRlcyc5pbI6eyJuYW1lc3BhY2Ui0iJkZWZhdWx0IiwickG9kIjp7Im5hbWUiOjJuZ2lueCIsInVpZCI6ImU3ZjUyYWY4LWM5MGUtNDY5Zi1i
MTg3LTc3MjliMmNmE5MyJ9LCJzZXJ2aWN1YWNjb3VudCI6eyJuYW1lIjoIZGVmYXVsdcIsInVpZCI6ImZlOTUzMTQ0LTThjNDYtNDF1My1iODFjLTMsMThkNTNhM
zAyOCJ9LCJ3YXJuYWZ0ZXIIoje2OTYyMjc1NzN9LCJuYmYiOjE2OTYyMjM5NjYsInN1Yi6In5c3R1bTpzZXJ2aWN1YWNjb3VudDpkZWZhdWx00mR1ZmF1bHQifQ
.i5jS2uEbNgj_1GA6LQh2YxQKEJsNjZbvoJh-Qjf-vH_f10w0KTvhmLPCL1UxCLfoI1NejT07Agckj3SFXYjEqmlvcbEYpLXt8eUjrHC8s6Ra105XGnpbt5uUy3GU
BYP4HnRE970zzU3HyU2gM14Kd8d8a9iiHb7yov82ph6moOPuuxCxBoWNo5edWMWnNDWLKLNOFX168oxAmQo8ZQI5k37INIE1oN5N2bzp7Y40Gv3CURK1X904B0YuT
UN8gSYZsHaQaVq8FT-Q_xGfGQbvjqUzi0rRaKNc9QJ0BiIE3CKQN7PL6C0Lxyt_9LieJV438xPgwer2V_aNeHBWgU99oAroot@nginx:#
```

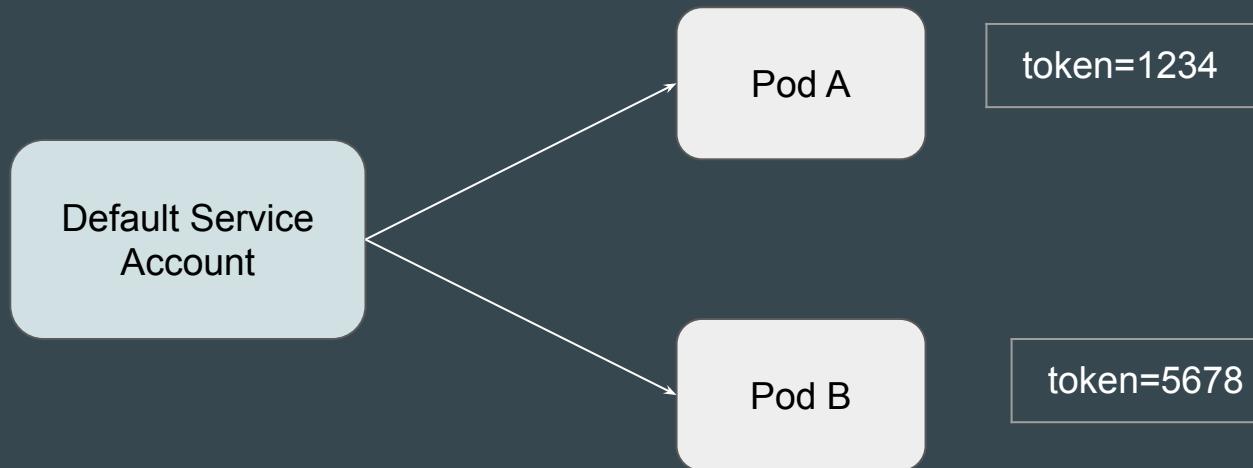
# Connecting to K8s using Token

Using the Service Account Token, you can connect to the Kubernetes Cluster to perform operations.

```
root@nginx:/var/run/secrets/kubernetes.io/serviceaccount# curl -k -H "Authorization: Bearer $t" https://0f3570d8-03b7-45af-a  
f4-4c1b90504be3.k8s.ondigitalocean.com/api/v1  
{  
    "kind": "APIResourceList",  
    "groupVersion": "v1",  
    "resources": [  
        {  
            "name": "bindings",  
            "singularName": "binding",  
            "namespaced": true,  
            "kind": "Binding",  
            "verbs": [  
                "create"  
            ]  
        },  
        {  
            "name": "namespaces",  
            "singularName": "namespace",  
            "namespaced": true,  
            "kind": "Namespace",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "pods",  
            "singularName": "pod",  
            "namespaced": true,  
            "kind": "Pod",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "events",  
            "singularName": "event",  
            "namespaced": true,  
            "kind": "Event",  
            "verbs": [  
                "list",  
                "patch"  
            ]  
        },  
        {  
            "name": "services",  
            "singularName": "service",  
            "namespaced": true,  
            "kind": "Service",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "endpoints",  
            "singularName": "endpoint",  
            "namespaced": true,  
            "kind": "Endpoint",  
            "verbs": [  
                "list",  
                "patch"  
            ]  
        },  
        {  
            "name": "leases",  
            "singularName": "lease",  
            "namespaced": true,  
            "kind": "Lease",  
            "verbs": [  
                "list",  
                "patch"  
            ]  
        },  
        {  
            "name": "configmaps",  
            "singularName": "configmap",  
            "namespaced": true,  
            "kind": "ConfigMap",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "secrets",  
            "singularName": "secret",  
            "namespaced": true,  
            "kind": "Secret",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "servicesaccounts",  
            "singularName": "serviceaccount",  
            "namespaced": true,  
            "kind": "ServiceAccount",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "nodes",  
            "singularName": "node",  
            "namespaced": false,  
            "kind": "Node",  
            "verbs": [  
                "list",  
                "patch"  
            ]  
        },  
        {  
            "name": "events",  
            "singularName": "event",  
            "namespaced": false,  
            "kind": "Event",  
            "verbs": [  
                "list",  
                "patch"  
            ]  
        }  
    ]  
}
```

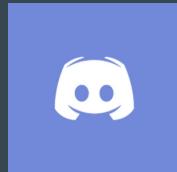
## Points to Note

Even though 2 Pods use same service account, each Pod will receive different set of tokens.



# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Service Accounts - Practical Scenarios**

# Creating Service Accounts

You can create new service account directly using kubectl.

```
C:\Users\zealv>kubectl create serviceaccount custom-token  
serviceaccount/custom-token created
```

# Mounting Custom Service Account to Pod

You can use `serviceAccountName` to specify custom service account token as part of the Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: custom-token
```

---

# Named Port

Networking Aspect

---

# Named Port

We can also associate a name associated with the port

- This name must be unique within the pod.

```
spec:  
  containers:  
    - image: nginx  
      name: nginx  
      ports:  
        - containerPort: 80
```



```
spec:  
  containers:  
    - image: nginx  
      name: nginx  
      ports:  
        - containerPort: 80  
          name: http
```

# Named Port Reference in Service File

Instead of Port Number, we can also specify the name while creating service.

```
kubectl expose pod nginx --port=80 --target-port=http --name "kplabs-svc"
```

```
spec:  
  ports:  
    - port: 80  
      protocol: TCP  
      targetPort: http  
  selector:  
    run: nginx  
  type: NodePort
```

# **Metrics Server**

# Setting the Base

By default, Kubernetes does not provide resource usage metrics.

Metrics Server collects resource usage data (CPU and memory) from the kubelet on each node and makes it available via the Kubernetes Metrics API.

C:\>kubectl top pods -A		CPU(cores)	MEMORY(bytes)
NAMESPACE	NAME		
default	curl	0m	0Mi
default	nginx	0m	2Mi
kube-system	cilium-fbqjm	11m	205Mi
kube-system	coredns-86bbcb7bcf-88kqj	2m	16Mi
kube-system	coredns-86bbcb7bcf-lvcjd	2m	16Mi
kube-system	cpc-bridge-proxy-ebpf-8t7zc	1m	1Mi
kube-system	csi-do-node-j5bkg	1m	10Mi
kube-system	do-node-agent-jjm5b	0m	21Mi
kube-system	hubble-relay-8cc9ccbbd-jj4jv	1m	23Mi
kube-system	hubble-ui-cc6cd98c6-76x9p	1m	22Mi
kube-system	konnectivity-agent-jj5ln	1m	11Mi
kube-system	kube-proxy-ebpf-8z6gk	0m	0Mi
kube-system	metrics-server-74bb687b8-8mpbn	3m	18Mi

# kubectl top pods

**kubectl top pods** command shows the CPU and memory usage of all running pods in the cluster (or within a specific namespace).

NAMESPACE	NAME	CPU(cores)	MEMORY(bytes)
default	curl	0m	0Mi
default	nginx	0m	2Mi
kube-system	cilium-fbqjm	11m	205Mi
kube-system	coredns-86bbcb7bcf-88kqj	2m	16Mi
kube-system	coredns-86bbcb7bcf-lvcjd	2m	16Mi
kube-system	cpc-bridge-proxy-ebpf-8t7zc	1m	1Mi
kube-system	csi-do-node-j5bkg	1m	10Mi
kube-system	do-node-agent-jjm5b	0m	21Mi
kube-system	hubble-relay-8cc9ccbdbd-jj4jv	1m	23Mi
kube-system	hubble-ui-cc6cd98c6-76x9p	1m	22Mi
kube-system	konnectivity-agent-jj5ln	1m	11Mi
kube-system	kube-proxy-ebpf-8z6gk	0m	0Mi
kube-system	metrics-server-74bb687b8-8mpbn	3m	18Mi

# kubectl top nodes

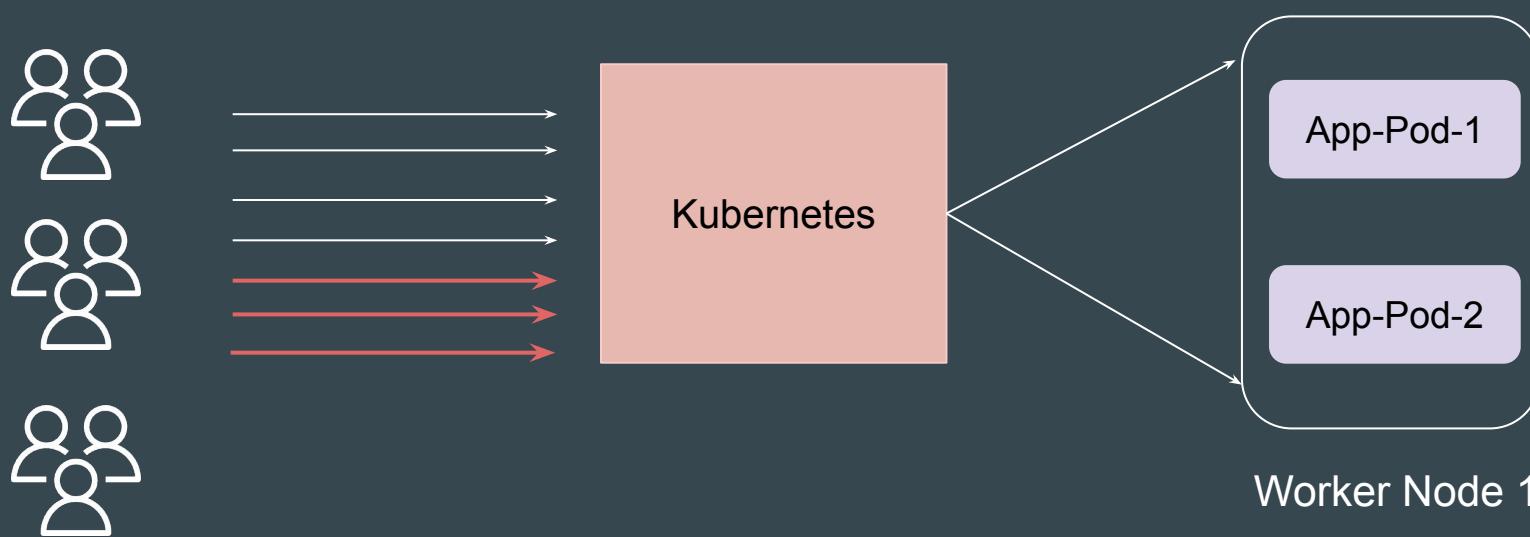
**kubectl top nodes** command displays CPU and memory usage across all nodes in the Kubernetes cluster

NAME	CPU(cores)	CPU(%)	MEMORY(bytes)	MEMORY(%)
kind-control-plane	123m	12%	604Mi	30%
kind-worker	45m	4%	268Mi	13%
kind-worker2	41m	4%	211Mi	10%

# **Horizontal Pod Autoscaling**

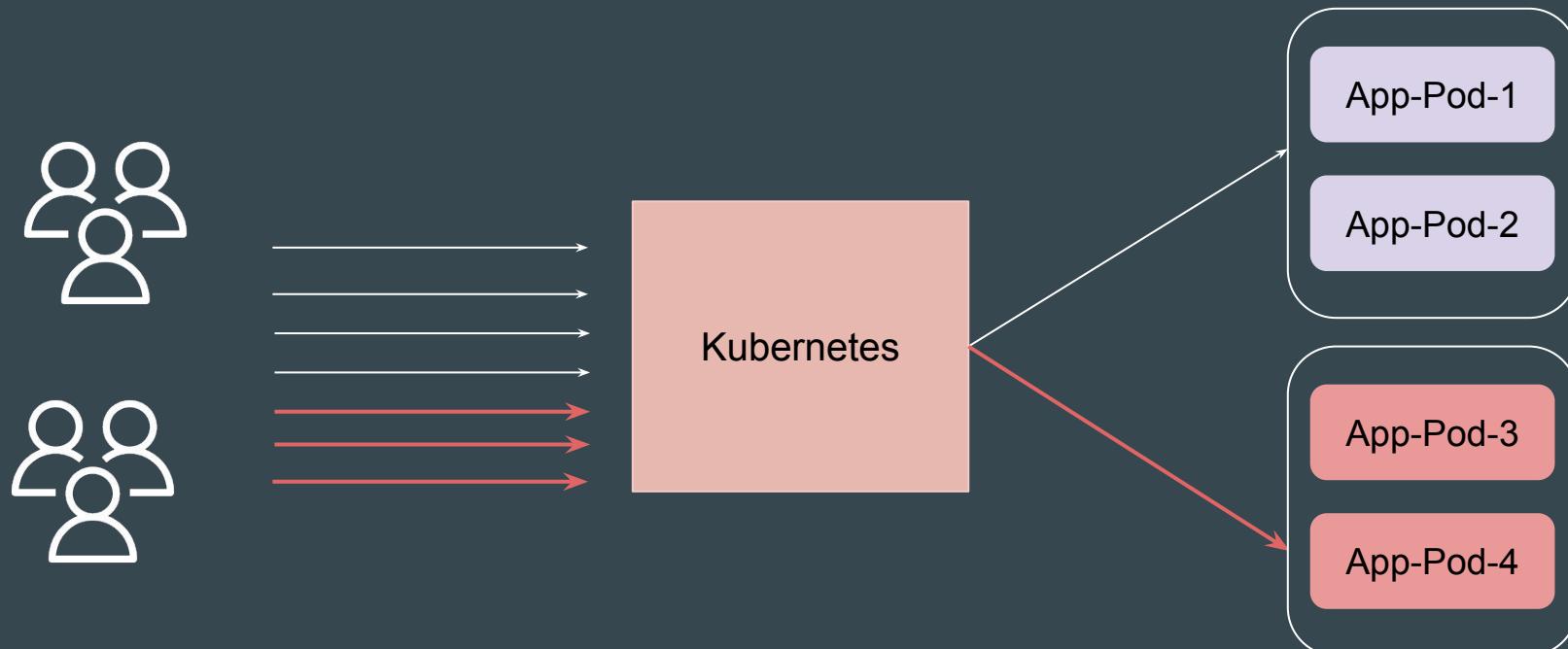
# Setting the Base

Workloads in a Kubernetes cluster are not always static; they **fluctuate based on user demand**, traffic spikes, or computational requirements.



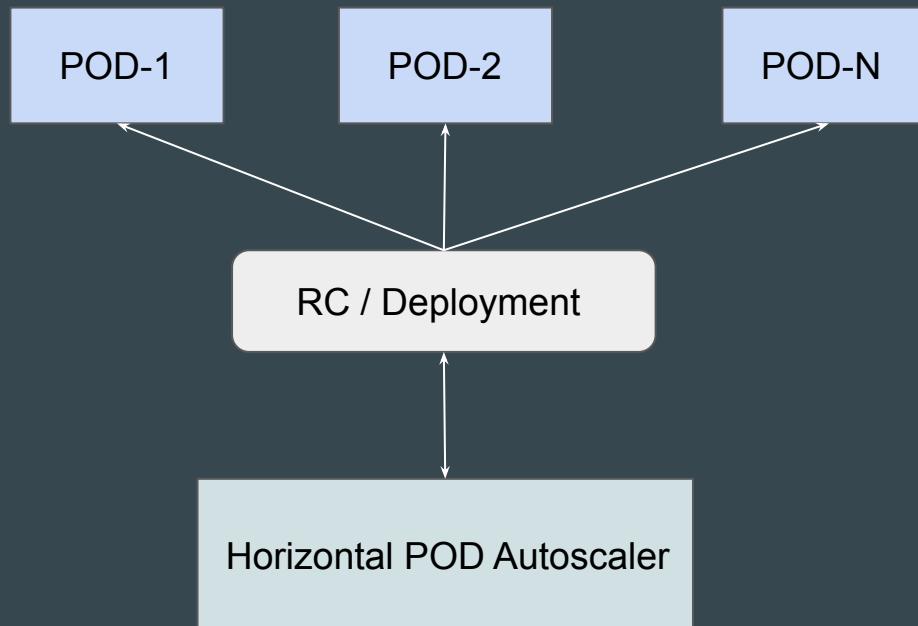
# Introducing HPA

Horizontal Pod AutoScaler (HPA) automatically adjusts the number of running pods in a deployment based on observed CPU utilization, memory usage, or custom metrics.



# The Problem that HPA Solves

1. Automatically scaling out (adding pods) when resource usage increases
2. Automatically scaling in (removing pods) when resource usage decreases



# HPA Workflow

kubectl get hpa php-apache --watch						
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	1	59s
php-apache	Deployment/php-apache	cpu: 380%/50%	1	3	1	90s
php-apache	Deployment/php-apache	cpu: 380%/50%	1	3	3	106s
php-apache	Deployment/php-apache	cpu: 902%/50%	1	3	3	2m1s
php-apache	Deployment/php-apache	cpu: 305%/50%	1	3	3	2m16s
php-apache	Deployment/php-apache	cpu: 323%/50%	1	3	3	2m31s
php-apache	Deployment/php-apache	cpu: 7%/50%	1	3	3	2m46s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	3	3m1s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	3	7m31s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	1	7m46s

## Points to Note

HPA does not apply to objects that cannot be scaled, like Daemonsets.

# **HPA - Stabilization Window**

# Understanding the Challenge

Temporary metric drops can create rapid scale down of Pods which might cause service instability.

Stabilization Window can adds delay before Pods are downscaled.

root@k8s:~# kubectl get hpa php-apache --watch						
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	1	59s
php-apache	Deployment/php-apache	cpu: 380%/50%	1	3	1	90s
php-apache	Deployment/php-apache	cpu: 380%/50%	1	3	3	106s
php-apache	Deployment/php-apache	cpu: 902%/50%	1	3	3	2m1s
php-apache	Deployment/php-apache	cpu: 305%/50%	1	3	3	2m16s
php-apache	Deployment/php-apache	cpu: 323%/50%	1	3	3	2m31s
php-apache	Deployment/php-apache	cpu: 7%/50%	1	3	3	2m46s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	3	3m1s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	3	7m31s
php-apache	Deployment/php-apache	cpu: 2%/50%	1	3	1	7m46s

# Setting the Base

Stabilization Window configuration tells HPA to wait for a certain period before scaling down.

Default Stabilization Window	Description
Scale Up	0 (no-delay)
Scale Down	300 seconds (5 minutes delay)

```
behavior:  
  scaleDown:  
    stabilizationWindowSeconds: 300  
  scaleUp:  
    stabilizationWindowSeconds: 0
```

# After Stabilization Windows of 60 seconds (scaleDown)

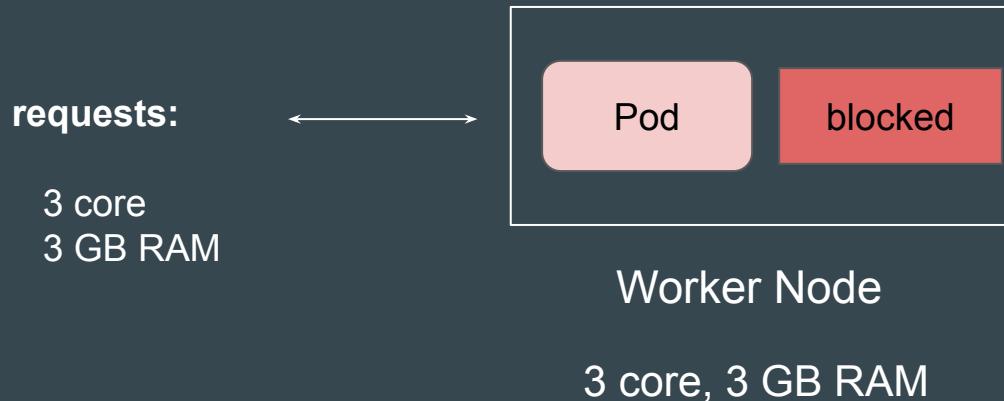
C:\kplabs-k8s>kubectl get hpa cpu-hpa --watch						
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
cpu-hpa	Deployment/php-apache	cpu: 2%/50%	1	5	1	38s
cpu-hpa	Deployment/php-apache	cpu: 840%/50%	1	5	1	75s
cpu-hpa	Deployment/php-apache	cpu: 1700%/50%	1	5	5	90s
cpu-hpa	Deployment/php-apache	cpu: 1698%/50%	1	5	5	105s
cpu-hpa	Deployment/php-apache	cpu: 712%/50%	1	5	5	2m
cpu-hpa	Deployment/php-apache	cpu: 2%/50%	1	5	5	2m30s
cpu-hpa	Deployment/php-apache	cpu: 2%/50%	1	5	5	3m15s
cpu-hpa	Deployment/php-apache	cpu: 2%/50%	1	5	1	3m30s

# **Vertical Pod Auto-Scaler**

# Understanding the Challenge

If the Kubernetes Requests and Limits are set too low, your app might crash; if too high, you waste resources.

Even if your application is idle and using very little CPU or memory, the requested amount is blocked off and cannot be used to schedule other pods onto that node



# Setting the Base

The **Vertical Pod Autoscaler** (VPA) is a Kubernetes component that automatically adjusts the CPU and memory requests for containers running within your pods.

```
Recommendation:  
Container Recommendations:  
  Container Name: nginx  
  Lower Bound:  
    Cpu:      25m  
    Memory:   262144k  
  Target:  
    Cpu:      25m  
    Memory:   262144k  
  Uncapped Target:  
    Cpu:      25m  
    Memory:   262144k  
  Upper Bound:  
    Cpu:      25m  
    Memory:   262144k
```

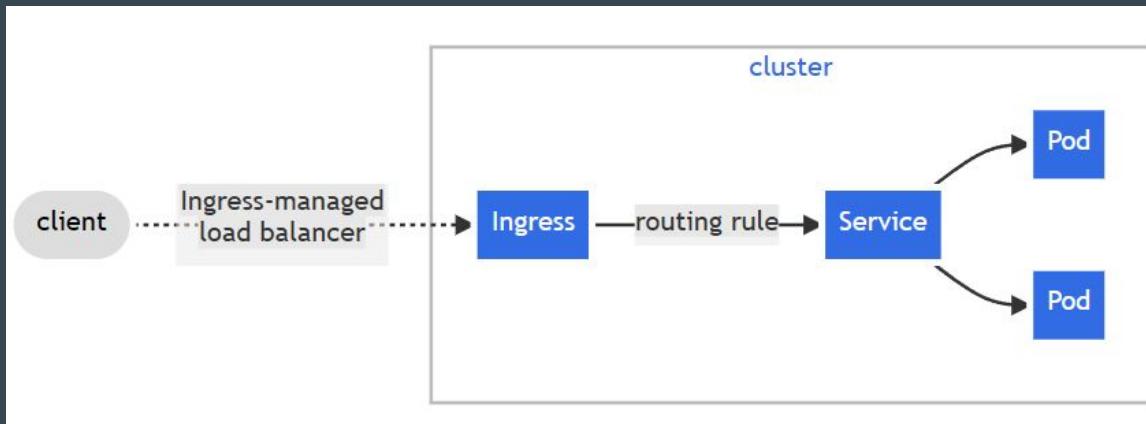
# Update Modes for VPA

<b>updateMode</b>	<b>Description</b>
Off	VPA only provides recommendations; no changes are applied.
Initial	VPA only assigns resource requests on pod creation and never changes them later.
Auto	VPA actively applies recommendations by evicting and restarting pods.

# **Gateway API**

# Revising Ingress

A Kubernetes ingress is an API object used to manage external user access to services running in a Kubernetes cluster.



# Limitations of Ingress

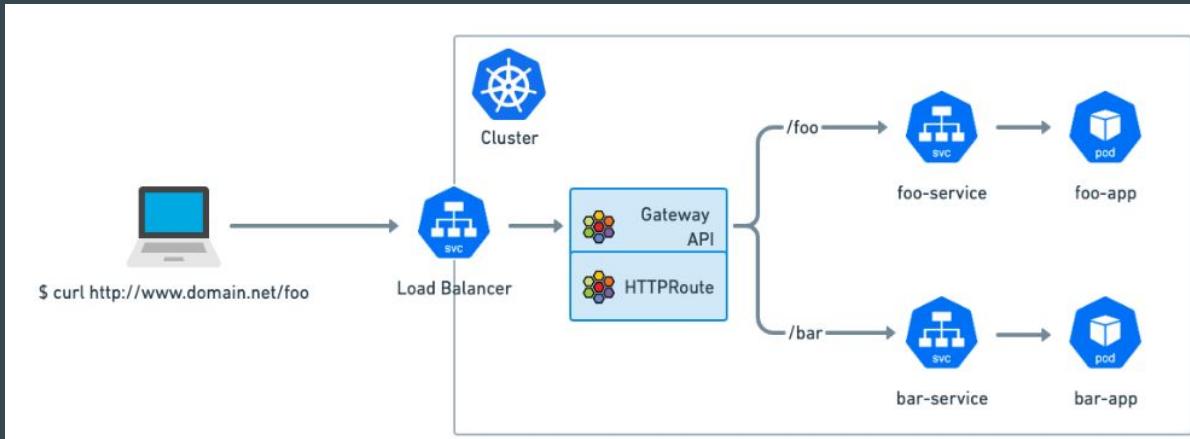
The Ingress API mainly handles HTTP and HTTPS traffic.

It lacks native support for advanced traffic management features needed in production scenarios

# Introducing Gateway API

To solve the limitations around Ingress, the Gateway API project was created.

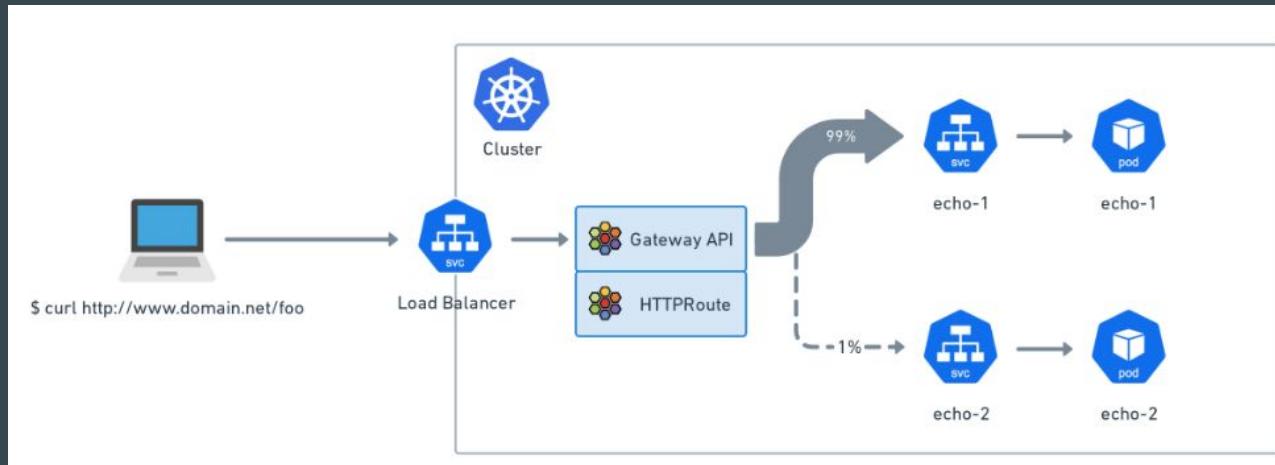
Gateway API supports both L4 and L7 protocols including TCP, UDP, HTTP, gRPC, and more are under consideration.



# HTTP Traffic Splitting / Weighting

HTTP traffic splitting is the process of sending incoming traffic to multiple backend services, based on predefined weights or other criteria.

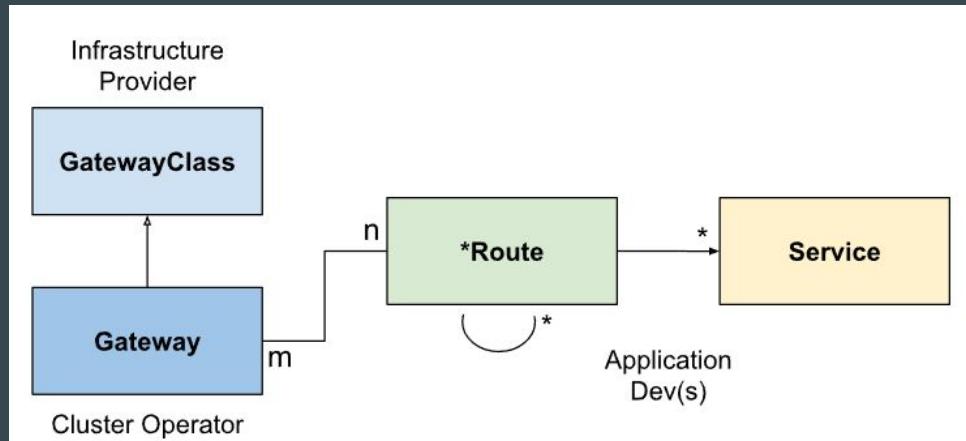
As part of the route rules, you can set appropriate weights for backends.

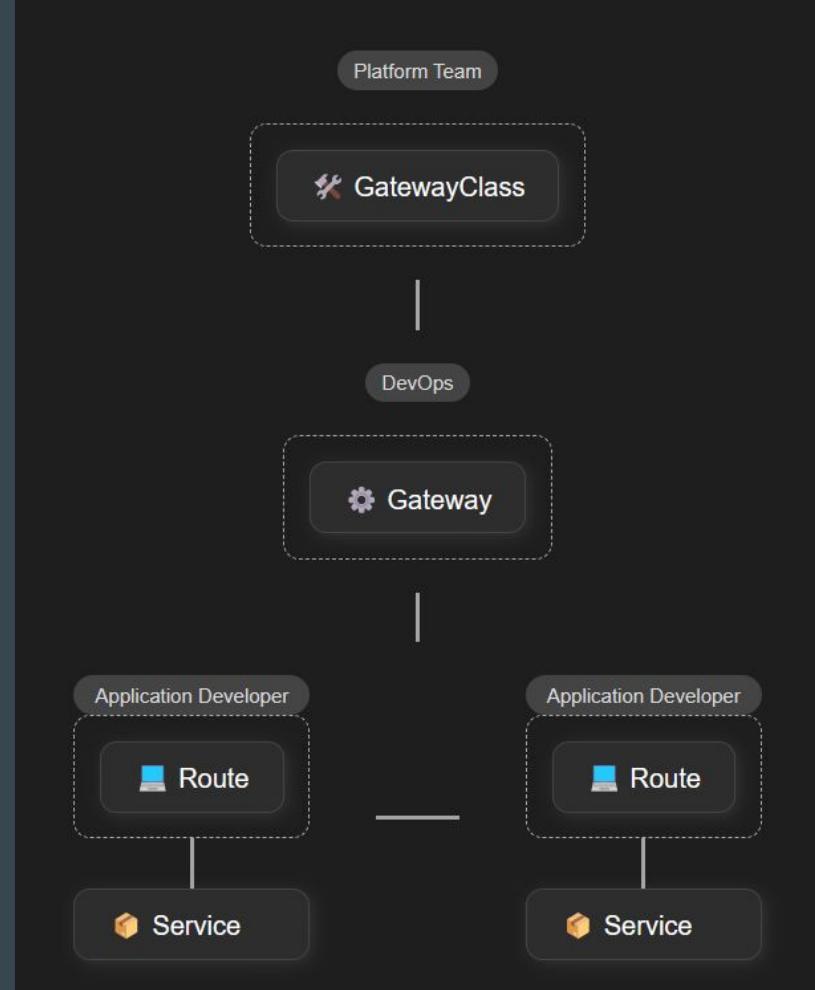


# **Structure - Gateway API**

# Understanding the Structure

The Gateway API includes `GatewayClass`, `Gateway`, and protocol-specific `Route`





# 1 - Gateway Class

Gateways can be implemented by different controllers.

GatewayClass specifies the controller used to implement the Gateway API resources.

```
C:\kplabs-k8s>kubectl get gatewayclass
NAME      CONTROLLER          ACCEPTED   AGE
nginx    gateway.nginx.org/gateway-controller  True       123m
```

## 2 - Gateway

A Gateway describes an instance of traffic handling infrastructure.

For example, a Gateway may represent a cloud load balancer or an in-cluster proxy server that is configured to accept HTTP traffic.

```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: Gateway
metadata:
  name: nginx-gateway
  namespace: default
spec:
  gatewayClassName: nginx
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```

## 3 - HTTPRoute

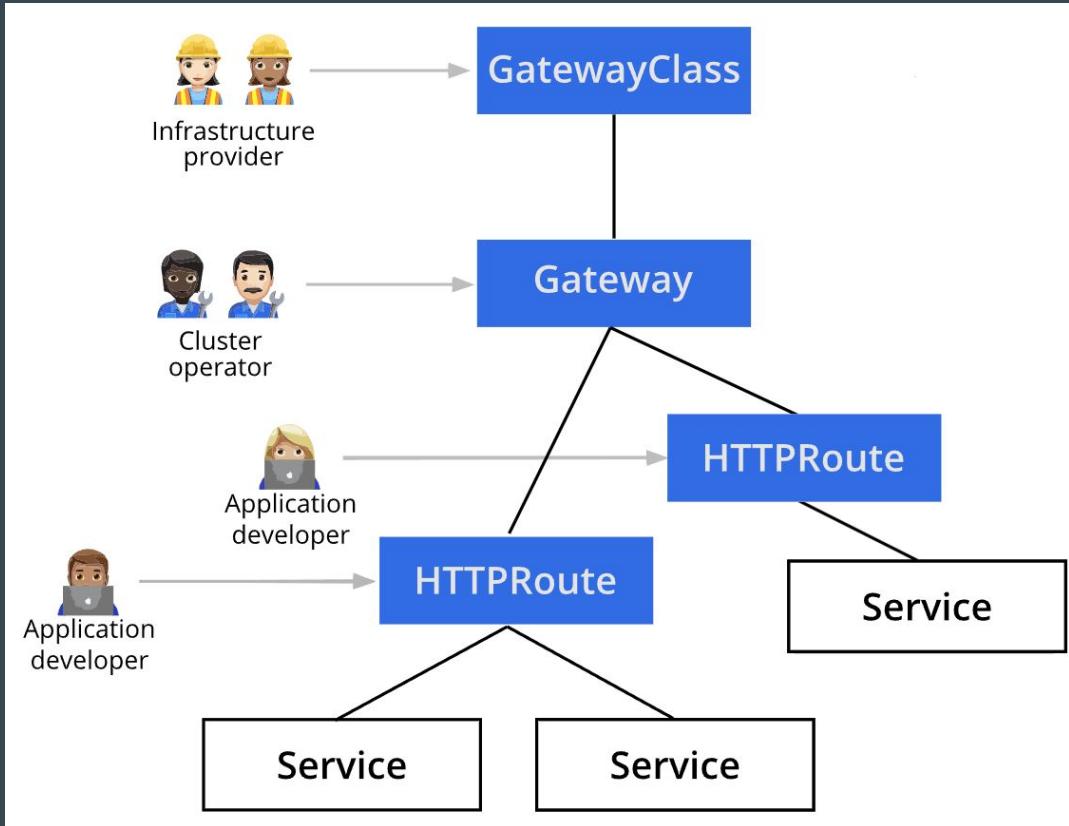
The HTTPRoute kind specifies routing behavior of HTTP requests from a Gateway listener to backend network endpoints.

```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: apache-route
spec:
  parentRefs:
  - name: nginx-gateway
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
  backendRefs:
  - name: apache-service
    port: 80
```

# 4 - Gateway Controllers

A Gateway Controller is the **implementation component** that watches and processes Gateway API resources.

- Acnodal EPIC
- Airlock Microgateway
- Amazon Elastic Kubernetes Service (GA)
- Apache APISIX (beta)
- Avi Kubernetes Operator
- Azure Application Gateway for Containers (GA)
- Cilium (beta)
- Contour (GA)
- Easegress (GA)
- Emissary-Ingress (Ambassador API Gateway) (alpha)
- Envoy Gateway (GA)
- Flomesh Service Mesh (beta)
- Gloo Gateway (GA)
- Google Kubernetes Engine (GA)
- HAProxy Ingress (alpha)
- HAProxy Kubernetes Ingress Controller (GA)
- HashiCorp Consul
- Istio (GA)



# Relationship between Components

1. A cluster admin creates a `GatewayClass` that references a specific `GatewayController`
2. A team creates a `Gateway` resource referencing that `GatewayClass`
3. The `GatewayController` watches for these resources and implements the actual networking setup
4. Routes (like `HTTPRoute`) are attached to the `Gateway` to define traffic rules

# Routes

Object	OSI Layer	Routing Discriminator	TLS Support	Purpose
HTTPRoute	Layer 7	Anything in the HTTP Protocol	Terminated only	HTTP and HTTPS Routing
TLSRoute	Somewhere between layer 4 and 7	SNI or other TLS properties	Passthrough or Terminated	Routing of TLS protocols including HTTPS where inspection of the HTTP stream is not required.
TCPRoute	Layer 4	destination port	Passthrough or Terminated	Allows for forwarding of a TCP stream from the Listener to the Backends
UDPRoute	Layer 4	destination port	None	Allows for forwarding of a UDP stream from the Listener to the Backends.
GRPCRoute	Layer 7	Anything in the gRPC Protocol	Terminated only	gRPC Routing over HTTP/2 and HTTP/2 cleartext

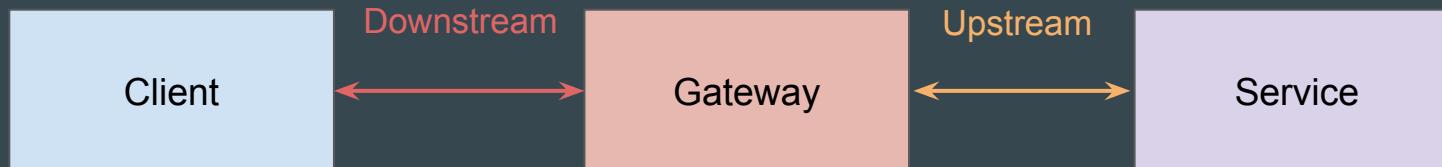
# **TLS - Gateway API**

# Setting the Base

For Gateways, there are two connections involved:

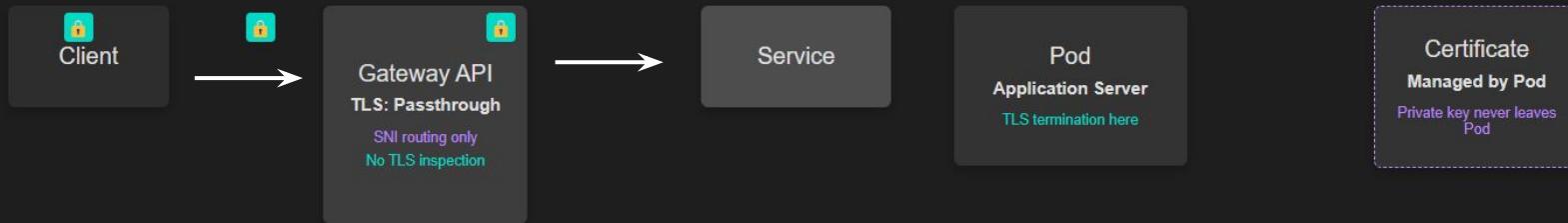
**downstream**: This is the connection between the client and the Gateway.

**upstream**: This is the connection between the Gateway and backend resources

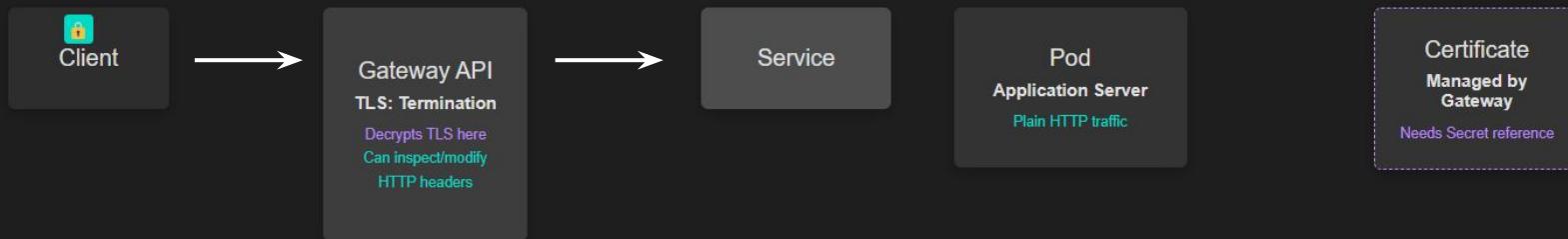


# Two Important Modes

## TLS Passthrough Mode

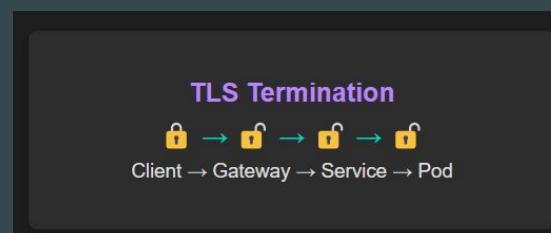
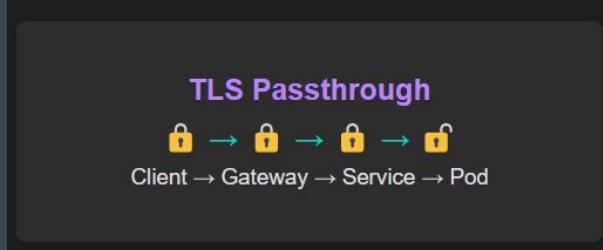


## TLS Terminate Mode



# Side-by-Side Comparison

Feature	TLS Passthrough	TLS Terminate
TLS Decryption Point	Backend Pod	Gateway
Certificate Management	Managed by backend	Managed by Gateway
Content Inspection	Not possible at Gateway	Possible at Gateway
Routing Capability	Limited to SNI-based routing	Full HTTP header-based routing
Request Modification	Not possible at Gateway	Possible at Gateway



# Supported Route Types

Listener Protocol	TLS Mode	Route Type Supported
TLS	Passthrough	TLSRoute
TLS	Terminate	TCPRoute
HTTPS	Terminate	HTTPRoute

# Gateway Configuration - TLS Terminate Mode

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-tls-gateway
  namespace: default
spec:
  gatewayClassName: nginx
  listeners:
    - name: https
      protocol: HTTPS
      port: 443
      tls:
        mode: Terminate
        certificateRefs:
          - kind: Secret
            name: example-tls
```

# **Mapping Ingress API features to Gateway API Features**

# 1 - Entry Points for Ingress

Every Ingress resource has two implicit entry points -- one for HTTP and the other for HTTPS traffic. An Ingress controller provides those entry points.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app-ingress
spec:
  rules:
  - host: kplabs.internal
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-app-service
            port:
              number: 80
```

# Entry Points for Gateway API

In Gateway API, entry points must be explicitly defined in a Gateway resource.

For example, if you want the data plane to handle HTTP traffic on port 80, you need to define a listener for that traffic.

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: nginx
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```

## 2 - TLS Termination

The Ingress resource supports TLS termination via the TLS section, where the TLS certificate and key are stored in a Secret.

In Gateway API, TLS termination is a property of the Gateway listener, and similarly to the Ingress, a TLS certificate and key are also stored in a Secret.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - example.com
    secretName: my-tls-secret
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
```

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: nginx
  listeners:
  - name: https
    protocol: HTTPS
    port: 443
    hostname: example.com
  tls:
    mode: Terminate
    certificateRefs:
    - name: my-tls-secret
```

# 3 - Routing Rules

We directly add Routing rules as part of the Ingress Resource.

In Gateway API, the rules can be added as part of HTTPRoute, gRPC route tc.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - example.com
    secretName: my-tls-secret
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
```

```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: apache-route
  namespace: default
spec:
  parentRefs:
  - name: nginx-gateway
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
  backendRefs:
  - name: apache-service
    port: 80
```

# Point to Note

When migrating from Ingress to Gateway API, you have the option of performing a manual migration or utilizing automated tools like Ingress2gateway.

## Ingress to Gateway

Ingress2gateway helps translate Ingress and provider-specific resources (CRDs) to Gateway API resources.

Ingress2gateway is managed by the [Gateway API](#) SIG-Network subproject.

### Scope

Ingress2gateway is primarily focused on translating Ingress and provider-specific resources(CRDs) to Gateway API resources. Widely used provider-specific annotations and/or CRDs *may* still not be supported. Please refer to [supported providers](#) for the current supported providers and their documentation. Contributions for provider-specific annotations and/or CRDs support are mostly welcomed as long as they can be translated to [Gateway API](#) directly.

Note: Ingress2gateway is not intended to copy annotations from Ingress to Gateway API.

# **Custom Resource Definitions**

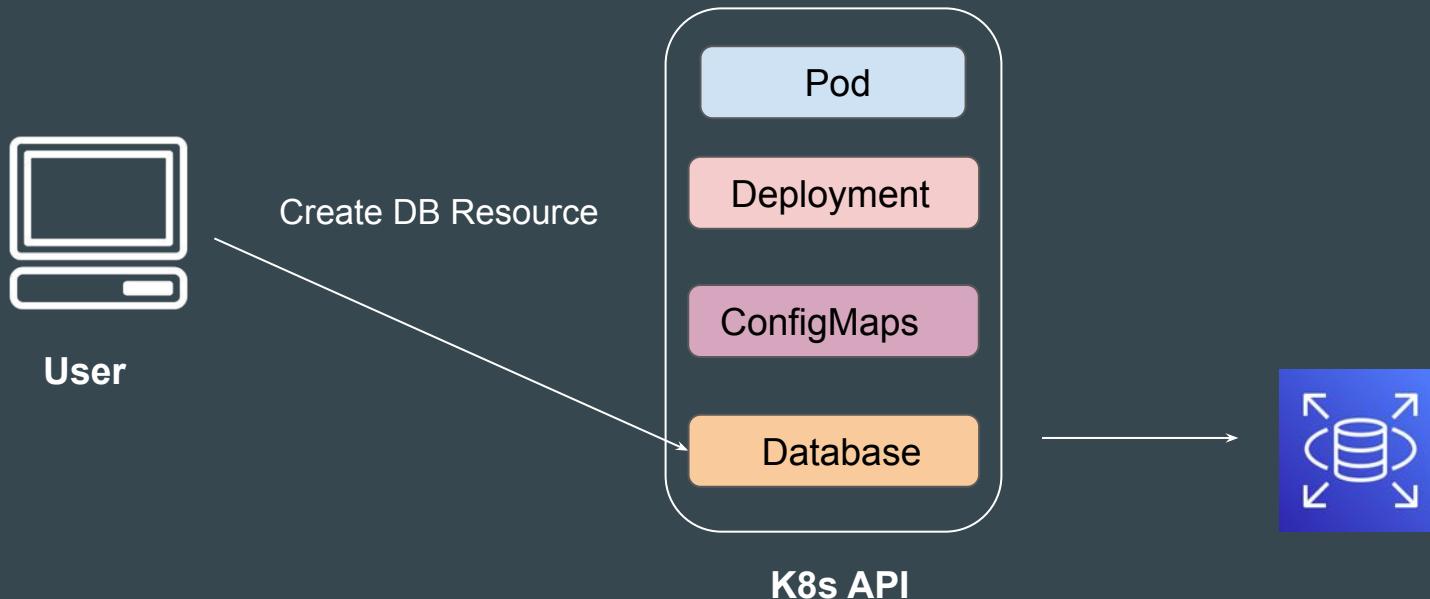
# Setting the Base

A resource is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind.

C:\>kubectl api-resources	SHORTNAMES	APIVERSION	NAMESPACE	KIND
NAME				
bindings		v1	true	Binding
componentstatuses	cs	v1	false	ComponentStatus
configmaps	cm	v1	true	ConfigMap
endpoints	ep	v1	true	Endpoints
events	ev	v1	true	Event
limitranges	limits	v1	true	LimitRange
namespaces	ns	v1	false	Namespace
nodes	no	v1	false	Node
persistentvolumeclaims	pvc	v1	true	PersistentVolumeClaim
persistentvolumes	pv	v1	false	PersistentVolume
pods	po	v1	true	Pod
podtemplates		v1	true	PodTemplate
replicationcontrollers	rc	v1	true	ReplicationController
resourcequotas	quota	v1	true	ResourceQuota
secrets		v1	true	Secret
serviceaccounts	sa	v1	true	ServiceAccount
services	svc	v1	true	Service

# Basics of CRDs

A **Custom Resource Definition** allows you to **extend the Kubernetes API** by defining your own custom resource types.



# Step 1 - Create Custom Resource Definition

To create a CRD, you need to create a file, that defines your object kinds.

Applying a CRD into the cluster makes the Kubernetes API server to serve the specified custom resource.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: databases.kplabs.internal
spec:
  group: kplabs.internal
  names:
    kind: Database
    listKind: DatabaseList
    plural: databases
    singular: database
  scope: Namespaced
  versions:
    - name: v1
      served: true
      storage: true
      schema:
```

## Step 2 - Create Custom Objects

After the CustomResourceDefinition object has been created, you can create custom objects.

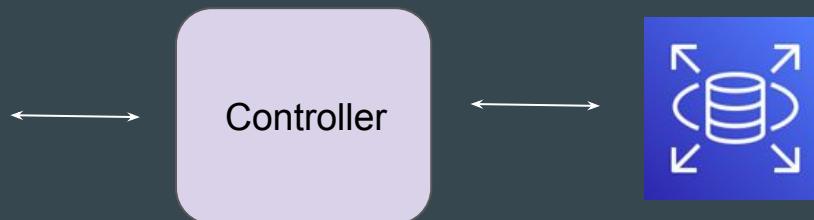
```
apiVersion: kplabs.internal/v1
kind: Database
metadata:
  name: my-database
spec:
  name: test-db
  replicas: 3
```

# Importance of Custom Controller

A controller tracks at least one Kubernetes resource type.

These objects have a spec field that represents the desired state.

```
apiVersion: kplabs.internal/v1
kind: Database
metadata:
  name: my-database
spec:
  name: test-db
  replicas: 3
```



Database

# Helm - Skipping CRDs

## Setting the Base

With the Helm 3, there is now a **special directory called crds** that you can create in your chart to hold your CRDs.

If the CRD already exists, it will be skipped with a warning. If you wish to skip the CRD installation step, you can pass the **--skip-crds** flag.

# Point to Note

Depending on the chart, there can be a different way to skip crds.

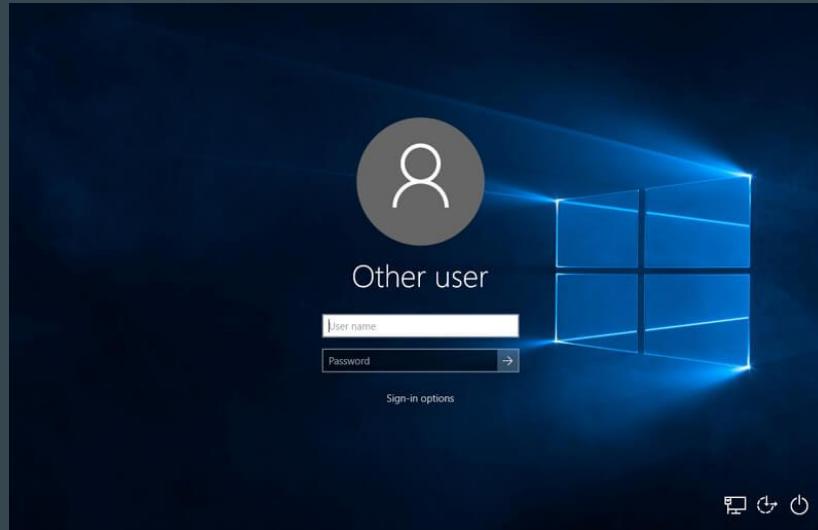
```
{{- if .Values.crds.install }}  
apiVersion: apiextensions.k8s.io/v1  
kind: CustomResourceDefinition  
metadata:  
  annotations:  
    {{- if .Values.crds.keep }}  
    "helm.sh/resource-policy": keep  
    {{- end }}  
    {{- with .Values.crds.annotations }}  
      {{- toYaml . | nindent 4 }}  
    {{- end }}  
  labels:  
    app.kubernetes.io/name: applications.argoproj.io  
    app.kubernetes.io/part-of: argocd  
    {{- with .Values.crds.additionalLabels }}  
      {{- toYaml . | nindent 4}}  
    {{- end }}  
  name: applications.argoproj.io
```

```
root@kubeadm:~# helm template my-argo-cd argo/argo-cd --set crds.install=false --version 7.8.23 > argo-no-crd.yaml  
root@kubeadm:~# cat argo-no-crd.yaml | grep CustomResourceDefinition  
root@kubeadm:~# |
```

# **Authentication in Kubernetes**

# Basics of Authentication

Authentication is the process of verifying a user's identity before granting them access to a system or resource



# Accessing Resources in Kubernetes

To access resources in Kubernetes cluster, we have to authenticate first.



# Analogy of AWS

In AWS, you can authenticate using multiple set of methods.

1. Username and Passwords.
2. Access Key and Secret Keys

```
C:\>aws ec2 describe-security-groups
{
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": [
                        {
                            "GroupId": "sg-01aa5110c343f107d",
                            "UserId": "430118823531"
                        }
                    ]
                },
            ]
        }
    ]
}
```



## Point to Note - Kubernetes

Kubernetes **does not manage the user accounts natively**.

Normal users cannot be added to a cluster through an API call



`kubectl create user alice`



# Authentication in Kubernetes

Kubernetes supports several authentication methods such as:

Client Certificates, Static Token Authentication, Service Account Tokens etc



# Example 1 - Static Token File

The API server reads bearer tokens from a file provided.

The token file is a csv file with a minimum of 3 columns: token, user name, user uid

```
root@control-plane:~# cat /root/token.csv  
Dem0Passw0rd#,bob,01,admins
```



```
[Service]  
ExecStart=/usr/local/bin/kube-apiserver --advertise-address=165.22.212.16 --etcd-cafile=/root/certificates/ca.crt --etcd-cert  
file=/root/certificates/etcd.crt --etcd-keyfile=/root/certificates/etcd.key --etcd-servers=https://127.0.0.1:2379 --service-a  
ccount-key-file=/root/certificates/service-account.crt --service-cluster-ip-range=10.0.0.0/24 --service-account-signing-key-f  
ile=/root/certificates/service-account.key --service-account-issuer=https://127.0.0.1:6443 --token-auth-file /root/token.csv
```

# Example 2 - X509 Certificates

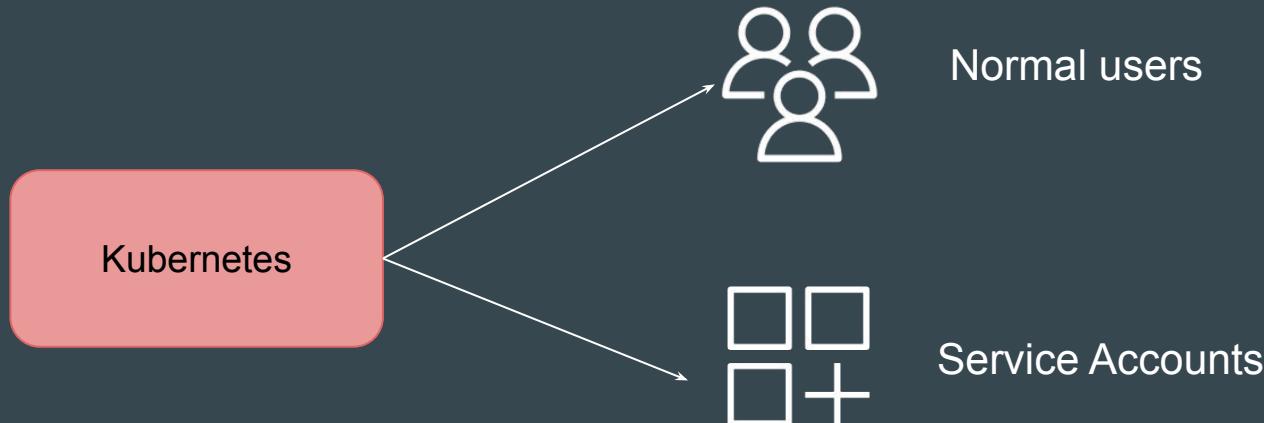
Uses the client certificates for authentication.

```
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR
    QVFTEJRQXdGVEVUTUJFR0ExVUUJKQXhNS2EzVm1aWEp1WlhSbGN6QVGdzB5TlRBeE1qVxdN
    kJBb1RGbXQxWW1WaFpHMDZZMngxYzNSbGNpMWhaRzFwYm5NeEdUQvHcZ05WQkFNVEVhdDFZb
    FRVUFBNelCRHdBd2dnRUtBb01CQVFER01aWjkKNnQ0ZC90NGhUNWpxb2p6SjRBT2JObnRTQe
    rTGtXaXoveCszTwdyREJwNGNheDjqS0ZTU0dNbU5udUZnT1NMR21GaS9yK3IyR2MyUUJaN3N
    RGtOQWVzd1BIVUVQcWc1RFQ5MU50eXpiUhdjN0UwdkEwODgKQuDyV3FKMWhTN291VmNhTmE0
    2srLzFydgpubGIwM1lrT1EwWUsvMU9jSEI3UEZQZ21Wb1AvWVeRk2xqNEgyWWpzUkE4UmFTT
    FHalZqQlVNQTRHQTfVZER3RUIvd1FFQXdJRm9EQVRCZ05WSFNRUREQUsKQmdnckJnRUZCUw
    O1NMQkNtVktP0xadwpGWUdtWxc1aGRRWkxNQTBHQ1NxR1NJYjNEUUVQCQ3dVQUE0SUJBUE
    TDIxdXN0UWjtZ3pubUN6cndyQXpwdHZwLzFORUY0MkpTVjBpem8veW1JWFZEVmJJMw8KSVI0
    nVabEdYZDYxbUNZTkwyckdpE9BZgp0L0R3OUZVcVdtcnVsaUp1cEJOMHNBeVZ4dUUxSDNYL
    1xMDNIUjdTUUY2NGV5SHB4SUt4QnoyNWJ3cVhETytEdnJjR3piUE5EcW9WUGFidWJZQzBKdU
    vNktzd15zOULqMEDBcFFER1YzUGdoejU5Ci0tLS0tRU5EIENFU1RJRk1DQVRFLS0tLS0K
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkfFURSLRVktLS0tLQpNSU1Fb2d
    aW0rdnF3RXpvVS9Sa1J5TFNwCnJZUVVicCs1cCtJdk1Qb21RZDRQTXBDNUZvcy84ZnR6SUt3
```

# Categories of Users

Kubernetes Clusters have **two categories of users**:

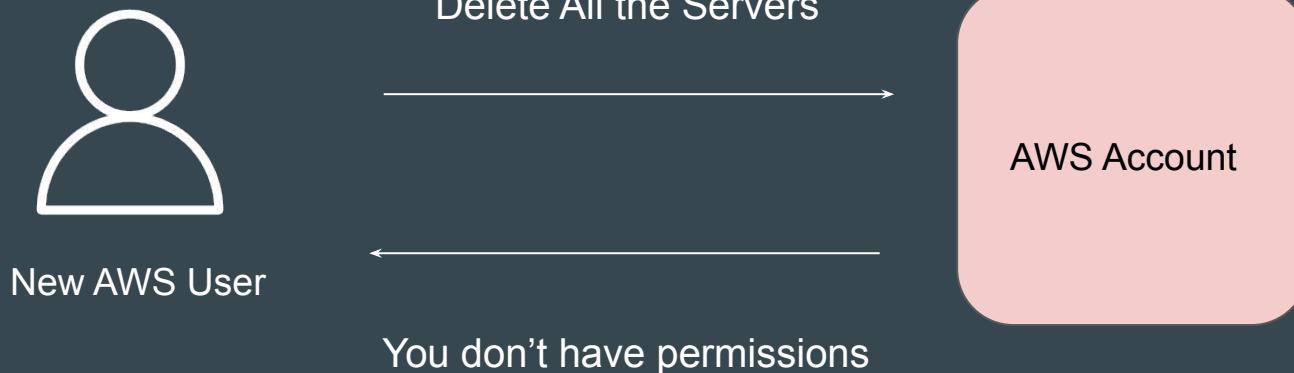
1. Normal Users (for humans)
2. Service Accounts (for apps)



# **Authorization**

# Basics of Authorization

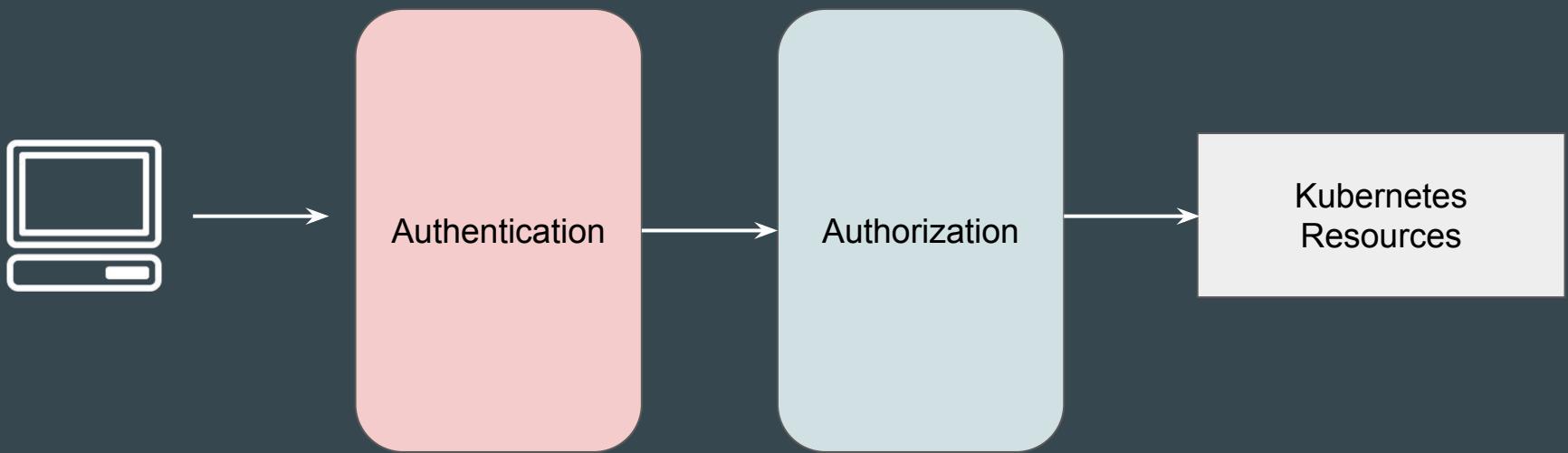
Authorization is the process of determining what an authenticated user or entity is allowed to do



# Authorization in Kubernetes

Kubernetes authorization takes place following authentication.

Usually, a client making a request must be authenticated (logged in) before its request can be allowed.



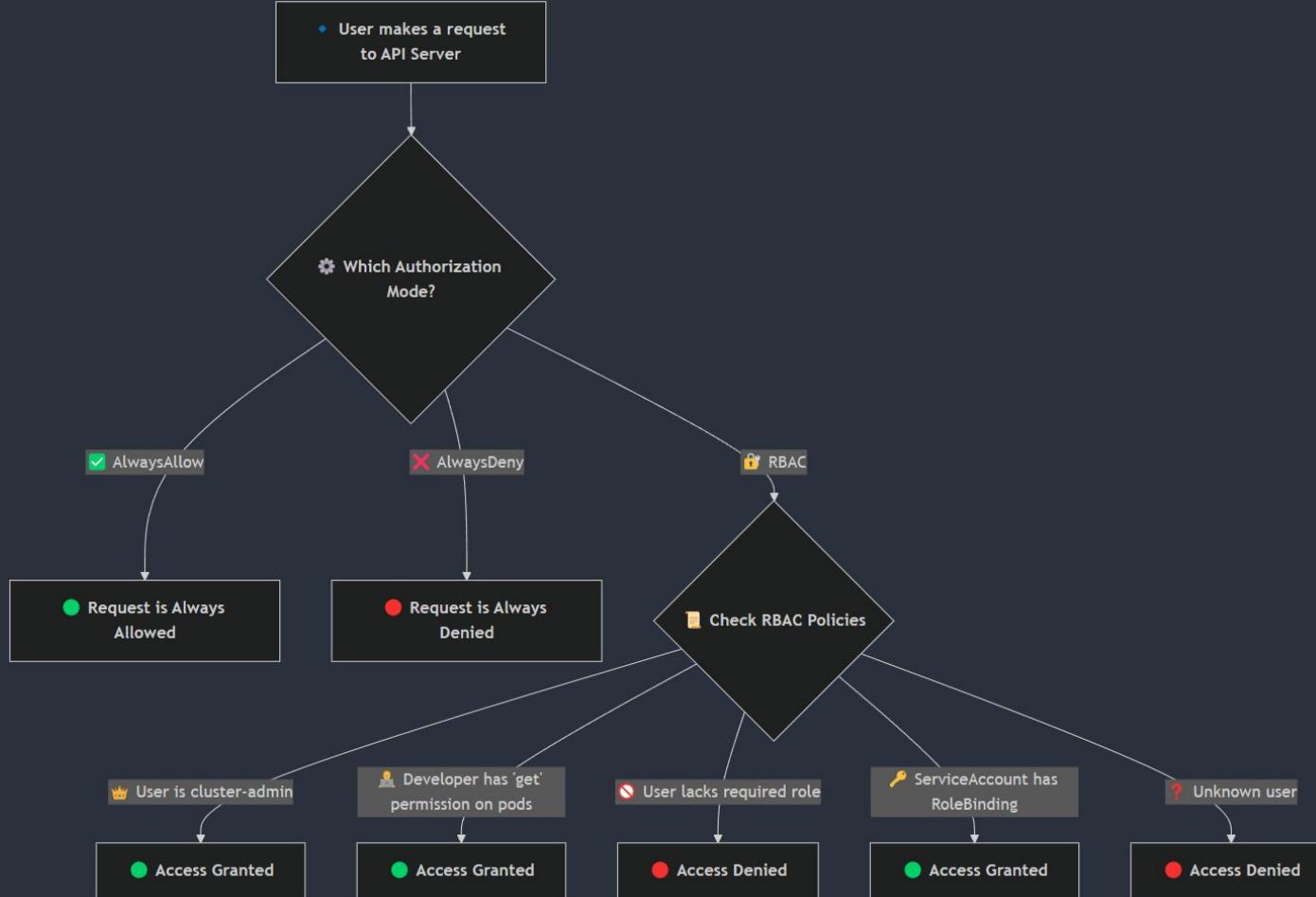
# Authorization Modes

The Kubernetes API server may authorize a request using one of several authorization modes. Some of these include:

Authorization Mode	Description
AlwaysAllow	<p>This mode allows all requests, which brings security risks.</p> <p>Use this authorization mode only for testing.</p>
AlwaysDeny	<p>This mode blocks all requests.</p> <p>Use this authorization mode only for testing.</p>
RBAC	<p>Defines set of permissions based on which access is granted.</p> <p>Recommended for Production.</p>

## Point to Note

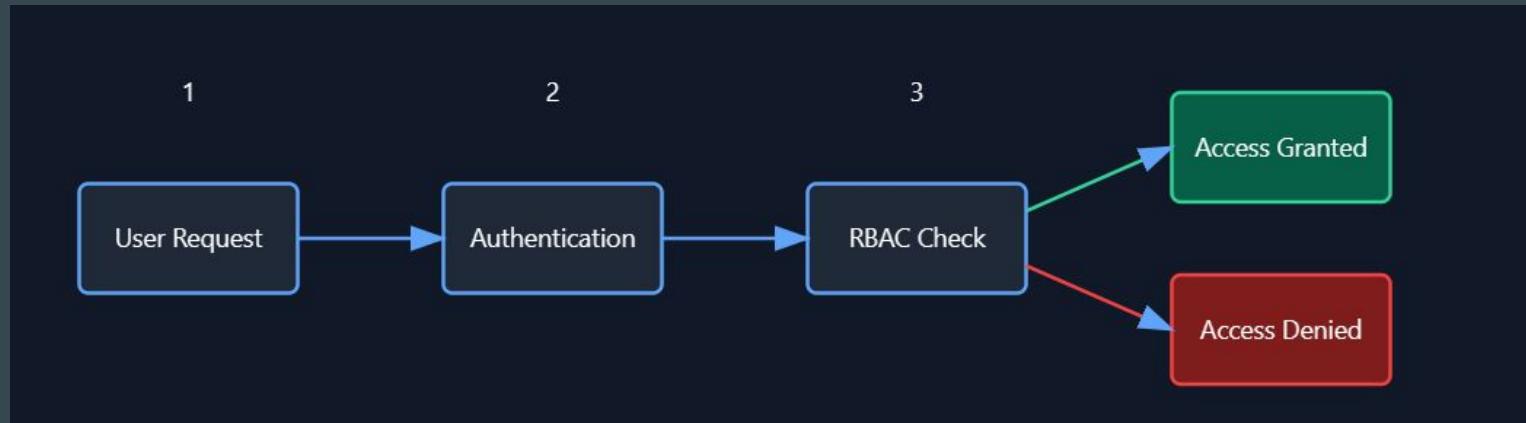
In Kubernetes, if the authorization mode is not explicitly defined in the API server configuration, the default mode used is AlwaysAllow.



# **Role-Based Access Control (RBAC)**

# Setting the Base

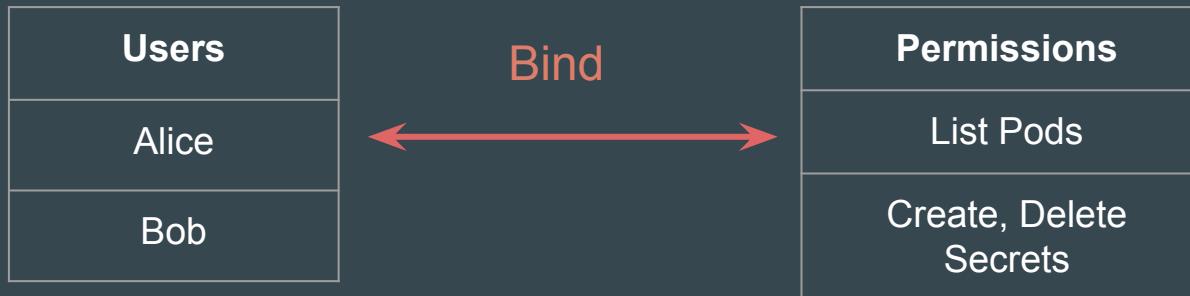
RBAC allows us to control what actions users and service accounts can perform on resources within your cluster.



# Basic Workflow

In the below diagram, we have a list of users in Table 1 and list of permissions in Table 2.

We have to bind these together for users to get the defined permissions.

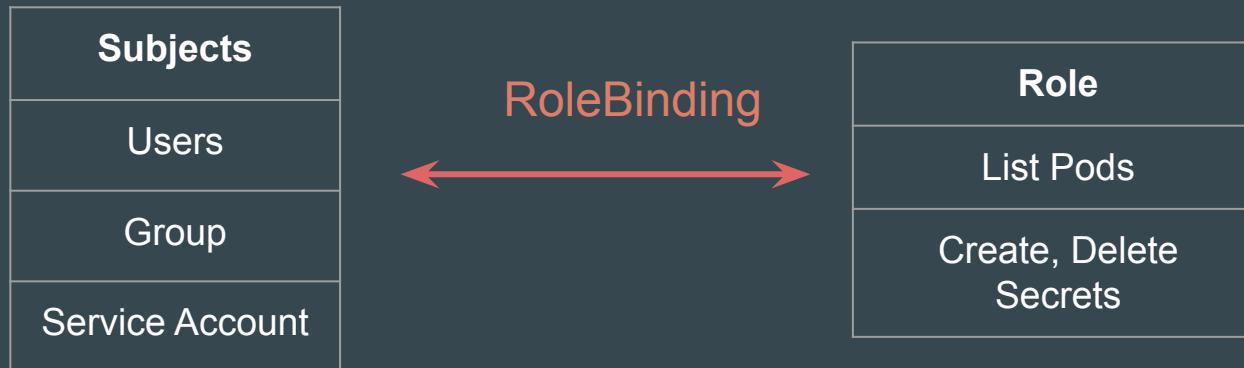


# 3 Important Concepts

Role defines a set of permissions.

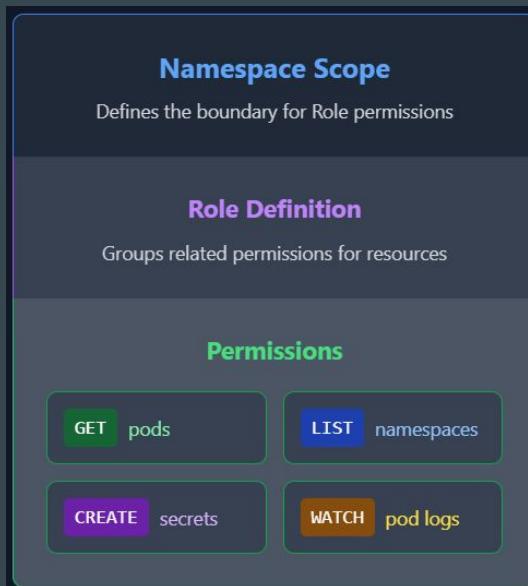
Subjects can be user, groups, service account.

RoleBinding ties the permission defined in the role to subjects like Users.



# Introducing Roles

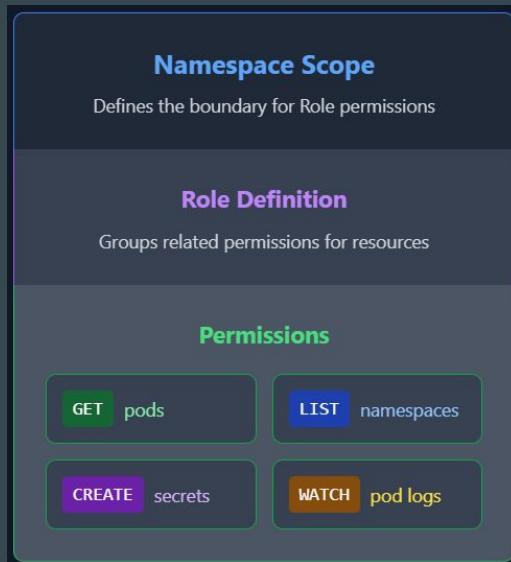
A Role always sets permissions within a particular namespace.



# Introducing RoleBinding

RoleBinding associates a Role with a user, group, or service account within a specific namespace.

It grants the defined permissions to the subjects in that namespace.



RoleBinding



# ClusterRole and ClusterRoleBinding

Similar to Role and RoleBinding, but the **main difference** is that the permissions granted by a **ClusterRole** apply across all namespaces in the cluster. ClusterRoleBinding connects ClusterRole to Subjects.



ClusterRoleBinding



# **Practical - Role and RoleBinding**

# Basic Structure of Role Manifest

The following image represents the basic structure of the first part of a Role manifest file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
```

# Defining Rules in Role Manifest

The **rules** field is a list of policies that define the permissions granted by the Role.

Each rule specifies which actions (verbs) are allowed on which resources (API objects).

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list"]
```

# 1 - API Groups

apiGroups specify which API group the rule applies to.

Kubernetes APIs are categorized into different API groups.

API Groups	Description
"" (empty string)	Refers to the core API group (e.g., pods, services, configmaps etc).
apps	Refers to the apps API group (e.g., deployments, daemonsets,replicasets)
batch	Includes Jobs, CronJobs.
networking.k8s.io	Handles Ingress and Network Policies.

## 2 - Resources

This field specifies which Kubernetes resources the rule applies to.

These resources belong to the specified API group.

C:\>kubectl api-resources --api-group="apps"				
NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
controllerrevisions		apps/v1	true	ControllerRevision
daemonsets	ds	apps/v1	true	DaemonSet
deployments	deploy	apps/v1	true	Deployment
replicasets	rs	apps/v1	true	ReplicaSet
statefulsets	sts	apps/v1	true	StatefulSet

# 3 - Verbs

Verb specifies what actions (operations) are allowed on the specified resources.

Common Verbs	Description
get	Read a specific resource.
list	List all resources of that type.
create	Create a new resource.
delete	Modify an existing resource.
update	Remove a resource.
watch	Observe changes to a resource.

# Structure - RoleBinding

While defining RoleBinding, we have to define subjects and Role Reference.



# Generate Role Manifest File

```
C:\>kubectl create role pod-reader --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: pod-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
```

# Generate Role Binding Manifest File

```
C:\>kubectl create rolebinding pod-reader --role=pod-reader --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: pod-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-reader
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```

# **Practical - ClusterRole and ClusterRoleBinding**

# Structure of ClusterRole Manifest

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-read-only
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list"]
```

# Structure of ClusterRoleBinding Manifest

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pod-rolebinding
  namespace: default
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-read-only
  apiGroup: rbac.authorization.k8s.io
```

# Generate ClusterRole Manifest File

```
C:\>kubectl create clusterrole pod-read-only --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: pod-read-only
rules:
- apiGroups:
  - ""
    resources:
    - pods
    verbs:
    - list
```

# Generate ClusterRoleBinding Manifest File

```
C:\>kubectl create clusterrolebinding pod-read --clusterrole=pod-read-only --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: pod-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pod-read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```

---

# Asymmetric Key Encryption

Right Architecture is the Key

---

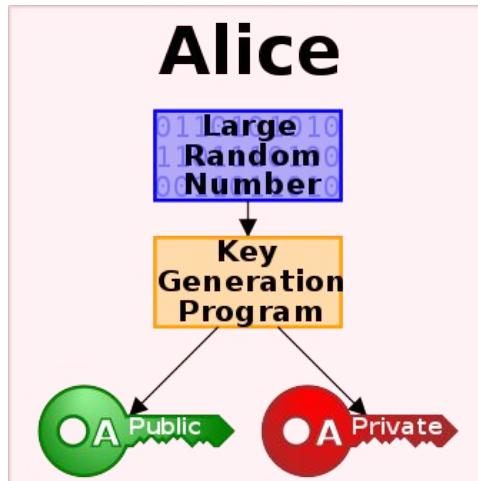
# Overview of Asymmetric Key Encryption

Asymmetric cryptography, uses public and private keys to encrypt and decrypt data.

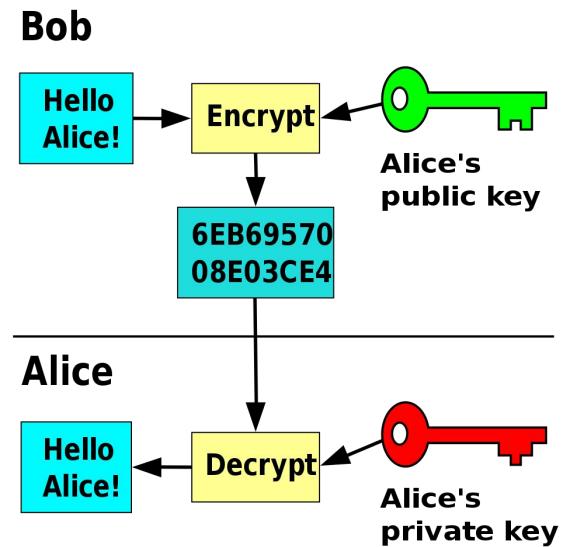
One key in the pair can be shared with everyone; it is called the public key. The other key in the pair is kept secret; it is called the private key.

Either of the keys can be used to encrypt a message; the opposite key from the one used to encrypt the message is used for decryption.

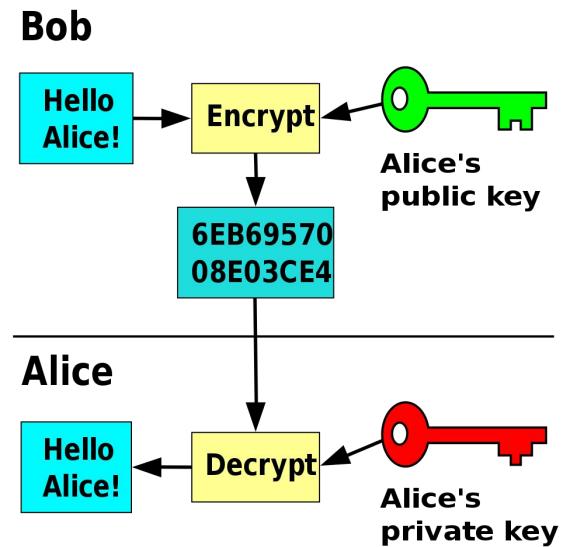
# Step 1: Generation of Keys



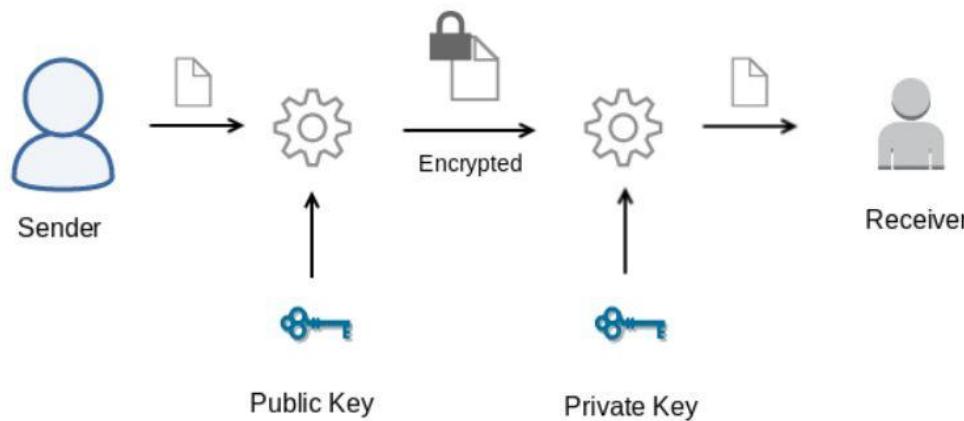
## Step 2: Encryption and Decryption



## Step 2: Encryption and Decryption

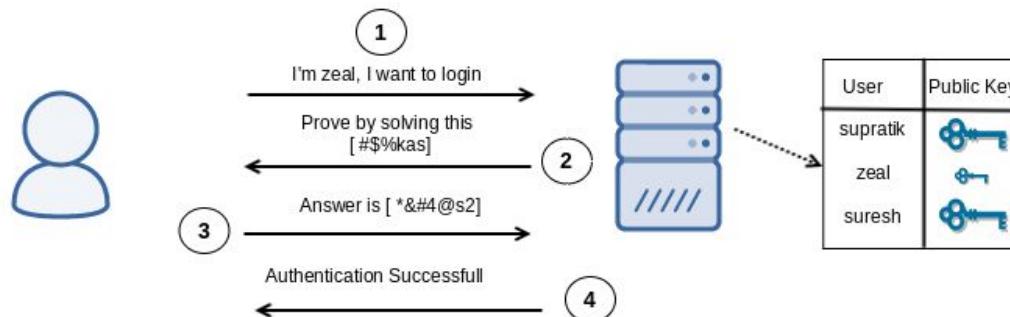


## Step 2: Encryption and Decryption



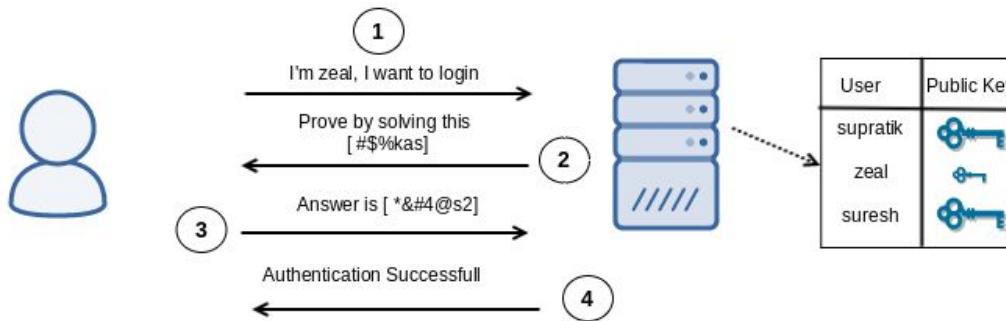
# Use-Case of Asymmetric Key Encryption - Step 1

User zeal wants to log in to the server. Since the server uses a public key authentication, instead of taking the password from the user, the server will verify if the User claiming to be zeal actually holds the right private key.



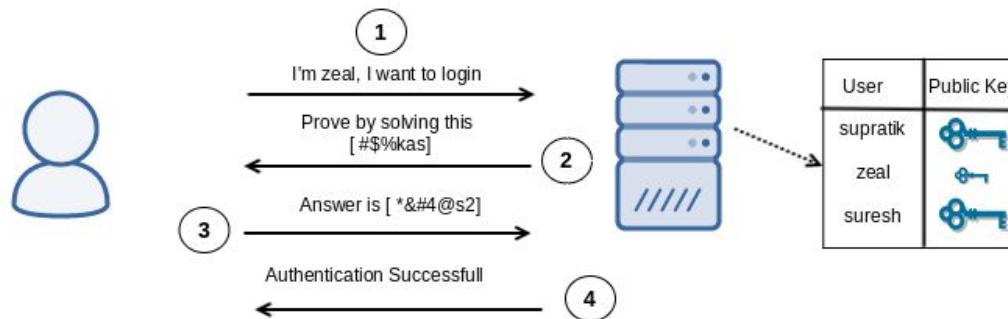
# Use-Case of Asymmetric Key Encryption - Step 2

The server creates a simple challenge,  $2+3=?$  and encrypts this challenge with the Public Key of the User and sends it back to the User. The challenge is sent in an encrypted format.



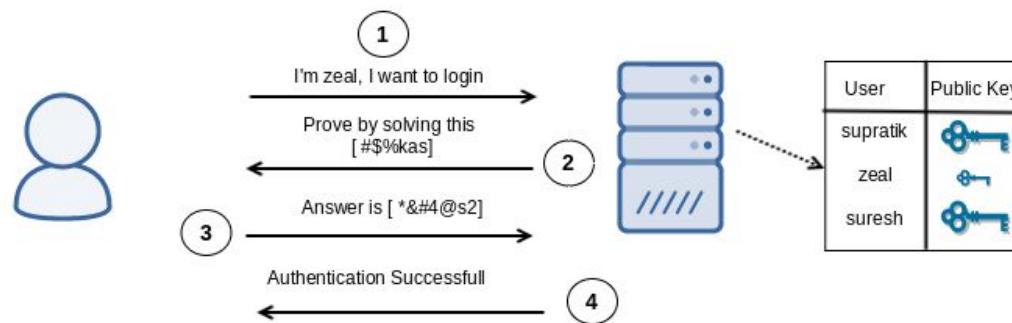
# Use-Case of Asymmetric Key Encryption - Step 3

Since the user zeal holds the associated private key, he will be able to decrypt the message and compute the answer, which would be 5. Then, he will encrypt the message with the private key and send it back to the server.



# Use-Case of Asymmetric Key Encryption - Step 4

The server decrypts the message with the user's Public Key and checks if the answer is correct. If yes, then the server will send an Authentication Successful message and the user will be able to log in.



# Protocols

Because of the advantage that it offers, Asymmetric key encryption is used by variety of protocols.

Some of these include:

- PGP
- SSH
- Bitcoin
- TLS
- S/MIME

---

# HTTPS

## Secure Communication

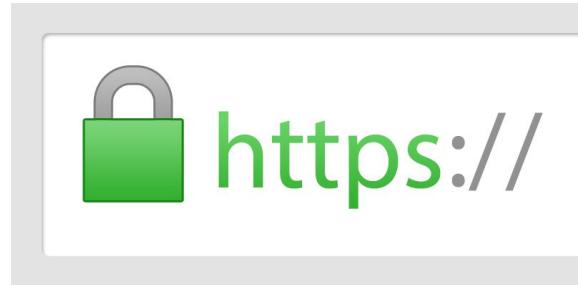
---

# Overview of HTTPS

HTTPS is an extension of HTTP.

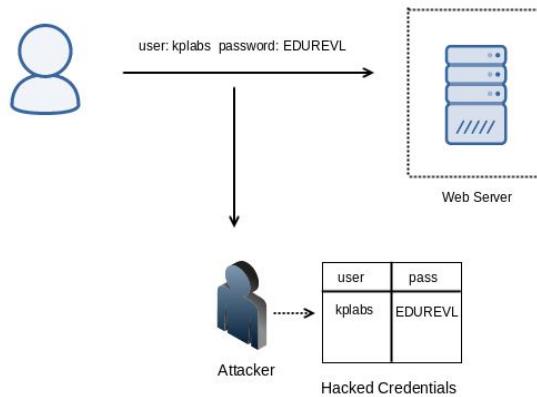
In HTTPS, the communication is encrypted using Transport Layer Security (TLS)

The protocol is therefore also often referred to as HTTP over TLS or HTTP over SSL.



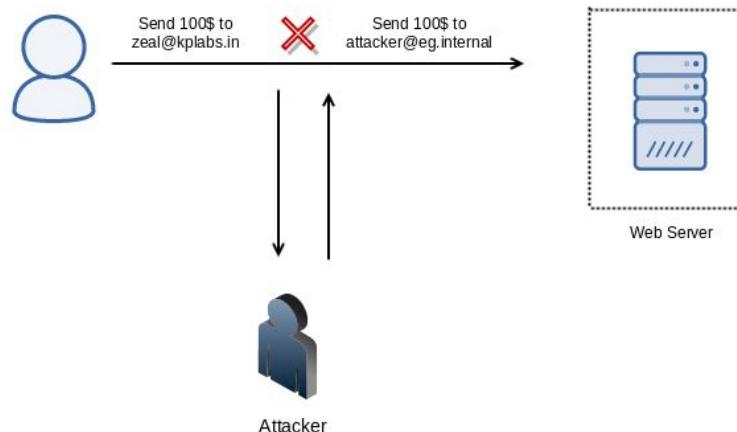
# Scenario 1: MITM Attacks

- User is sending their username and password in plaintext to a Web Server for authentication over a network.
- There is an Attacker sitting between them doing a MITM attack and storing all the credentials he finds over the network to a file:



## Scenario 2: MITM & Integrity Attacks

- Attacker changing the payment details while packets are in transit.



# Introduction to SSL/TLS

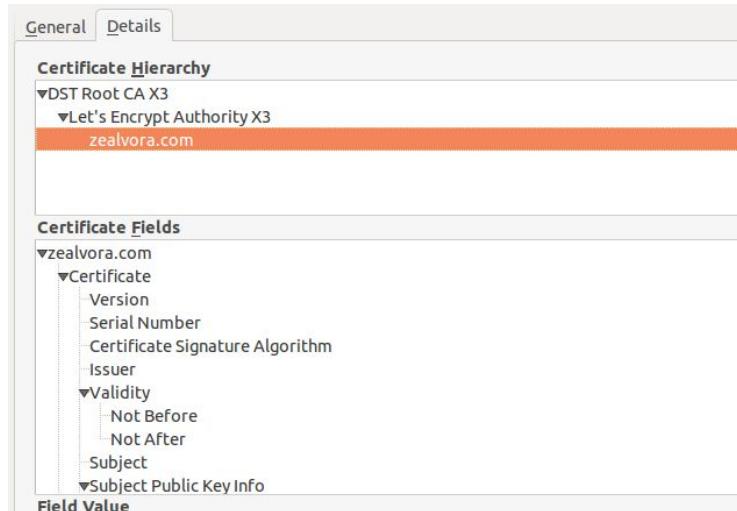
To avoid the previous two scenarios (and many more), various cryptographic standards were clubbed together to establish a secure communication over an untrusted network and they were known as SSL/TLS.

Protocol	Year
SSL 2.0	1995
SSL 3.0	1996
TLS 1.0	1999
TLS 1.1	2006
TLS 1.2	2008
TLS 1.3	2018

# Understanding it in easy way

Every website has a certificate (like a passport which is issued by a trusted entity).

Certificate has lot of details like domain name it is valid for, the public key, validity and others.



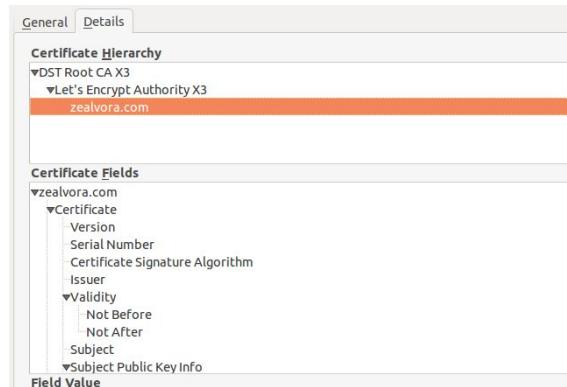
# Understanding it in easy way

Browser (clients) verifies if it trusts the certificate issuer.

It will verify all the details of the certificate.

It will take the public key and initiate a negotiation.

Asymmetric key encryption is used to generate a new temporary symmetric key which will be used for secure communication.



# Web Server Configuration

```
server {
    listen      80;
    server_name zealvora.com;
    return      301 https://$server_name$request_uri;
}

server {
    server_name zealvora.com;
    listen 443 default ssl;
    server_name zealvora.com;
    ssl_certificate /etc/letsencrypt/archive/zealvora.com/fullchain1.pem;
    ssl_certificate_key /etc/letsencrypt/archive/zealvora.com/privkey1.pem;

    location / {
        root /websites/zealvora/;
        include location-php;
        index index.php;
    }
    location ~ /.well-known {
        allow all;
    }
}
```

---

# Certificate Based Auth

Kubernetes Authentication

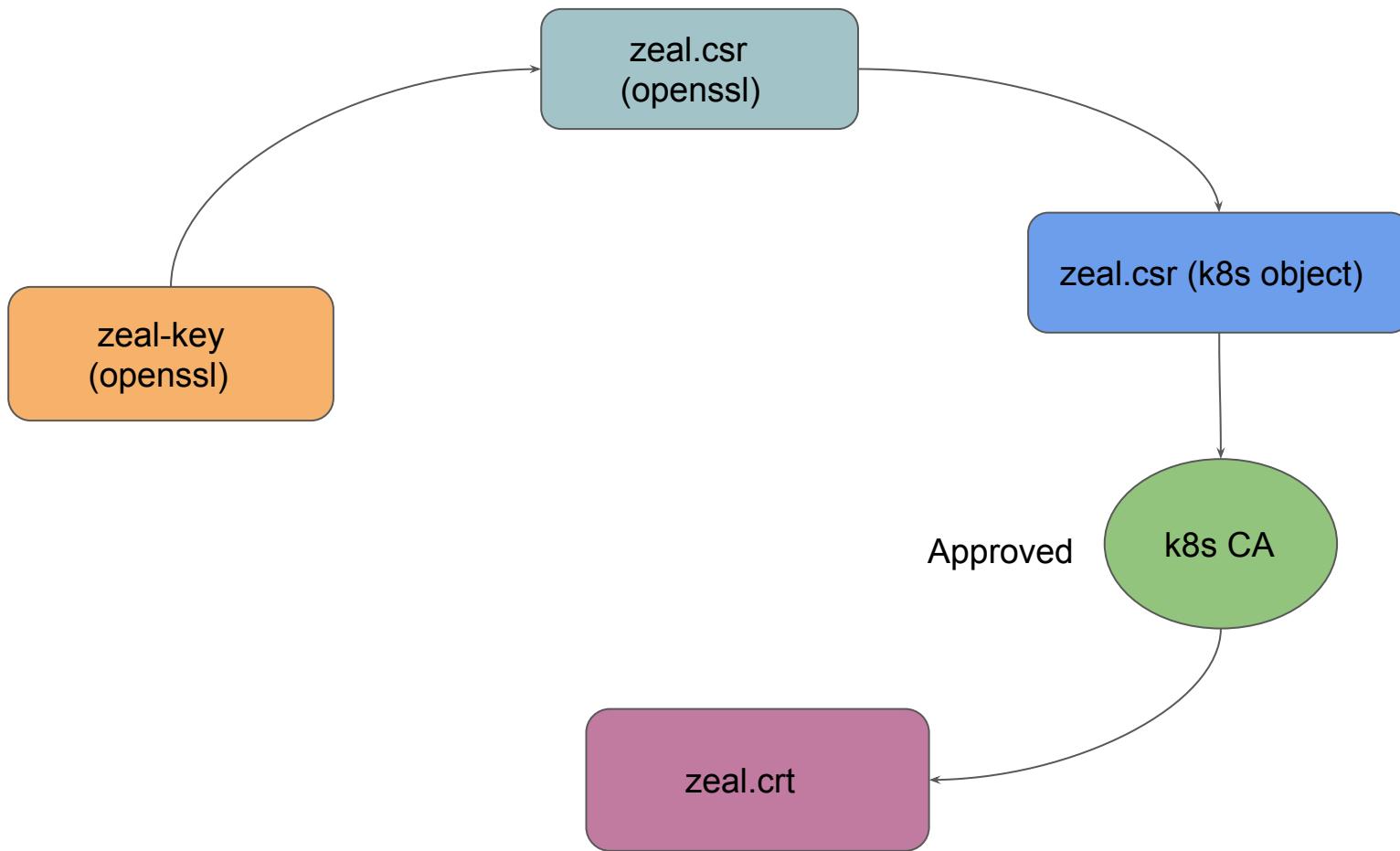
---

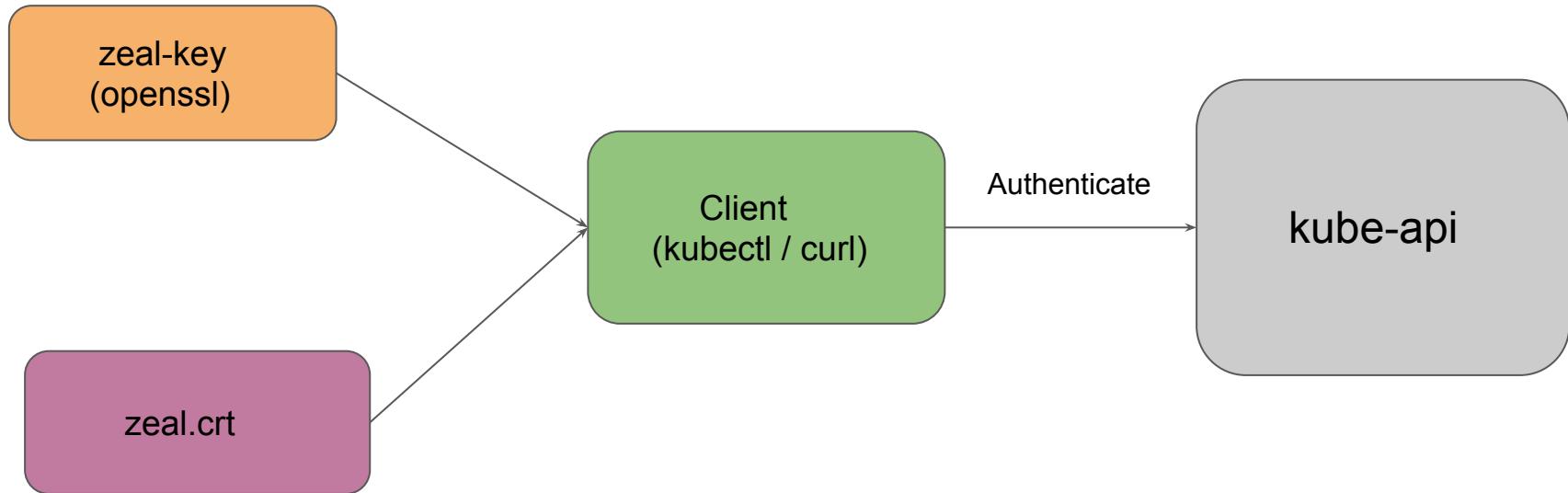
# Our Goal for Today

Our goal is to setup a authentication based on X509 Client certificates.

User A should be able to authenticate with Kubernetes Cluster with his certificates.







---

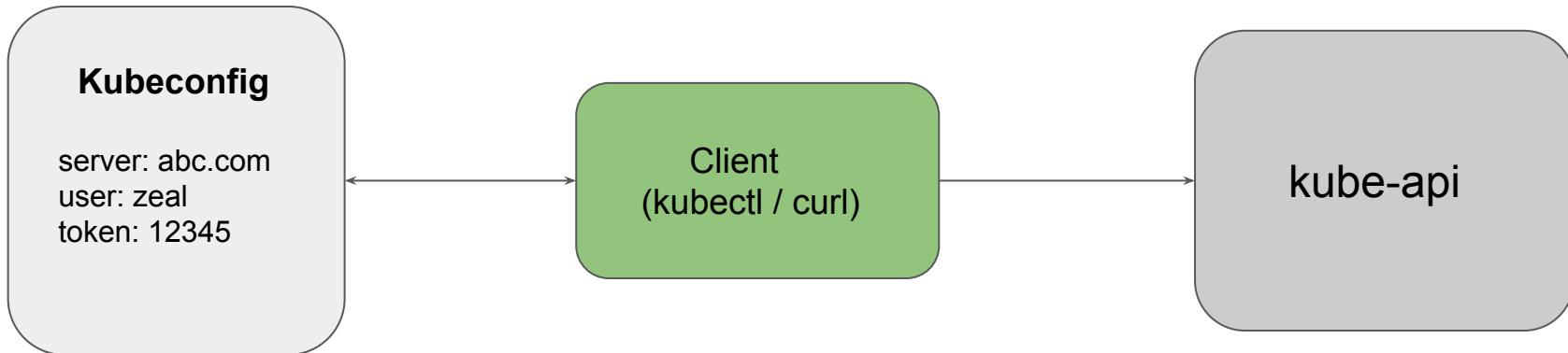
# Kubeconfig

## Security Aspect

---

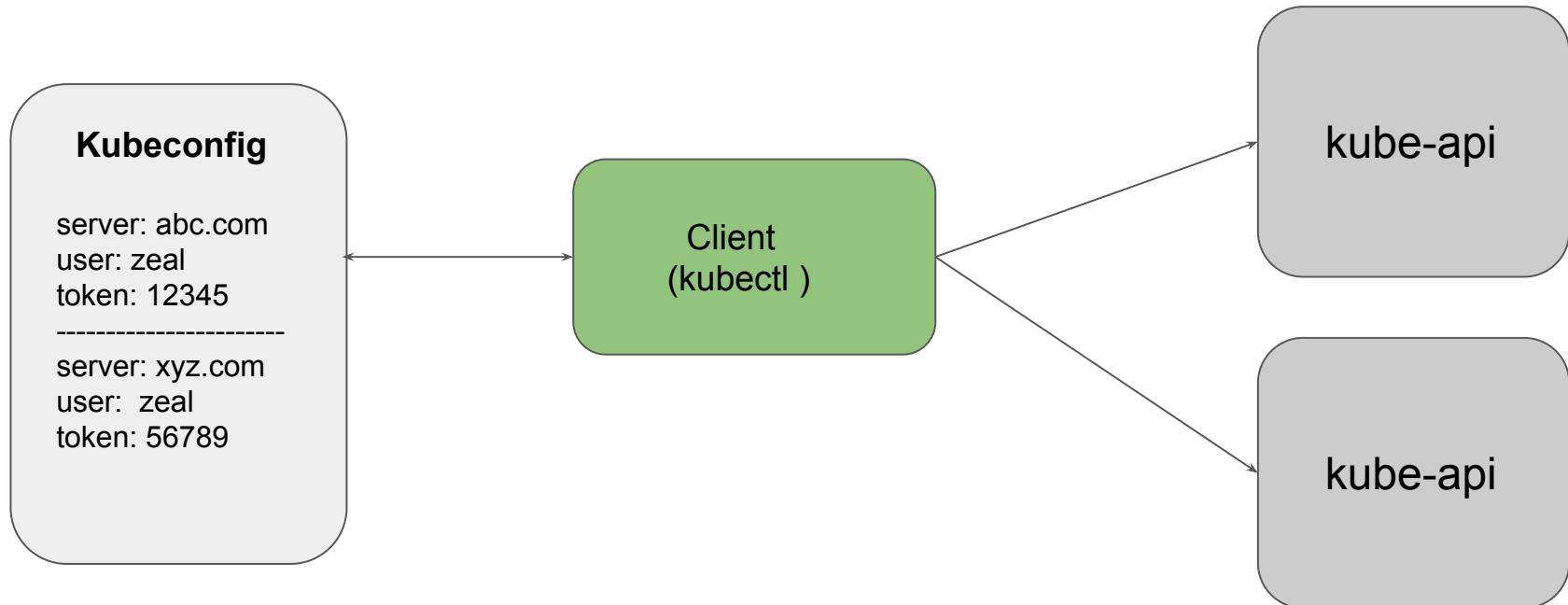
# Understanding Kubeconfig

Kubectl command uses kubeconfig files to find the information about the cluster, username, authentication mechanisms and others.



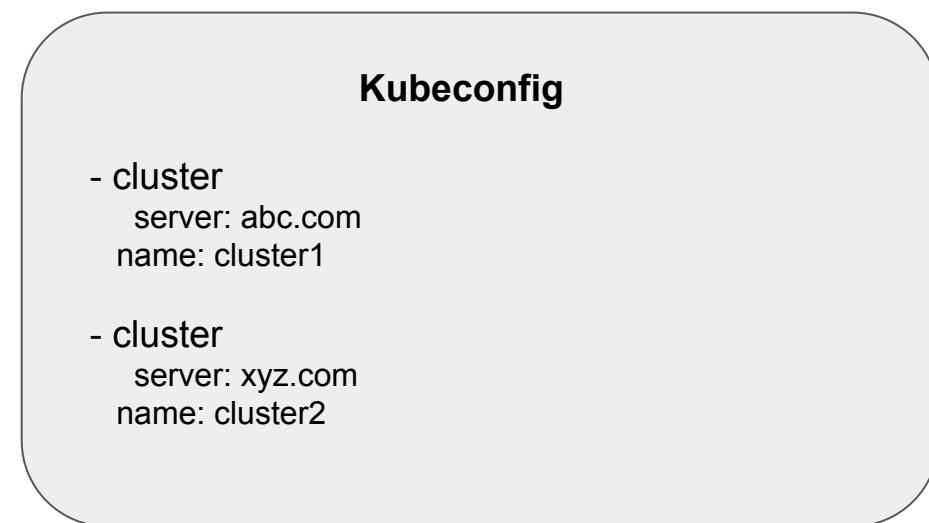
# Access to Multiple Clusters

Kubeconfig file can also have details associated with multiple kubernetes clusters.



# Step 1 - Understanding Clusters Field

Cluster field has details related to the URL of your Kubernetes Cluster and it's associated information.



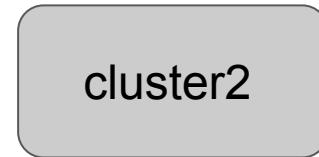
cluster1

cluster2

## Step 2 - Users Field

User field contains authentication specific information like username, passwords.

There can be different type of authentication mechanisms (user/pass, certificates, tokens etc)



# Step 3 - Context Field

Context groups information related to cluster, user and namespace.

## Kubeconfig

```
- cluster
  server: abc.com
  name: cluster1

- context:
  cluster: cluster1
  namespace: kplabs
  user: adminuser

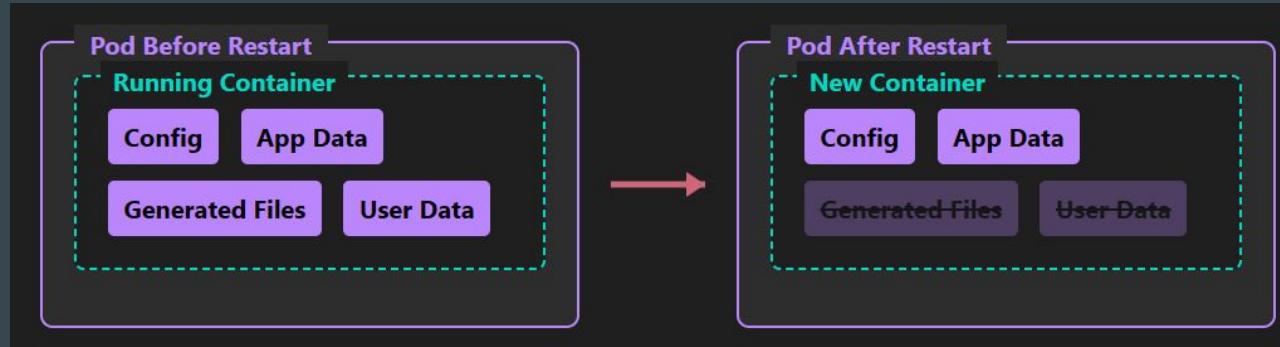
users:
- name: adminuser
  user:
    token: 12345
```

# **Volumes in Kubernetes**

# Understanding Challenge - State Persistence

When a container crashes or restarted, the container state is not saved so all of the files that were created during the lifetime of the container are lost.

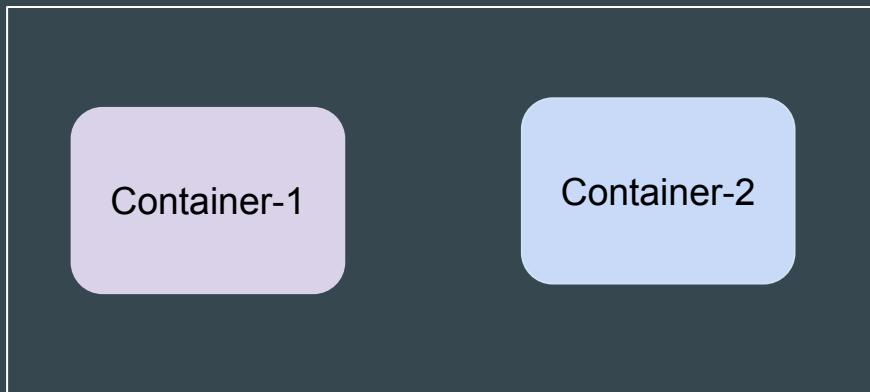
After a crash, kubelet restarts the container with a clean state.



# Understanding Challenge - Shared Storage

Another problem occurs when **multiple containers** are running in a Pod and need to share files.

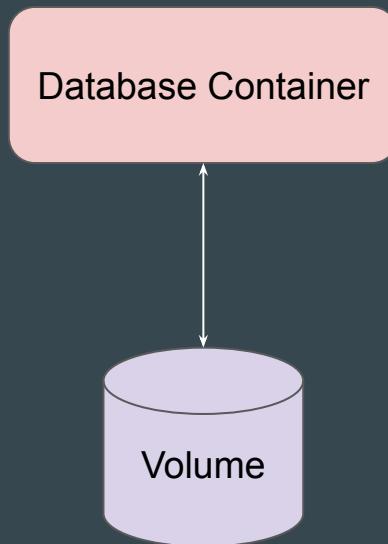
It can be challenging to set up and access a shared filesystem across all of the containers.



Multi-Container Pod

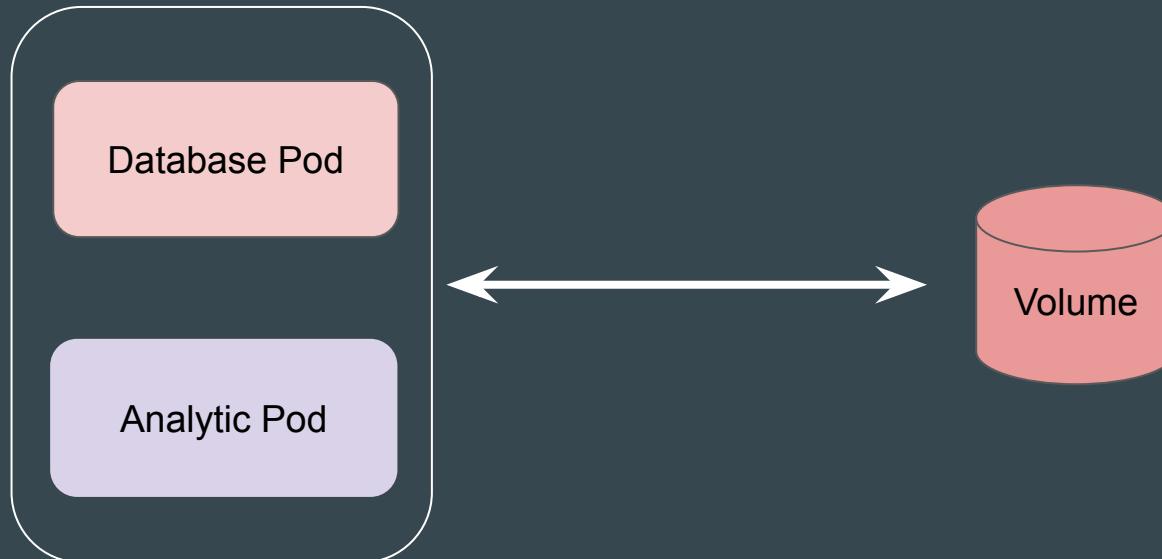
# Introducing Volumes

Volumes can provide a way to **store data that persists beyond the lifecycle of a container**.



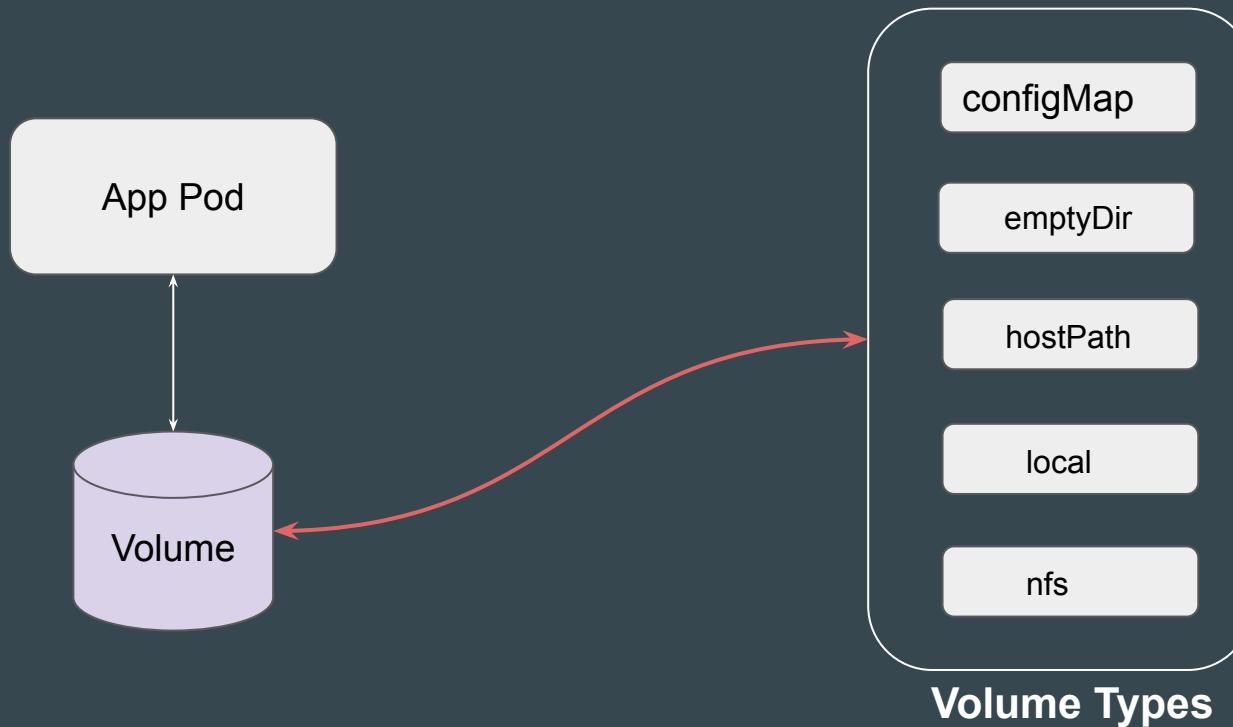
# Benefit - Shared Storage

A single volume can be attached to multiple Pods.



# Kubernetes Volumes

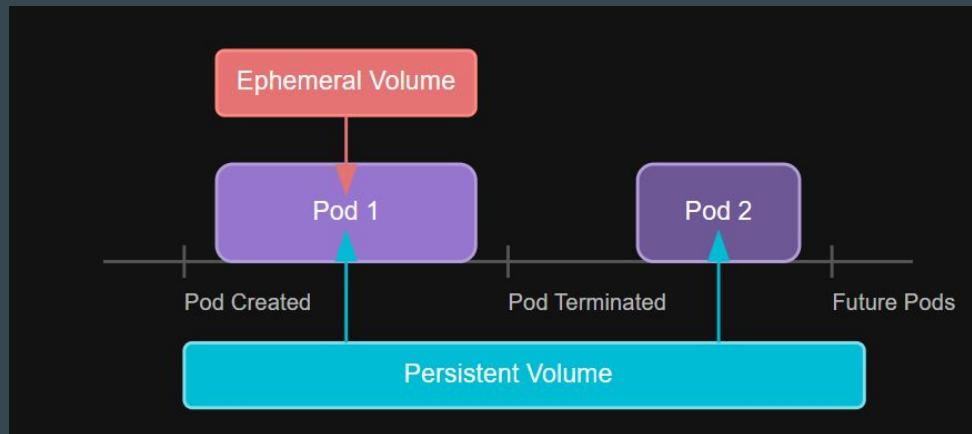
Kubernetes offers many volume types depending on the use-case.

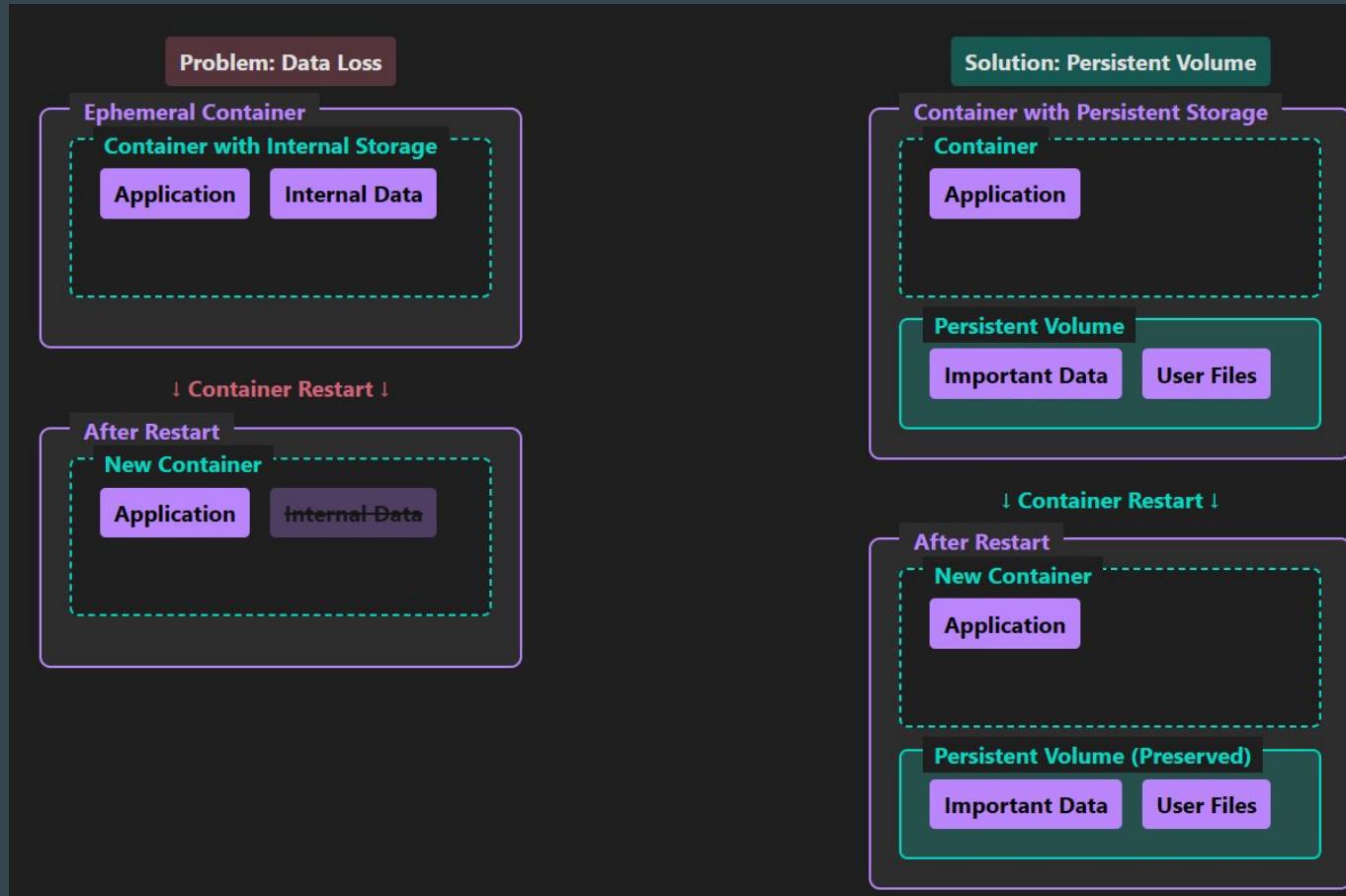


# Ephemeral and Persistent Volumes

Ephemeral volume types have a lifetime linked to a specific Pod, BUT persistent volumes exist beyond the lifetime of any individual pod.

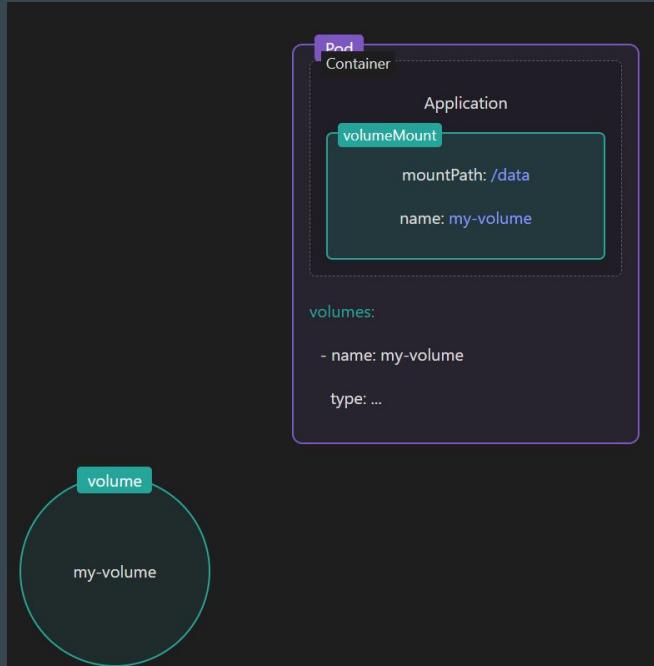
When a pod ceases to exist, Kubernetes destroys ephemeral volumes; however, Kubernetes does not destroy persistent volumes.

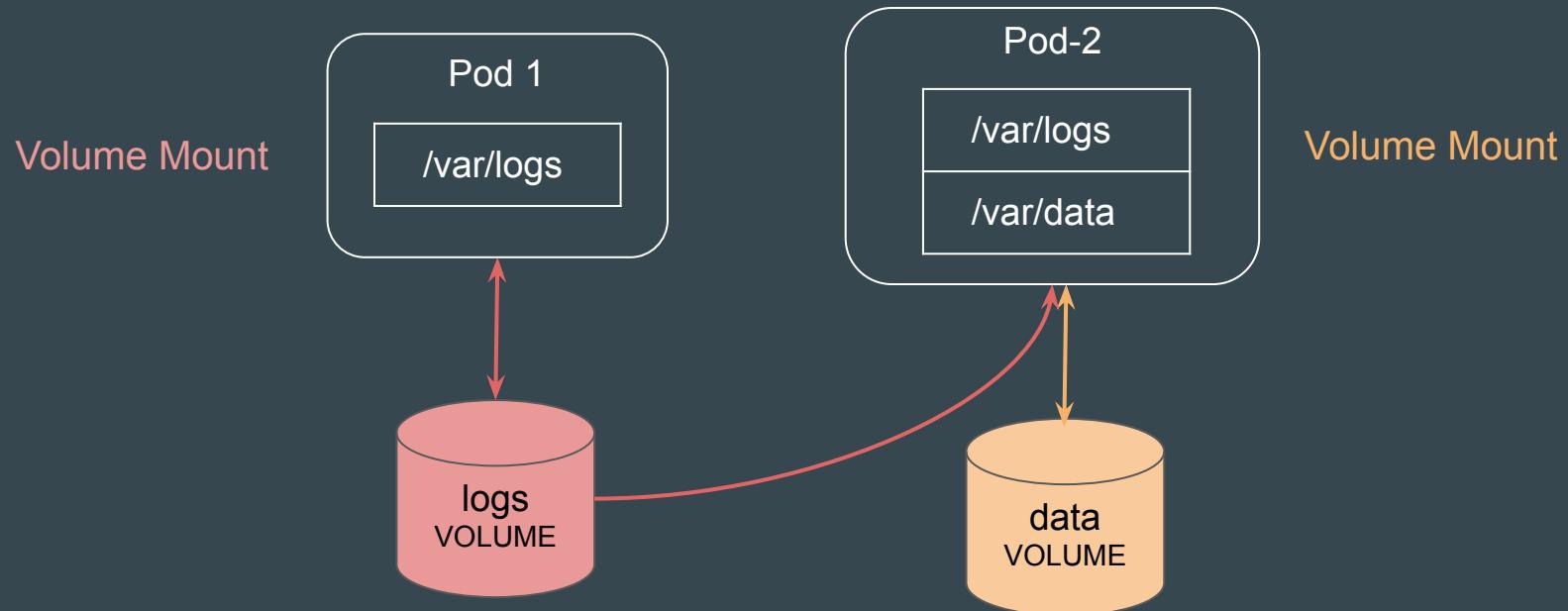




# Volume and Volume Mounts

To use a volume, specify the volumes to provide for the Pod in `.spec.volumes` and declare where to mount those volumes into containers in `.spec.containers[*].volumeMounts`.





# Sample Reference Code

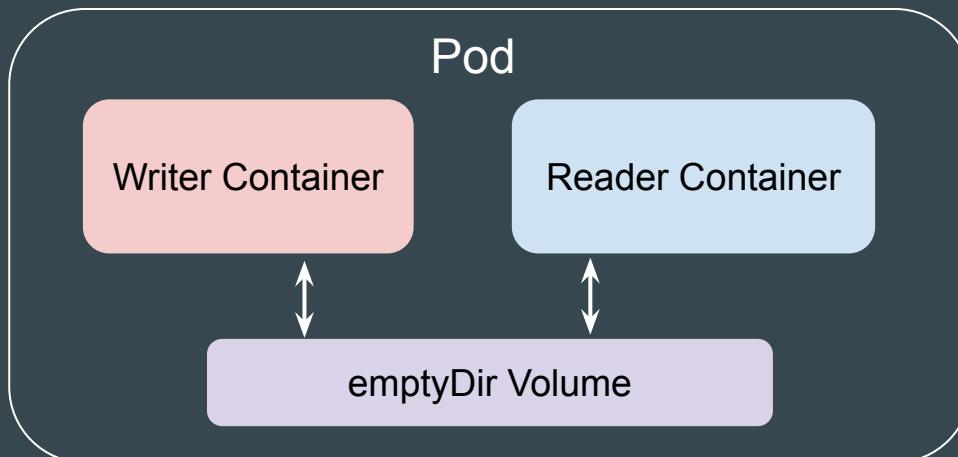
```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: nginx:latest
      name: test-container
      volumeMounts:
        - mountPath: /my-data
          name: data-volume
  volumes:
    - name: data-volume
      emptyDir:
        sizeLimit: 500Mi
```

## **Volume Type - emptyDir**

# Setting the Base

An emptyDir volume is a **temporary storage directory**.

All containers in the Pod can read and write the same files in the emptyDir volume.



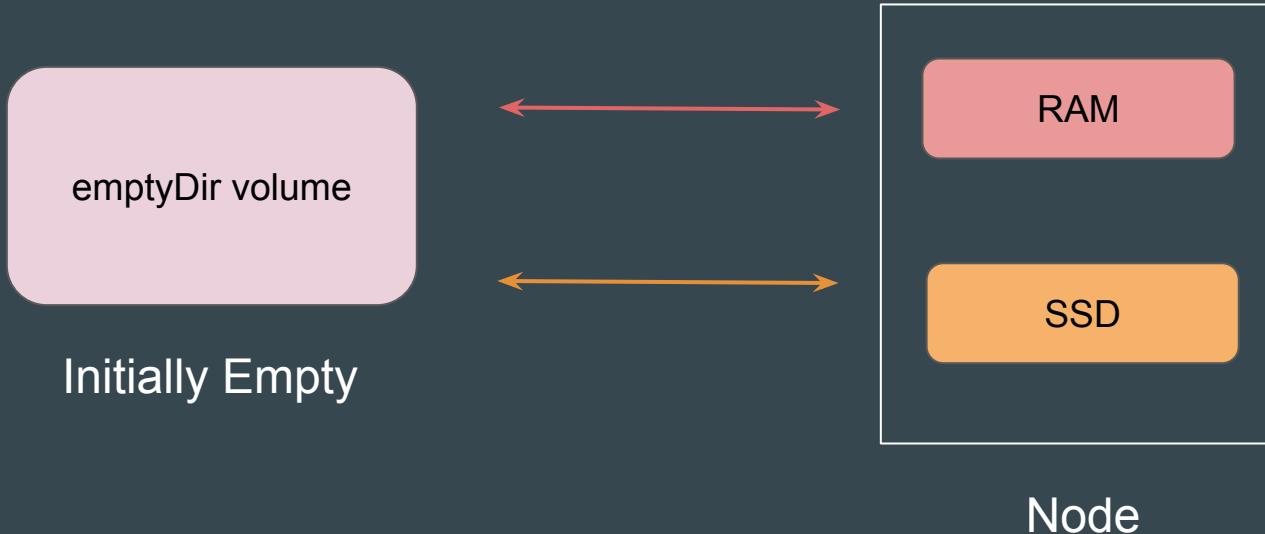
# Key Features of emptyDir

It is deleted when the pod is removed.

It can use memory instead of disk for performance.

A container crashing does not remove a Pod from a node. The data in an emptyDir volume is safe across container crashes.

# Workflow - EmptyDir



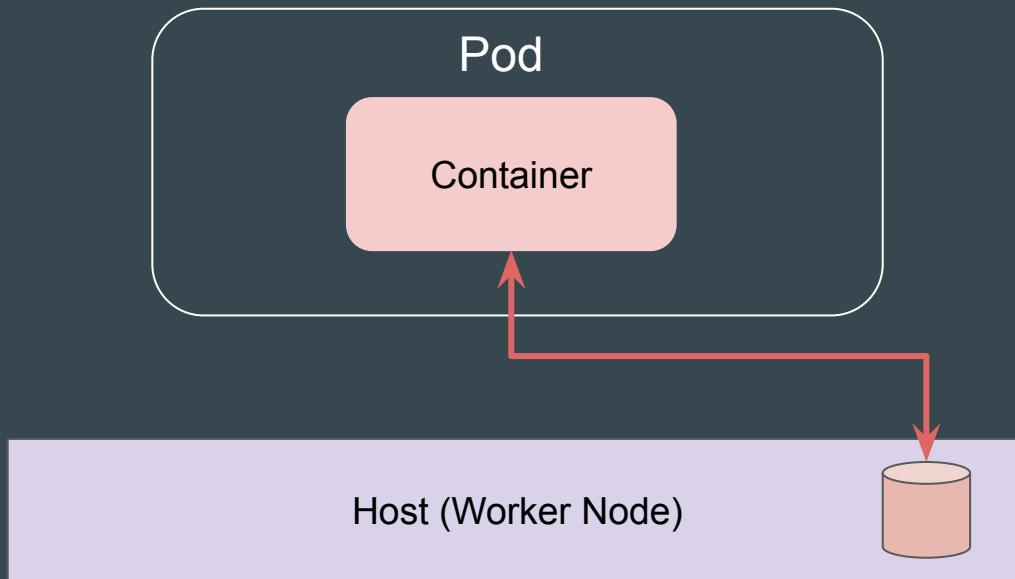
```
apiVersion: v1
kind: Pod
metadata:
  name: emptydir-demo
spec:
  volumes:
    - name: shared-storage
      emptyDir: {}
  containers:
    - name: busybox-container-1
      image: busybox
      command: ["sleep", "36000"]
      volumeMounts:
        - name: shared-storage
          mountPath: /data

    - name: busybox-container-2
      image: busybox
      command: ["sleep", "36000"]
      volumeMounts:
        - name: shared-storage
          mountPath: /data
```

## **Volume Type - hostPath**

# Setting the Base

A hostPath volume mounts a file or directory from the host node's filesystem into your Pod.



# Use-Cases of hostPath

Container needing access to worker node-specific logs like /var/logs for analysis.

Container that needs access to worker node-specific configuration files

Container that wants to write persistent data to a specific path in the node.

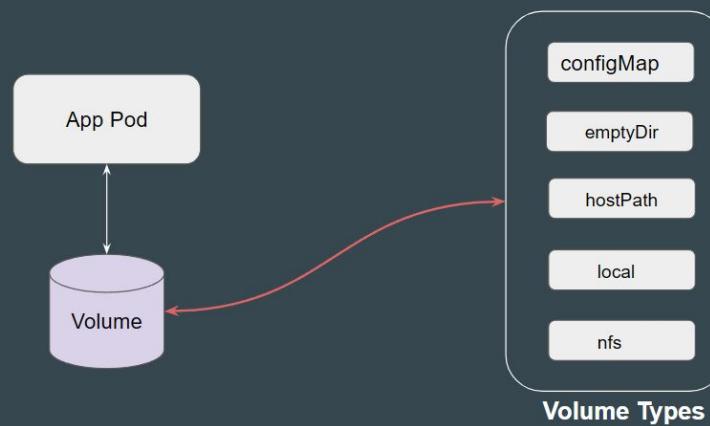
# **PersistentVolume and PersistentVolumeClaim**

# Setting the Base

Developers deploy application pods based on their requirements.

Supplying storage specific configurations can introduce complexity due to the many different storage types and their settings

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-client-pod
spec:
  containers:
    - name: app-container
      image: nginx
      volumeMounts:
        - name: nfs-volume
          mountPath: /mnt/nfs
  volumes:
    - name: nfs-volume
      nfs:
        server: 192.168.1.100
        path: /exported/path
        readOnly: false
```





Developer

Being a developer, my goal is to create a straightforward Pod manifest that includes volume information. I'd prefer not to handle storage provisioning and its associated configurations.



Storage Administrator

As a Storage Administrator, I am responsible for provisioning storage and managing its configurations. Developers can then reference this storage within their Pod specifications.

# Overview of Persistent Volume

**Persistent Volume** is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.

Every volume is created can be of different type and specifications.



Volume 1  
Size: Small  
Speed: Fast



Volume 2  
Size: Medium  
Speed: Fast



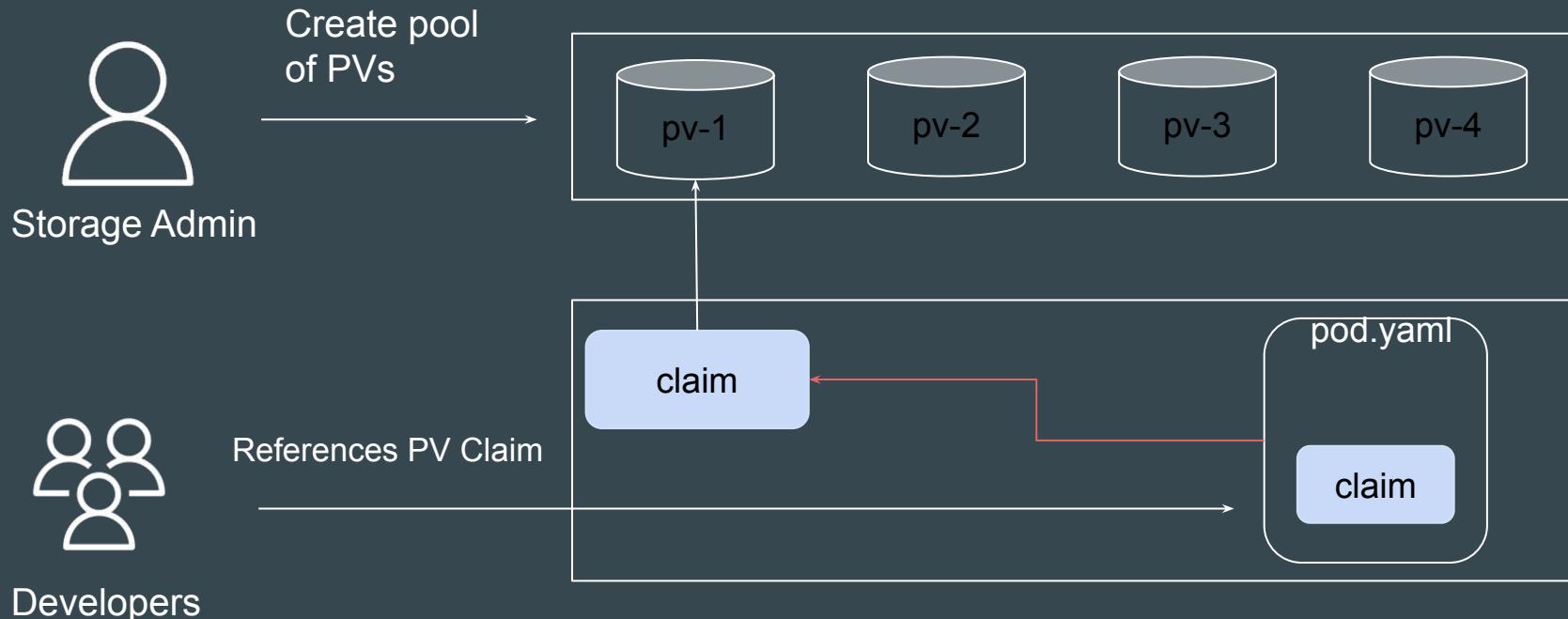
Volume 3  
Size: Big  
Speed: Slow

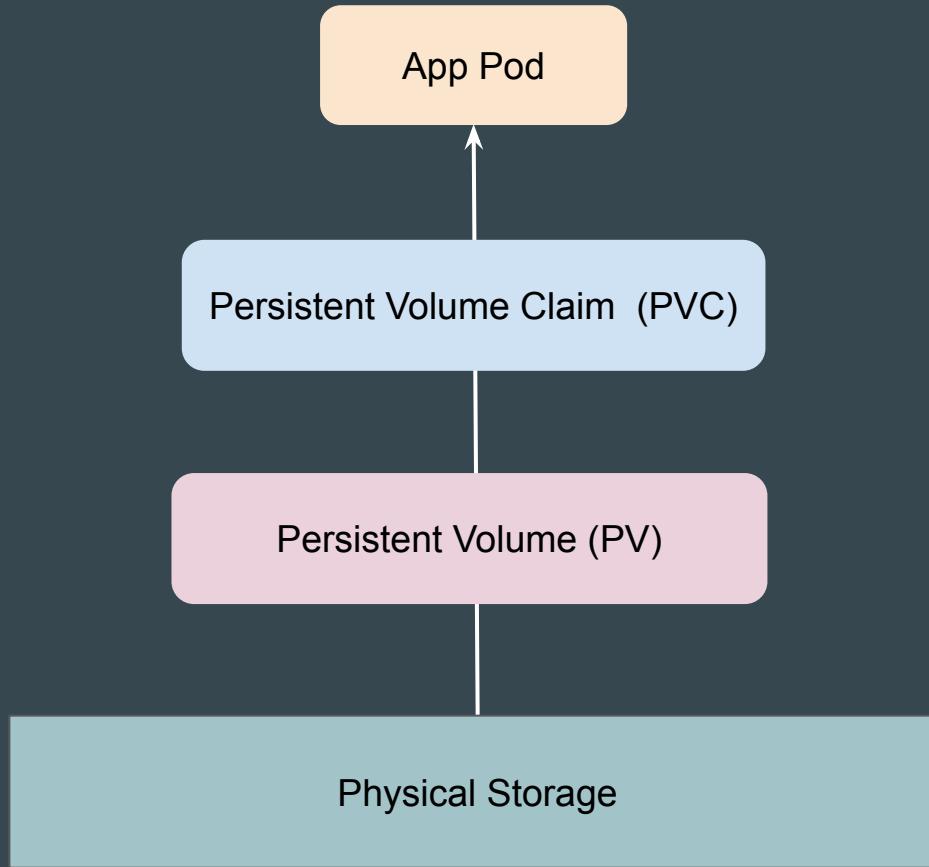


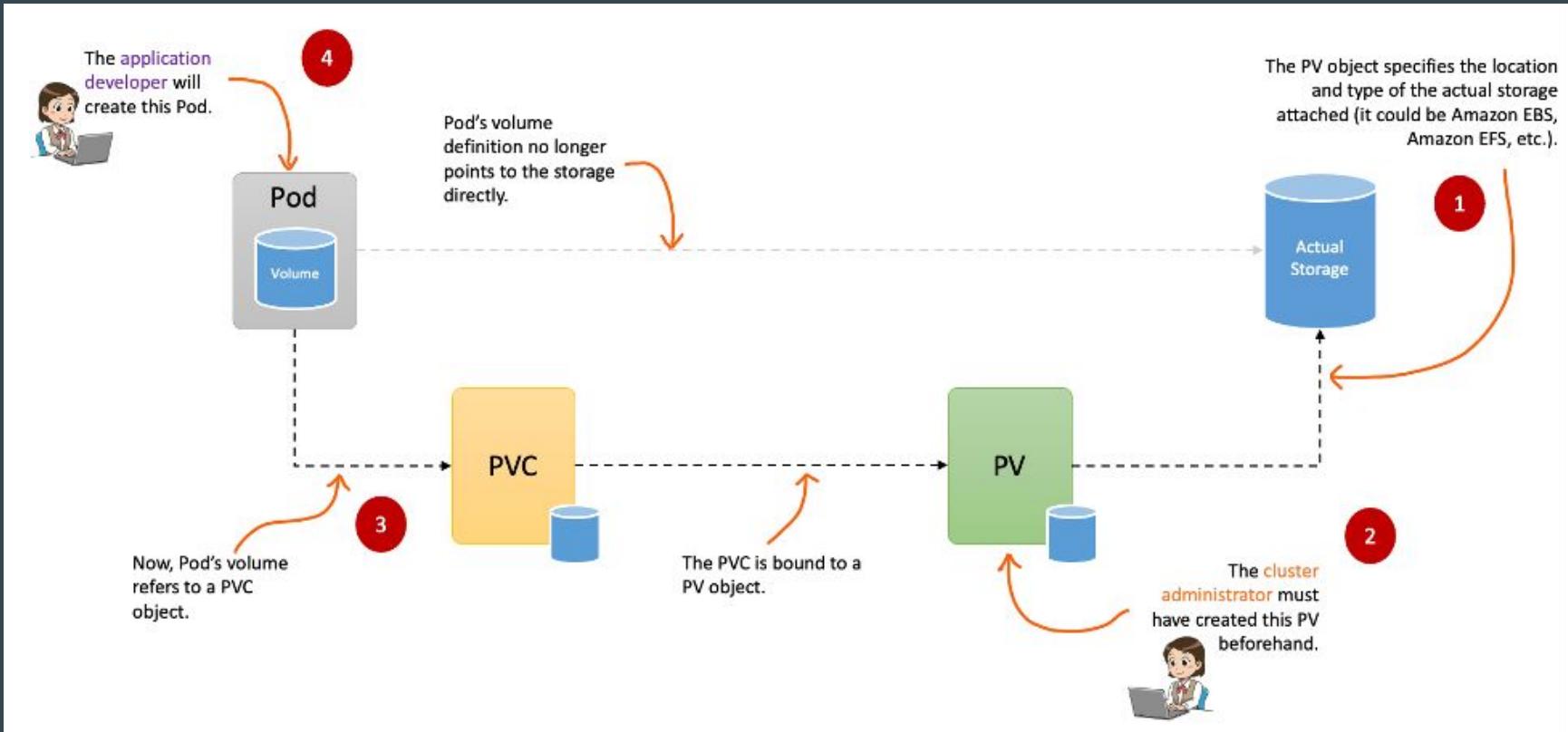
Volume 4  
Size: VSmall  
Speed: Ultra Fast

# PersistentVolumeClaim

PersistentVolumeClaim (PVC) is a request for storage by a user (e.g., a developer). It specifies size, access modes, and other requirements.

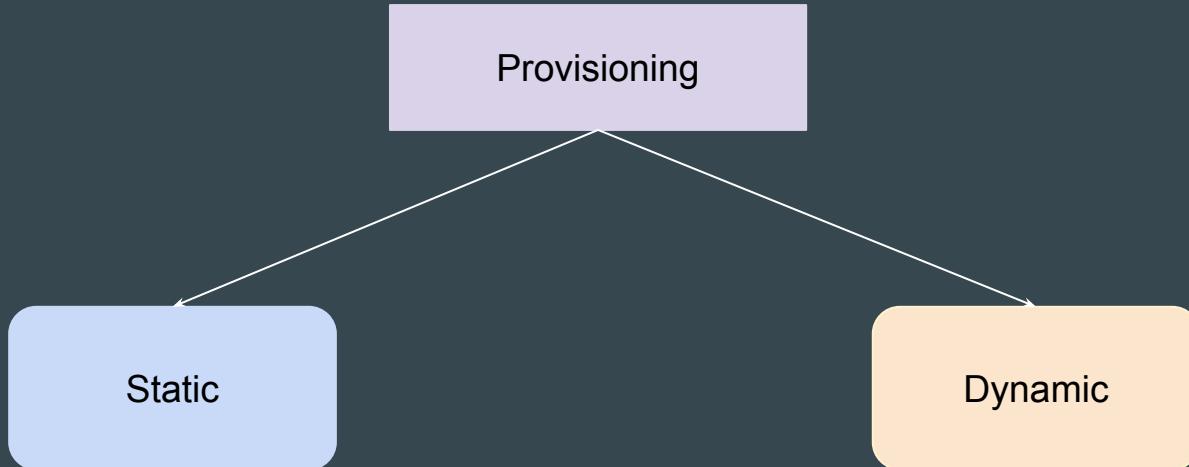






# **Persistent Volumes - Static vs Dynamic Provisioning**

# Provisioning of Persistent Volumes



PV must be created before PVC

PV is created dynamically at same type of PVC.

# Basic of Static Provisioning

Admin manually creates PVs ahead of time.

Users create PVCs that match the available PVs.

If no matching PV exists, the PVC remains unbound.

C:\kplabs-k8s>kubectl get pv							
NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	
manual-pv-1gb	1Gi	RWO	Retain	Available		manual	
manual-pv-3gb	3Gi	RWO	Retain	Available		manual	

# Basic of Dynamic Provisioning

With dynamic provisioning, you do not have to create a PV object.

Instead, it will be automatically created under the hood when you create the PVC. Kubernetes does so using another object called Storage Class

C:\>kubectl get pvc						
NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	
SCLASS	AGE					
do-pvc-1gb	Bound	pvc-86690381-df9a-4966-be5b-db45c61971c8	1Gi	RWO		do-block-storage

# Storage Class

**StorageClass** specifies the plugin or driver that determines how persistent volumes (PVs) are dynamically provisioned.

They contain appropriate provisioner used to provision volumes.

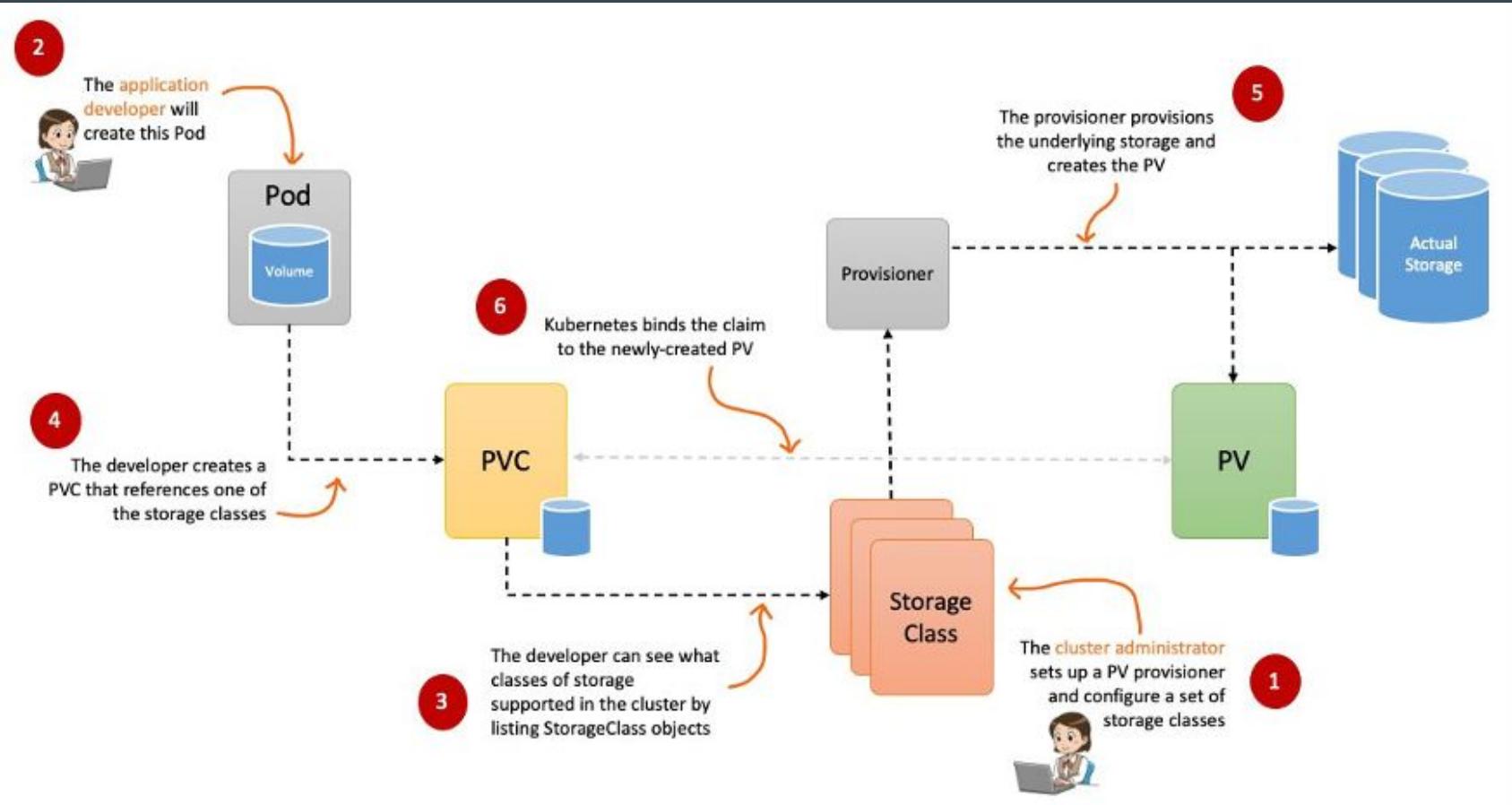
C:\>kubectl get storageclass	NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION
	do-block-storage (default)	dobs.csi.digitalocean.com	Delete	Immediate	true
	do-block-storage-retain	dobs.csi.digitalocean.com	Retain	Immediate	true
	do-block-storage-xfs	dobs.csi.digitalocean.com	Delete	Immediate	true
	do-block-storage-xfs-retain	dobs.csi.digitalocean.com	Retain	Immediate	true

# **Dynamic Provisioning using Local Path Provisioner**

# Setting the Base

The Local Path Provisioner offered by Rancher is a lightweight and simple **dynamic storage provisioner** for Kubernetes that uses local storage on the host node to provide PersistentVolumes (PVs).

```
root@kubeadm:~# kubectl get storageclass
NAME          PROVISIONER           RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-path    rancher.io/local-path Delete          WaitForFirstConsumer  false                  15h
```



# **Storage Class**

# Setting the Base

**StorageClass** specifies the plugin or driver that determines how persistent volumes (PVs) are dynamically provisioned.

They contain appropriate provisioner used to provision volumes.

C:\>kubectl get storageclass	NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION
	do-block-storage (default)	dobs.csi.digitalocean.com	Delete	Immediate	true
	do-block-storage-retain	dobs.csi.digitalocean.com	Retain	Immediate	true
	do-block-storage-xfs	dobs.csi.digitalocean.com	Delete	Immediate	true
	do-block-storage-xfs-retain	dobs.csi.digitalocean.com	Retain	Immediate	true

# Understanding the Structure

Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned to satisfy a PersistentVolumeClaim (PVC).

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: low-latency
provisioner: csi-driver.example-vendor.example
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

# Default Storage Class

You can mark a StorageClass as the default for your cluster.

When a PVC does not specify a storageClassName, the default StorageClass is used.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: low-latency
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-driver.example-vendor.example
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

# Volume binding mode

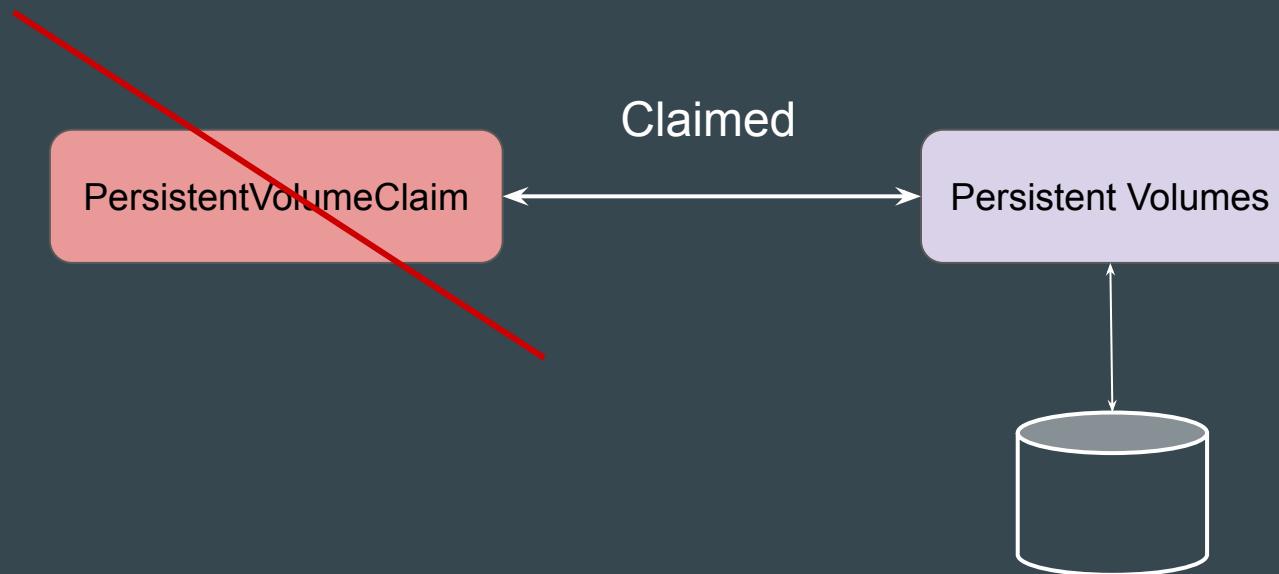
VolumeBindingMode determines when volume binding and dynamic provisioning occurs.

Binding Mode	Bind Time	Description
Immediate	At PVC creation	Volumes are provisioned and bound as soon as the PVC is created.
WaitForFirstConsumer	Volume binding and provisioning is delayed until a Pod using the PVC is scheduled.	At Pod scheduling

## **Persistent Volumes - Reclaim Policy**

# Setting the Base

The **ReclaimPolicy** specifies what happens to a Persistent Volume (PV) and its underlying storage resource after the Persistent Volume Claim (PVC) that was bound to it is deleted.



# Types of Reclaim Policy

Access Mode	Description
Delete	<p>When the PVC is deleted, both the PV object in Kubernetes and the associated underlying storage volume in the infrastructure are deleted.</p>
Retain	<p>When the PVC is deleted, the PV object remains in Kubernetes with its status marked as Released.</p> <p>Crucially, the underlying storage volume and its data are preserved.</p>

# Setting Reclaim Policy

Reclaim policy can be set both at a persistent volume level as well as Storage class level.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /tmp
    server: 172.17.0.2
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Retain
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /tmp
    server: 172.17.0.2
```

↔

Use this when you're statically provisioning a volume (i.e., creating the PV manually).

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Retain
```

↔

Use this when you're using dynamic provisioning, where PVCs automatically create PVs through the StorageClass.

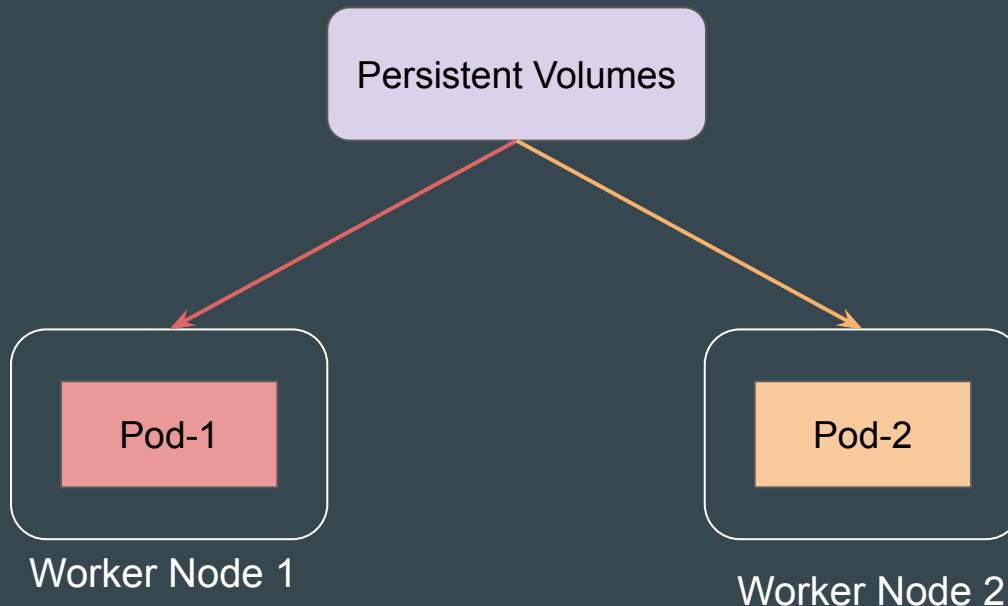
# Summary Table

Feature	Retain	Delete
<b>Data Retention After PVC Deletion</b>	<input checked="" type="checkbox"/> Yes — data is preserved	<input checked="" type="checkbox"/> No — underlying storage is deleted
<b>Underlying Storage Reused Automatically</b>	<input checked="" type="checkbox"/> No — requires manual intervention	<input checked="" type="checkbox"/> Yes — storage is deleted and cleaned up
<b>Manual Admin Cleanup Needed</b>	<input checked="" type="checkbox"/> Yes — admin must delete or reuse volume	<input checked="" type="checkbox"/> No — handled automatically
<b>Use Case</b>	Critical data, backups, manual recovery	Temporary or dynamically provisioned data
<b>Common Storage Backend Support</b>	All backends	Dynamic provisioners (e.g., AWS EBS, GCE PD)
<b>Recommended for</b>	Long-term storage, compliance, audits	Short-lived workloads, automation

# **Persistent Volumes - Access Modes**

# Setting the Base

Access Modes define how a Persistent Volume can be mounted and accessed by Nodes and Pods in the cluster.



<b>Access Mode</b>	<b>Abbreviation</b>	<b>Description</b>
ReadWriteOnce	RWO	Volume can be mounted as read-write by Pods in a single node.
ReadOnlyMany	ROX	Volume can be mounted as read-only by many nodes.
ReadWriteMany	RWX	Volume can be mounted as read-write by many nodes.
ReadWriteOncePod	RWOP	Volume can be mounted as read-write by a single Pod.

# 1 - ReadWriteOnce (RWO)

The volume can be mounted as read-write by a single Node at a time.

Even if multiple Pods on the same Node use the same PVC requesting RWO, they can all potentially read from and write to the volume.

However, Pods on different Nodes cannot simultaneously mount this volume as read-write.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: standard
```

## 2 - ReadOnlyMany (ROX)

The volume can be mounted as read-only by many Nodes simultaneously.

Multiple Pods across multiple Nodes can mount and read from this volume concurrently. No Pod can write to it.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: config-data
spec:
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 5Gi
  storageClassName: nfs-storage
```

## 3 - ReadWriteMany (RWX)

The volume can be mounted as read-write by many Nodes simultaneously.

Multiple Pods running on different Nodes can all read from and write to the same volume concurrently. This requires a storage backend that supports shared access.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  storageClassName: nfs-storage
```

## 4 - ReadWriteOncePod (RWOP)

The volume can be mounted as **read-write** by a single Pod only.

This is the most restrictive mode. Only one specific Pod in the entire cluster can mount this volume as read-write.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: secure-pvc
spec:
  accessModes:
    - ReadWriteOncePod
  resources:
    requests:
      storage: 5Gi
  storageClassName: csi-storage
```

# Important Considerations

Not all the volume types support all the access modes.

Always check the cloud provider or volume plugin documentation to see which access modes are supported.

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany	ReadWriteOncePod
AzureFile	✓	✓	✓	-
CephFS	✓	✓	✓	-
CSI	depends on the driver			
FC	✓	✓	-	-
FlexVolume	✓	✓	depends on the driver	-
HostPath	✓	-	-	-
iSCSI	✓	✓	-	-
NFS	✓	✓	✓	-

# Summary Table

Access Mode	Abbreviation	Mounted By	Read	Write	Use Case Example
ReadWriteOnce	RWO	Single Node	✓	✓	AWS EBS, GCP PD, Azure Disk — standard volume for stateful apps
ReadWriteOncePod	RWOP	Single Pod (1 Node)	✓	✓	Security-sensitive apps needing strict single-pod access
ReadOnlyMany	ROX	Multiple Nodes	✓	✗	Shared read-only config or datasets
ReadWriteMany	RwX	Multiple Nodes	✓	✓	NFS, CephFS — shared writable volume across pods/nodes

# **ConfigMaps**

# Understanding the Challenge - Part 1

Whenever an App pod gets deployed, it might need to connect to an external database to store data.

**Issue:** In many cases, these details are hard coded as part of the container image.

Key	Value
db_user	dbadmin
db_pass	A123#
db_url	172.31.10.30:3306

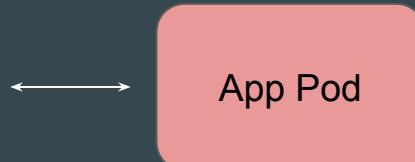


Hardcoded Data

# Understanding the Challenge - Part 2

If a container has hardcoded data, any change to the data requires you to create a new set of Docker images and recreate the Pod

Key	Value
db_user	dbadmin
db_pass	A123#
db_url	172.31.10.30:3306



Hardcoded Data

# Introduction to the ConfigMaps

A ConfigMap in Kubernetes is used to **store key-value pairs** of configuration data.

ConfigMap



Key	Value
db_user	dbadmin
db_pass	A123#
db_url	172.31.10.30:3306

Key	Value
db_user	devadmin
db_pass	A123#
db_url	10.77.0.5:3306

Fetch from Prod  
ConfigMap

App Pod (Prod)

Fetch from Dev  
ConfigMap

App Pod (Dev)

## Point to Note

The primary purpose of ConfigMap is to store non-sensitive configuration data, such as configuration files, environment variables, etc.

It stores data in plain-text and is not intended for sensitive information.

# Practical Approach for Entire Workflow

Part 1: Create ConfigMap

Part 2: Configure Pod to use that appropriate ConfigMap

# **ConfigMap Practical - Part 1 (Create ConfigMap)**

# Command to Create ConfigMap

The `kubectl create configmap` command allows us to create ConfigMap from a file, directory, or specified literal value

Examples:

```
# Create a new config map named my-config based on folder bar
kubectl create configmap my-config --from-file=path/to/bar
```

```
# Create a new config map named my-config with specified keys instead of file basenames on disk
kubectl create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-file=key2=/path/to/bar/file2.txt
```

```
# Create a new config map named my-config with key1=config1 and key2=config2
kubectl create configmap my-config --from-literal=key1=config1 --from-literal=key2=config2
```

```
# Create a new config map named my-config from the key=value pairs in the file
kubectl create configmap my-config --from-file=path/to/bar
```

```
# Create a new config map named my-config from an env file
kubectl create configmap my-config --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

# Approach 1 - From a Literal

The **--from-literal** option allows you to directly specify key-value pairs as command-line arguments.

```
C:\>kubectl create configmap dev-config --from-literal=db_user=dbadmin --from-literal=db_host=172.31.0.5  
configmap/dev-config created
```

## Approach 2 - From a File

The **--from-file** option allows you to create a ConfigMap from the contents of one or more files.

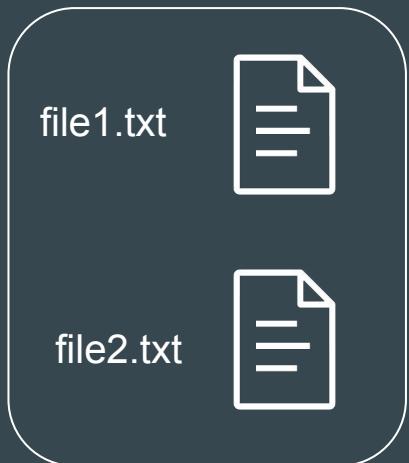


```
kubectl create configmap --from-file=large-file.txt
```

large-file.txt

## Approach 3 - From a Directory

You can also use the `--from-file` option with a directory instead of a single file.



```
kubectl create configmap demo --from-file=<path/to/directory>
```

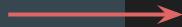
# Comparison of All the Methods

Method	Use-Case	Advantage	Disadvantage
--from-literal	Simple, one-off key-value pairs	Quick and easy for simple configurations	Not suitable for complex configurations or large amounts of data
--from-file (file)	Individual configuration files	Good for managing separate configuration files	Can become cumbersome for many files
--from-file (dir)	Multiple configuration files organized in a directory	Convenient for grouping related configuration files	Less control over individual key names (filenames are used as keys)

# Type of Data In ConfigMap

In the ConfigMap manifest file, you can represent data in multiple distinct formats.

Simple Key Value pair



```
! configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: demo-configmap
data:
  key1: "value1"
  key2: "value2"

  essay: |
    This is the first line of the essay1.
    It spans multiple lines and contains various information.
    This is Line 3

  json_data: |
    {
      "name": "Alice",
      "age": 26,
      "skills": ["Kubernetes", "Docker", "DevOps"]
    }
```

Multiline Block literal

## Point to Note - Directory Approach

If you are referencing to an entire directory, Kubernetes will create a ConfigMap with each file in that directory becoming a key-value pair.

The filename (without extension) becomes the key, and the file content becomes the value

## **ConfigMap Practical - Part 2 (Mounting to Pods)**

# Setting the Base

Once ConfigMaps is created, we also need to reference it to appropriate Pod.

ConfigMap

Key	Value
db_user	dbadmin
db_pass	A123#
db_url	172.31.10.30:3306

Key	Value
db_user	devadmin
db_pass	A123#
db_url	10.77.0.5:3306

Mount from Prod  
ConfigMap

App Pod (Prod)

Mount from Dev  
ConfigMap

App Pod (Dev)

# Different Approaches for Reference

There are multiple ways through which a Pod can fetch the data of a ConfigMap.



# Approach 1 - Volume Mount

In this method, the ConfigMap is mounted directly as a volume in the Pod.

Each key-value pair in the ConfigMap appears as a file in the volume.



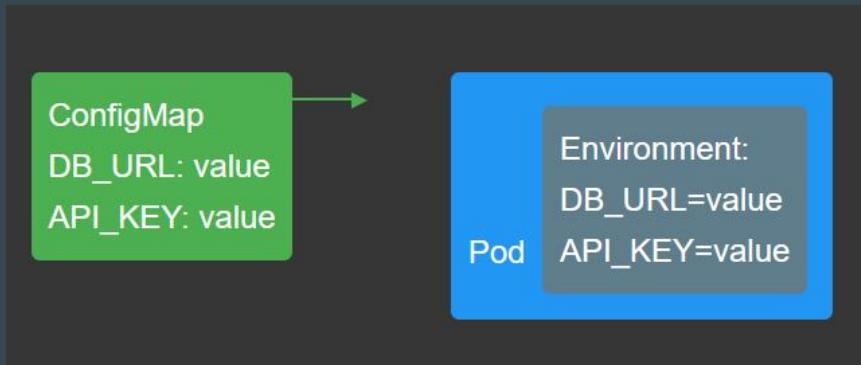
# Reference Manifest File

```
! pod-volume.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: app-pod
  spec:
    containers:
      - name: app-container
        image: nginx
        volumeMounts:
          - name: config-volume
            mountPath: /etc/config
    volumes:
      - name: config-volume
        configMap:
          name: app-config
```

## Approach 2 - Environment Variables

In this method, the values in the ConfigMap are exposed as environment variables to the container.

The application can then access these values using standard environment variable lookup.



# Reference Screenshot

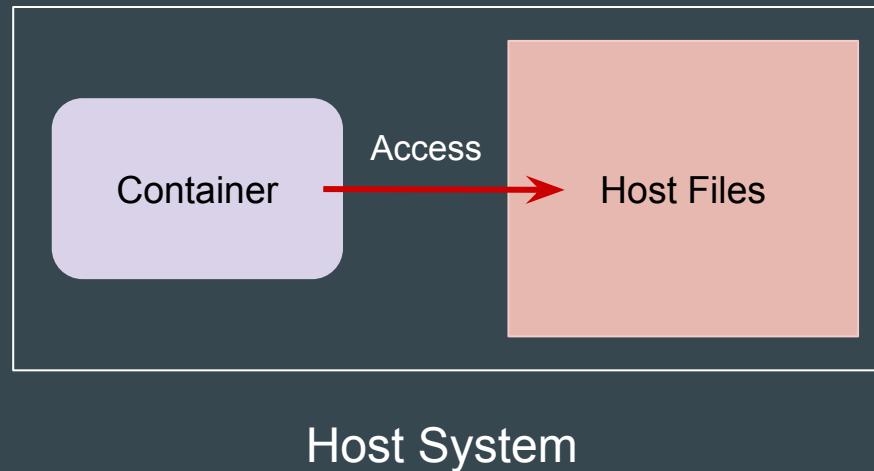
```
! pod-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
    - name: app-container
      image: nginx
      env:
        - name: APP_MODE
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: APP_MODE
        - name: APP_ENV
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: APP_ENV
```

# **Security Context**

# Understanding the Challenge

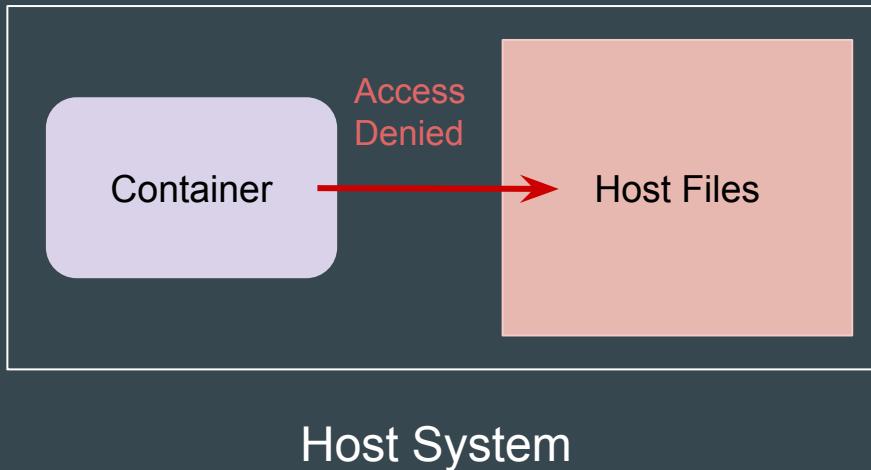
Many times, the containers run with **root user privileges**.

In case of **container breakouts**, an attacker can get full access to the host system.



# Running Container with Non Root User

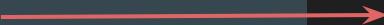
If the container runs with non-root privileges, it will be unable to modify the critical host files and will have limited access to the host system.



# Introduction to Security Context

A security context defines privilege and access control settings for a Pod or Container.

Run as non-privileged user



```
apiVersion: v1
kind: Pod
metadata:
  name: better-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 1000
  containers:
  - name: better-container
    image: busybox
    command: ["sleep", "36000"]
```

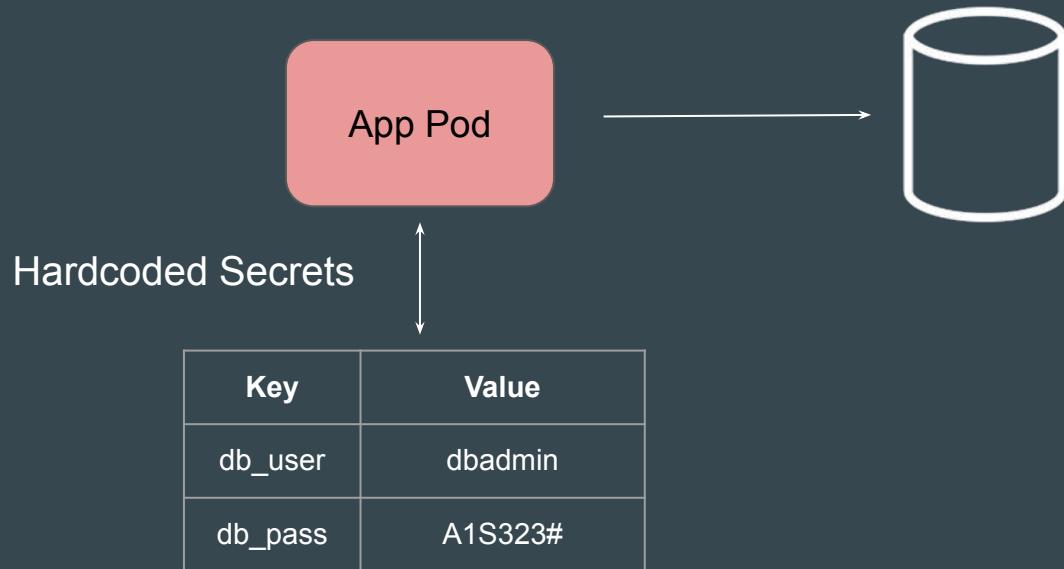
# Comparison Table

Field	Description	Use-Case
runAsUser	Specifies the user ID (UID) a container's process runs as.	Use when you want the container to run as a specific user rather than the default (commonly root).
runAsGroup	Specifies the primary group ID (GID) a container's process runs as.	Use when you want the container's primary group to be a specific GID.
fsGroup	Specifies a group ID (GID) for volume-mounted files. Files created in mounted volumes will be owned by this GID.	Use when you need to control file permissions for a shared volume (e.g., for multiple containers in a Pod).

# **Kubernetes Secrets**

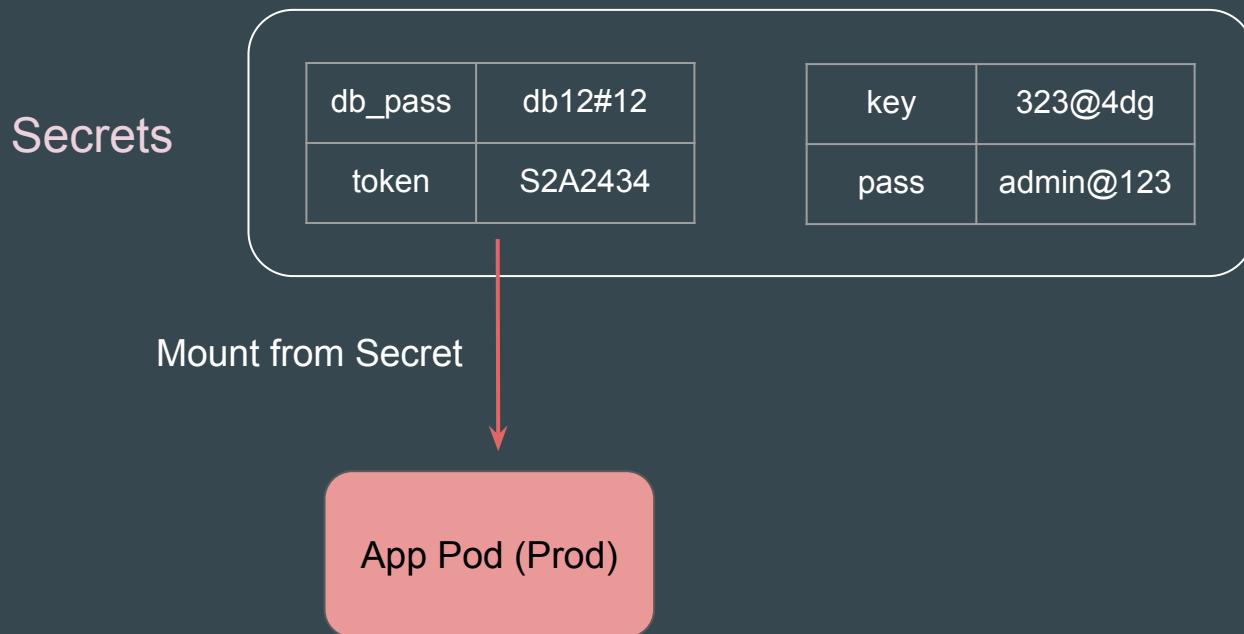
# HardCoding Secrets Should be Avoided

It is frequently observed that sensitive data like passwords, tokens, etc., are hard-coded as part of the container image.



# Introducing Secret

Kubernetes Secrets is a feature that allows us to store these sensitive data.



# Reference Screenshot

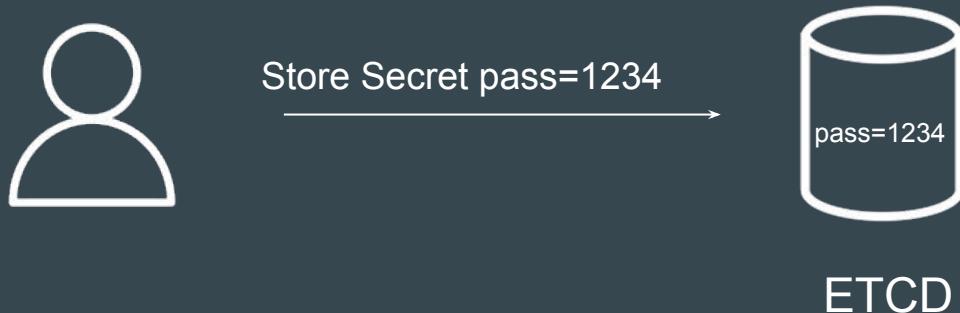
```
C:\>kubectl get secret
NAME          TYPE        DATA   AGE
my-secret     Opaque      1      22m
```

```
C:\>kubectl get secret my-secret -o yaml
apiVersion: v1
data:
  db_pass: QTIjMTI1U0A=
kind: Secret
metadata:
  creationTimestamp: "2025-01-16T02:34:21Z"
  name: my-secret
  namespace: default
  resourceVersion: "5877928"
  uid: d3c383cb-c2e3-4f9b-95ef-d1f0ec6a04be
type: Opaque
```

## Point to Note - Part 1

By default, Secrets are not very secure as they are not stored in encrypted format in the data store (ETCD). You can setup this configuration manually.

You can also additionally protect access to secrets using RBAC for access control.



## Point to Note - Part 2

When you view a secret, Kubectl will print the Secret in **base64** encoded format.

You'll have to use an external base64 decoder to decode the Secret fully

```
C:\>kubectl get secret my-secret -o yaml
apiVersion: v1
data:
  db_pass: QTIjMTI1U0A= ←
kind: Secret
metadata:
  creationTimestamp: "2025-01-16T02:34:21Z"
  name: my-secret
  namespace: default
  resourceVersion: "5877928"
  uid: d3c383cb-c2e3-4f9b-95ef-d1f0ec6a04be
type: Opaque
```

base64 encoded

**sysctl**

# Setting the Base

At its core, the Linux kernel has many internal settings that control its behavior.

These settings, often called "**kernel parameters**" and allow administrators to fine-tune the kernel's operation for specific workloads.

```
root@kubeadm:~# ls -l /proc/sys/net/ipv4/
total 0
-rw-r--r-- 1 root root 0 Apr 10 13:05 cipso_cache_bucket_size
-rw-r--r-- 1 root root 0 Apr 10 13:05 cipso_cache_enable
-rw-r--r-- 1 root root 0 Apr 10 13:05 cipso_rbm_optfmt
-rw-r--r-- 1 root root 0 Apr 10 13:05 cipso_rbm_strictvalid
dr-xr-xr-x 1 root root 0 Apr 10 12:46 conf
-rw-r--r-- 1 root root 0 Apr 10 13:05 fib_multipath_hash_fields
-rw-r--r-- 1 root root 0 Apr 10 13:05 fib_multipath_hash_policy
-rw-r--r-- 1 root root 0 Apr 10 13:05 fib_multipath_use_neigh
-rw-r--r-- 1 root root 0 Apr 10 13:05 fib_notify_on_flag_change
-rw-r--r-- 1 root root 0 Apr 10 13:05 fib_sync_mem
-rw-r--r-- 1 root root 0 Apr 10 13:05 fwmark_reflect
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_echo_enable_probe
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_echo_ignore_all
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_echo_ignore_broadcasts
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_errors_use_inbound_ifaddr
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_ignore_bogus_error_responses
-rw-r--r-- 1 root root 0 Apr 10 13:05 icmp_msgs_burst
```

# Introducing sysctl

The **sysctl** is the standard utility for interacting with kernel parameters exposed via /proc/sys.

It provides a more convenient interface than directly manipulating files.

```
root@kubeadm:~# cat /proc/sys/net/ipv4/ip_forward
1
root@kubeadm:~# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

# Temporary Modification

We can use sysctl to directly modify some of the kernel parameters.

Directly modifying parameter is a temporary way and will not persist during restart.

```
root@kubeadm:~# sysctl net.ipv4.conf.all.accept_redirects  
net.ipv4.conf.all.accept_redirects = 0  
root@kubeadm:~# sysctl net.ipv4.conf.all.accept_redirects=1  
net.ipv4.conf.all.accept_redirects = 1  
root@kubeadm:~# sysctl net.ipv4.conf.all.accept_redirects  
net.ipv4.conf.all.accept_redirects = 1
```

# Permanent Modification

To ensure your kernel parameter changes survive a reboot, **you need to store them in configuration files** that the system reads during boot.

Configuration Files	Description
/etc/sysctl.conf	The traditional main configuration file. You can add your parameter settings here.
/etc/sysctl.d/*.conf	The modern, preferred approach  You should place your custom settings in a file here, typically named something like 99-custom.conf

```
root@kubeadm:~# ls -l /etc/sysctl.d/
total 48
-rw-r--r-- 1 root root  481 Sep 26  2024 10-bufferbloat.conf
-rw-r--r-- 1 root root   77 Mar 31  2024 10-console-messages.conf
-rw-r--r-- 1 root root  490 Mar 31  2024 10-ipv6-privacy.conf
-rw-r--r-- 1 root root 1229 Mar 31  2024 10-kernel-hardening.conf
-rw-r--r-- 1 root root 1184 Mar 31  2024 10-magic-sysrq.conf
-rw-r--r-- 1 root root  164 Mar 31  2024 10-map-count.conf
-rw-r--r-- 1 root root  158 Mar 31  2024 10-network-security.conf
-rw-r--r-- 1 root root 1292 Mar 31  2024 10-ptrace.conf
-rw-r--r-- 1 root root  506 Mar 31  2024 10-zeropage.conf
-rw-r--r-- 1 root root  185 Jan  8 15:11 99-cloudimg-ipv6.conf
lrwxrwxrwx 1 root root    14 Aug  8 2024 99-sysctl.conf -> ../../sysctl.conf
-rw-r--r-- 1 root root  798 Mar 24  2024 README.sysctl
```

# Applying the Changes

After editing `/etc/sysctl.conf` or files in `/etc/sysctl.d/`, the changes won't take effect until the next boot unless you apply them manually.

You can run the `sysctl --system` to load the parameters from these configuration files again.

```
root@kubeadm:~# sysctl --system
* Applying /usr/lib/sysctl.d/10-apparmor.conf ...
* Applying /etc/sysctl.d/10-bufferbloat.conf ...
* Applying /etc/sysctl.d/10-console-messages.conf ...
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
* Applying /etc/sysctl.d/10-magic-sysrq.conf ...
* Applying /etc/sysctl.d/10-map-count.conf ...
* Applying /etc/sysctl.d/10-network-security.conf ...
* Applying /etc/sysctl.d/10-ptrace.conf ...
* Applying /etc/sysctl.d/10-zero-page.conf ...
```

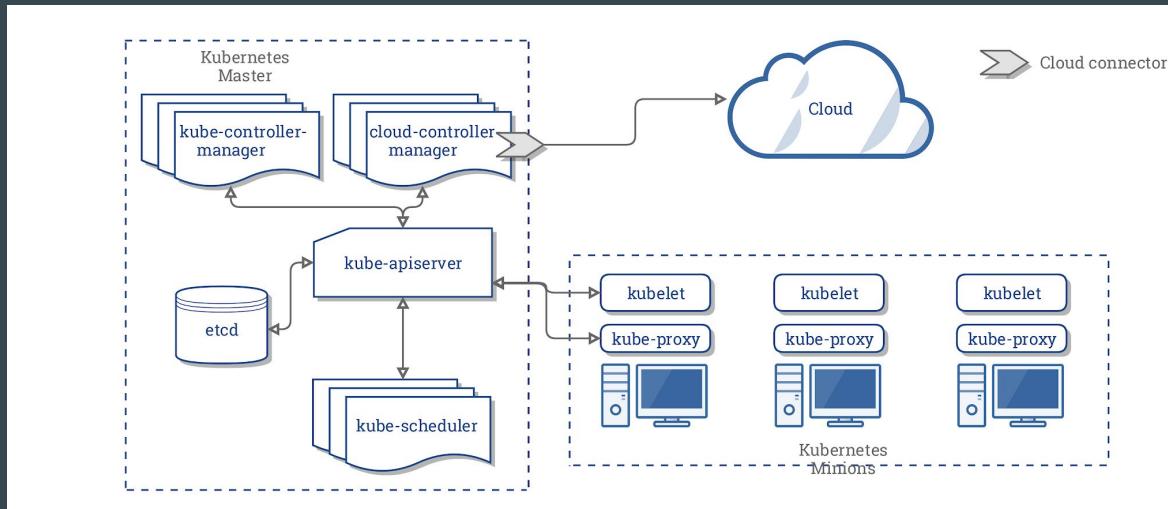
# Ways to Apply sysctl settings

Method	Scope	Persistent?	Notes
<code>sysctl key=value</code>	Runtime	✗	Resets on reboot
<code>echo value &gt; /proc/sys/...</code>	Runtime	✗	Equivalent to <code>sysctl</code>
<code>/etc/sysctl.conf</code>	Boot-time	✓	Main config file
<code>/etc/sysctl.d/*.conf</code>	Boot-time	✓	Per-package/app settings
<code>sysctl -p</code>	Runtime	✓ (if file is already edited)	Loads from <code>/etc/sysctl.conf</code>
<code>sysctl --system</code>	Runtime	✓	Loads from all config files

# **Overview - Setting Up Kubernetes Cluster with Kubeadm**

# Understanding the Need

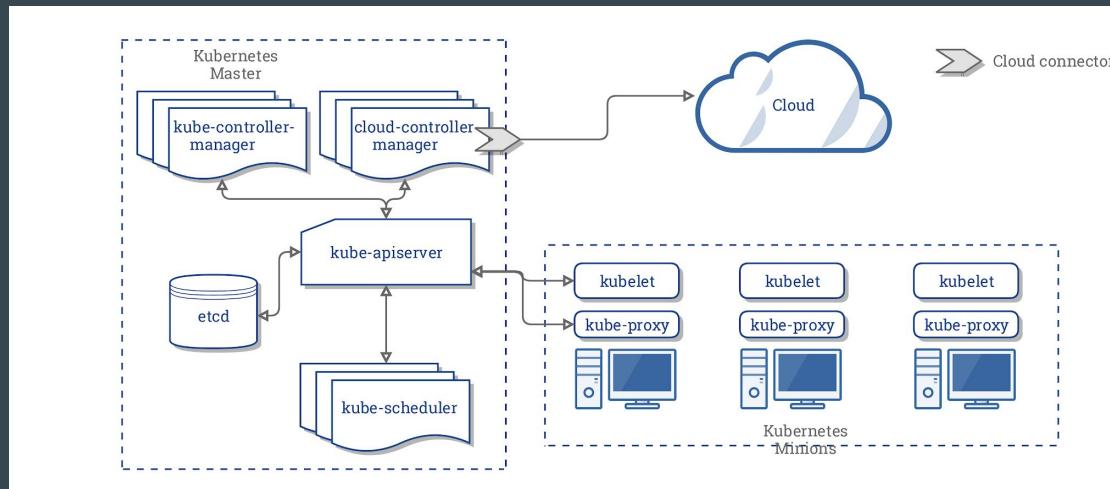
**kubeadm** allows us to provision a **secure** Kubernetes cluster quickly.



# Two Part to Remember

First important component is Kubernetes **Master Node**

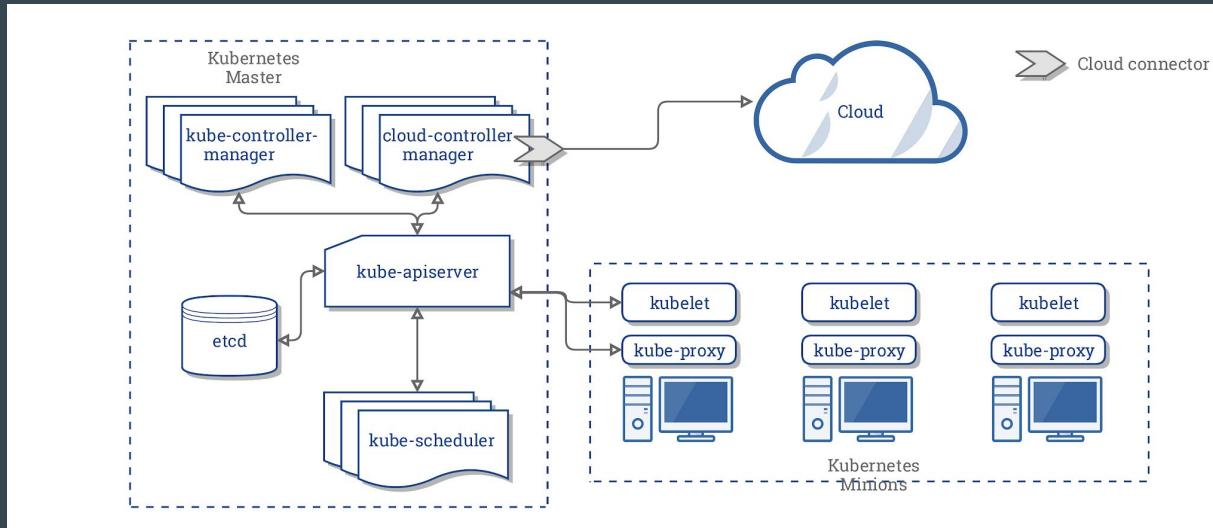
Second component is Kubernetes **Worker Node**



# **Kubernetes Cluster from Scratch**

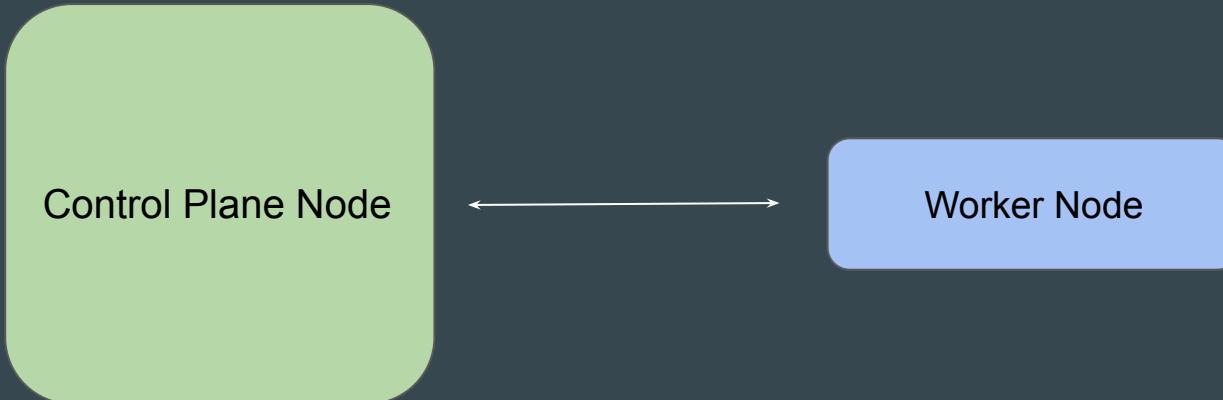
# Goal of this Section

We will be configuring Kubernetes Control Plane + Worker node from Scratch.



# Our Architecture

We will use **two servers**; one will be for the Control Plane node and the second for the Worker node.



# Point to Note - Exam Perspective

Deploying the Kubernetes cluster from scratch is **NOT** part of the exams.

You can expect small troubleshooting-related questions.

Your high-level understanding of K8s Cluster deployment will help you solve the questions.

# Version Constraints

New Kubernetes versions are released very frequently.

It is important to use the same version of Kubernetes as shown in our setup guide.



Instructor



K8s Cluster 1.32



Users

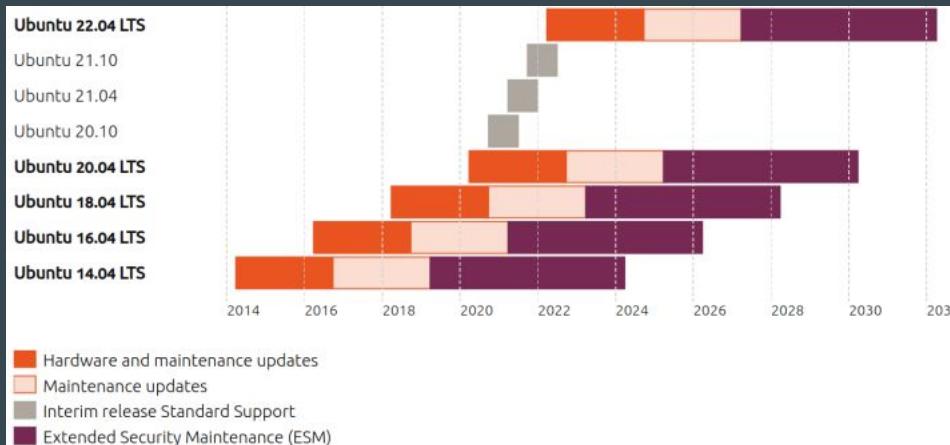


K8s Cluster 1.33

# OS Constraints

It is recommended to use the same OS version as that of the demo.

We have intentionally used Ubuntu LTS (Long Term Support)

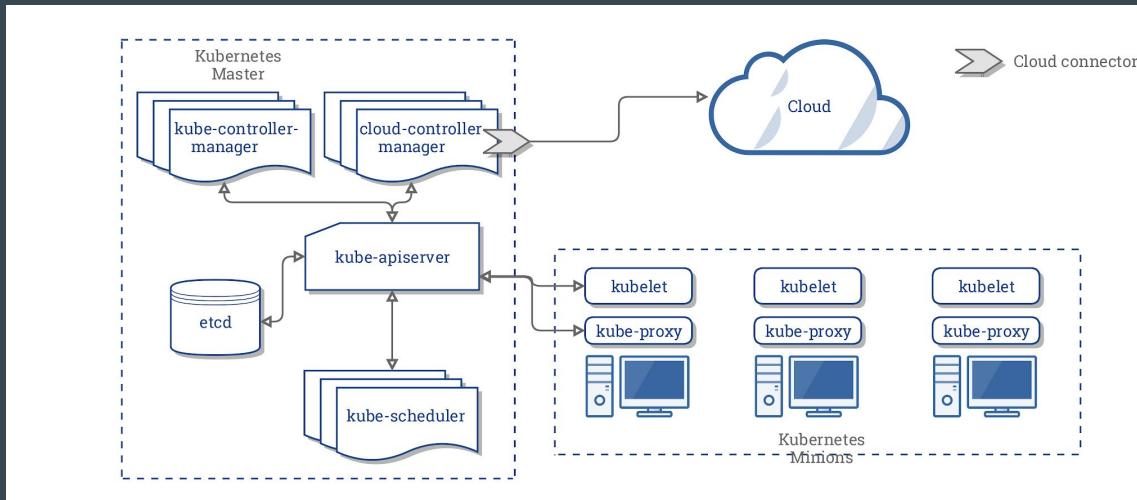


# **Common Patterns - K8s from Scratch**

# Point to Note

While configuring individual K8s components, there are common steps that we will perform for each component.

This can include creating certificates, and kubeconfig files.



# Common Pattern 1 - Certificates

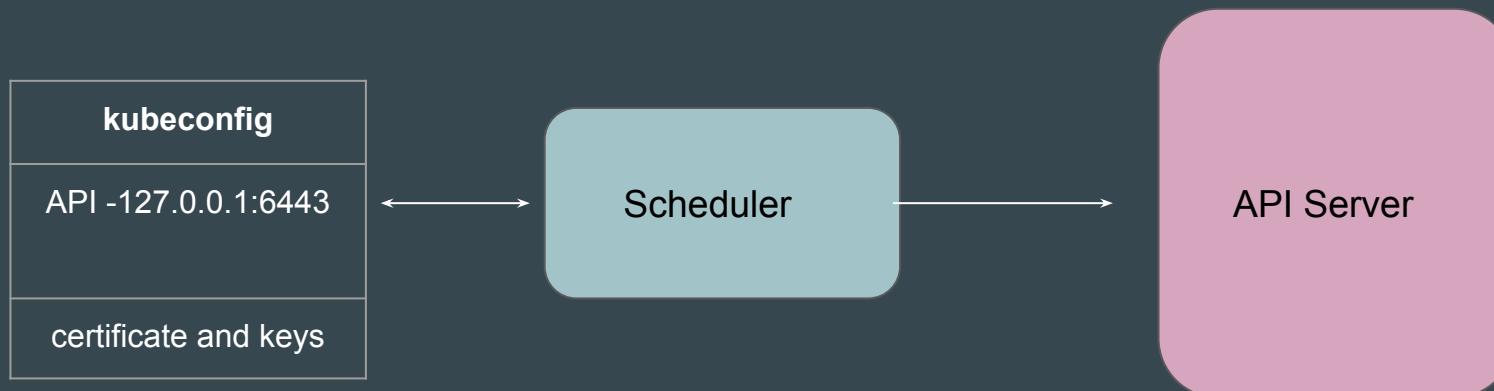
We will be generating a set of certificates and keys for each component as part of the cluster for secure communication.



# Common Pattern 2 - Kubeconfig

Multiple components need to interact with the API server.

For this, we will have to create a kubeconfig file for individual components.



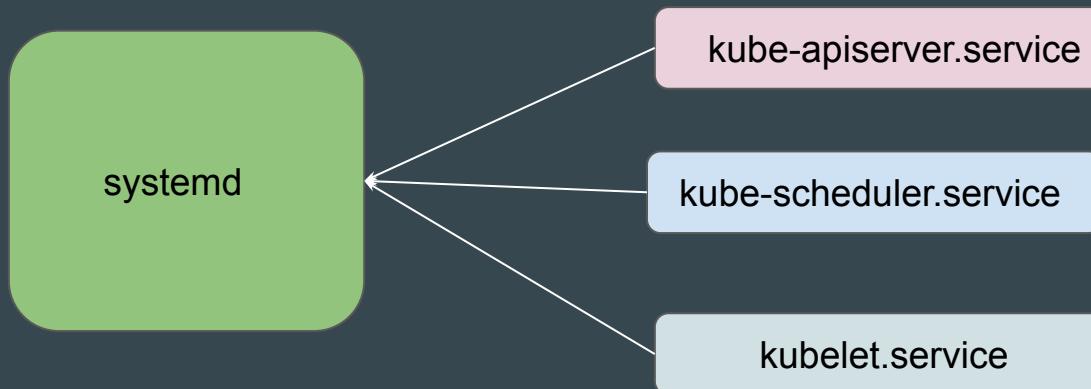
# Common Pattern 3 - Flags

Each component of a cluster will have a unique set of configuration options that control the functionality.



# Common Pattern 4 - Systemd File

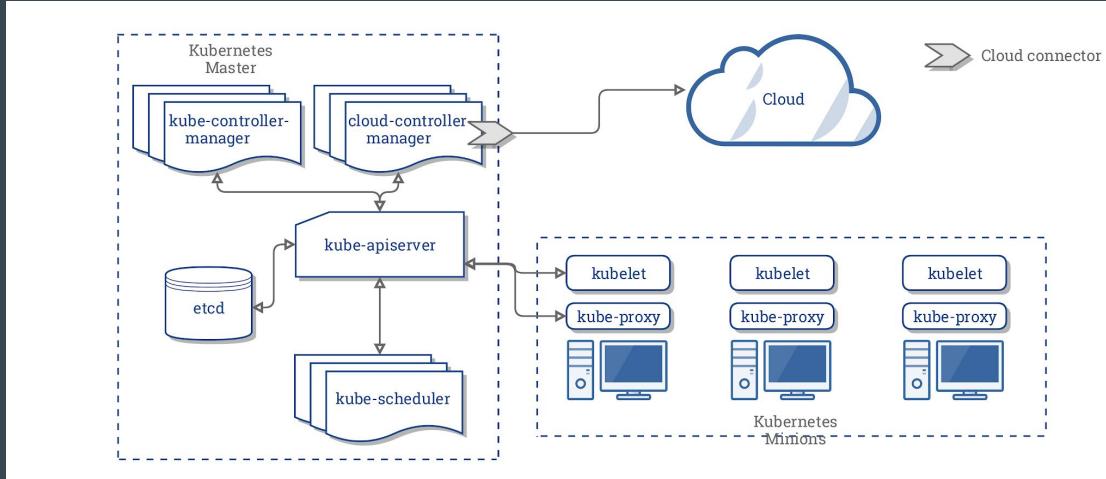
We will also create a systemd unit configuration file so that we can start and manage the component using systemd.



# **Kubernetes Component Binaries**

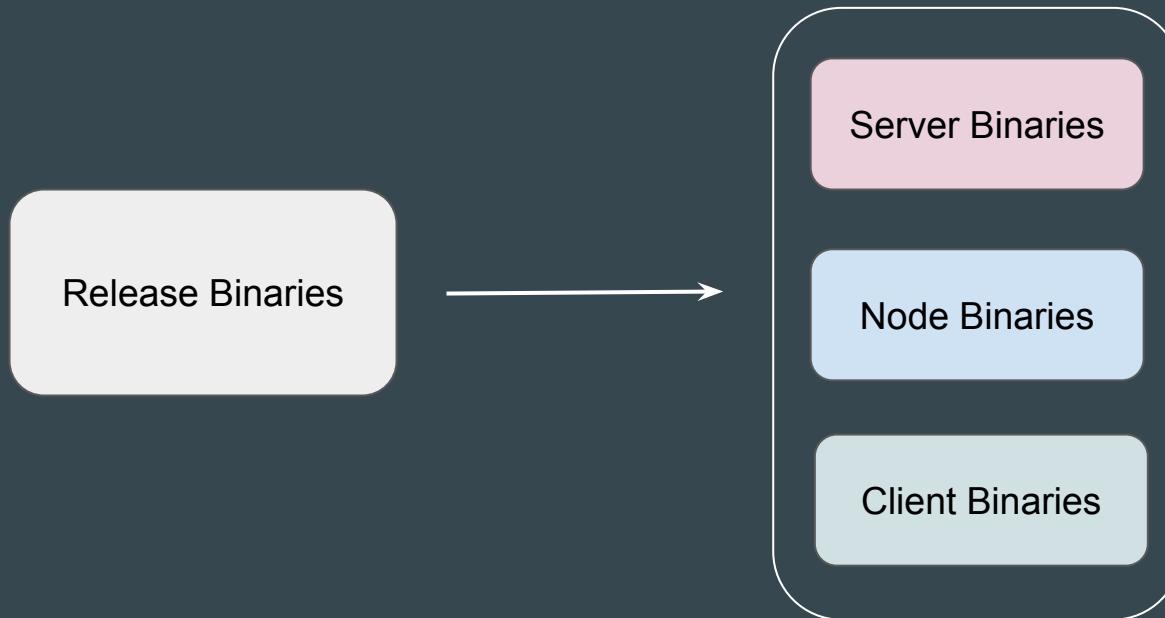
# Setting the Base

Before we start to setup Kubernetes cluster, we have to first download the binaries associated with all the components.



# Types of Binaries

Kubernetes Release Binaries are divided into following categories



# Types of Binaries

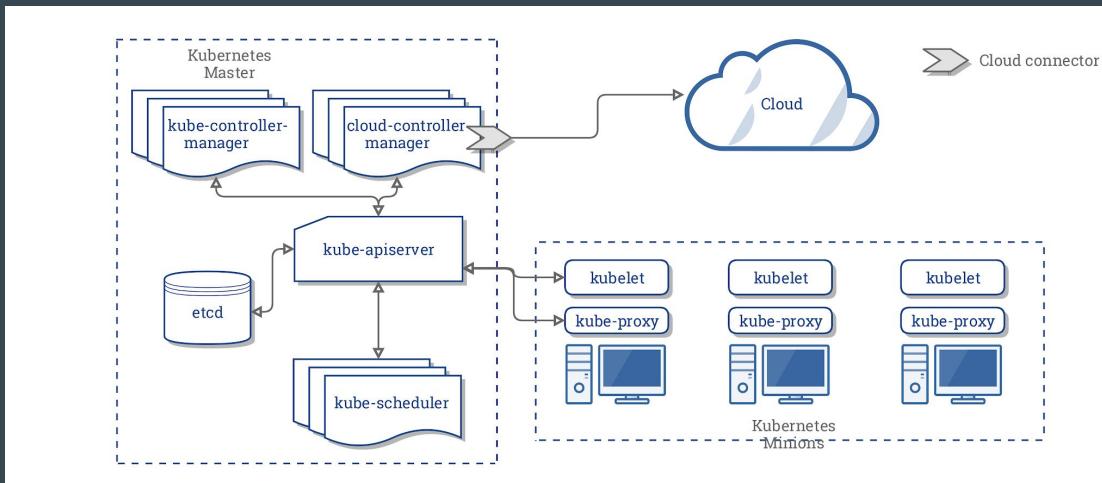
Release Binaries	Description
Server	Contains binaries for control plane components like API Server, Controller Manager, Scheduler and others.
Node	Containers Binaries required by worker nodes. kubelet, kube-proxy
Client	Binaries required by client like kubectl.

# **Certificate Authority**

# Setting the Base

Kubernetes components, such as the API server, kubelet, controller manager should communicate with each other over secure channels.

These components need a mechanism to verify each other's identity.



# Certificate Authority

Certificate Authority is an entity which issues digital certificates.

Key part is that both the receiver and the sender trusts the CA.

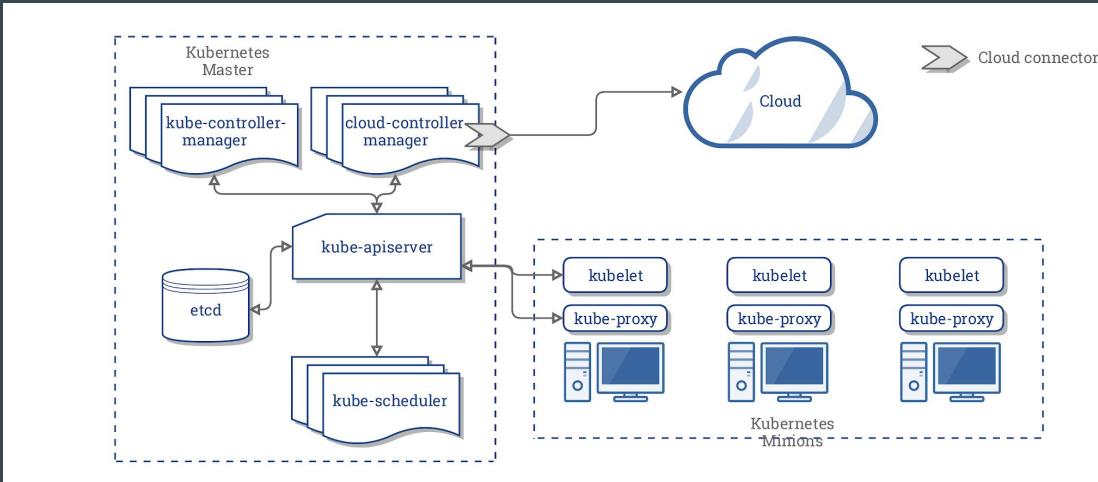


# Configuring ETCD

# Setting the Base

etcd reliably stores the configuration data of the Kubernetes cluster

We need to deploy ETCD before we deploy any component of Kubernetes.



# Remember the Certificate Creation Workflow

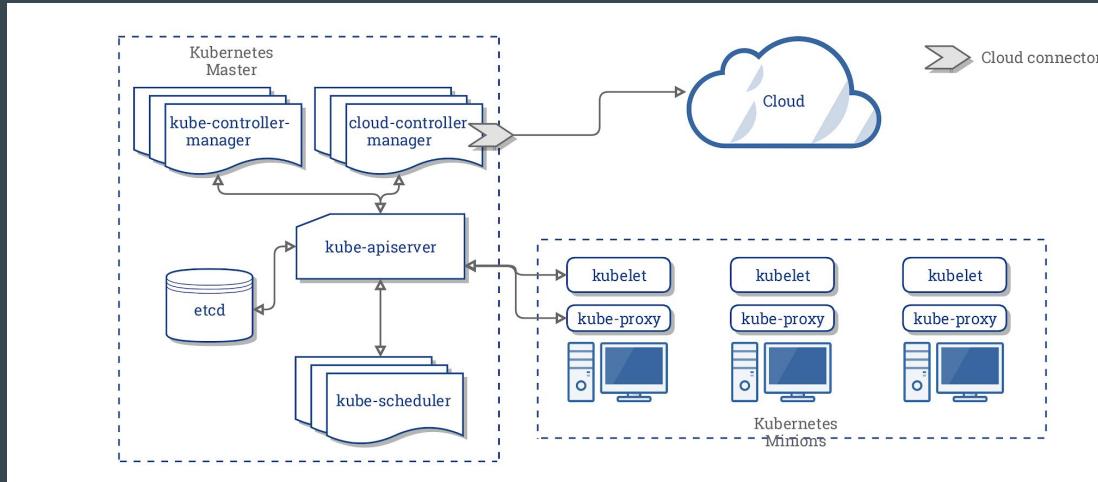


# **Overview of Configuring API Server**

# Setting the Base

**kube-apiserver** is the only component that interacts with the etcd.

While starting kube-apiserver, we have to supply etcd specific information.



# Important Flags

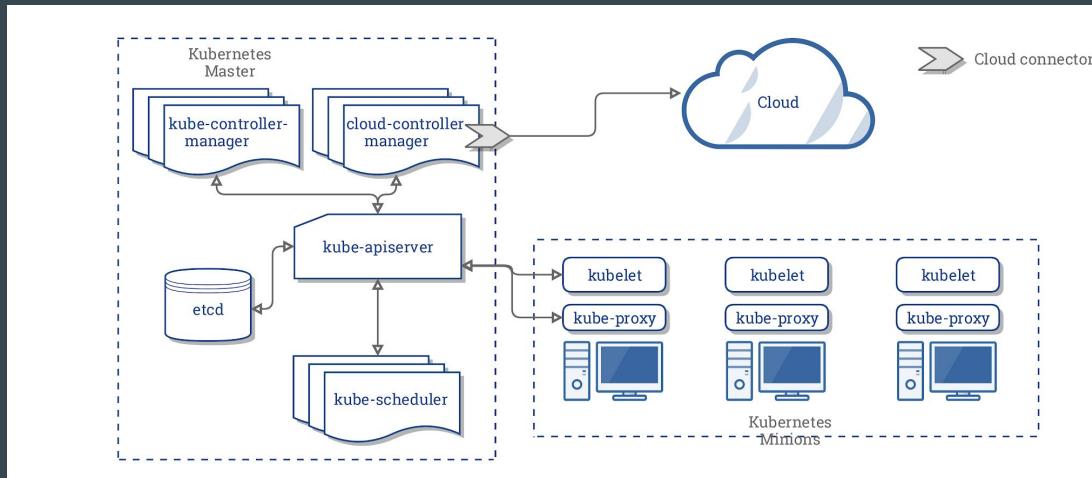
Options	Description
--etcd-servers	Specifies the address(es) of the etcd server(s) that the kube-apiserver should connect to.
--etcd-cafile	CA file used to secure etcd communication.
-etcd-certfile	Specifies the client certificate that the kube-apiserver should present to the etcd server for mutual TLS (mTLS) authentication.
--etcd-keyfile	Specifies the private key associated with the client certificate provided via --etcd-certfile.

# **Configure Controller Manager**

# Setting the Base

Controller Manager is an important component that is responsible for maintaining the desired state of your applications and infrastructure.

[ Node Controllers, Job Controllers, Service Account Controller etc ]

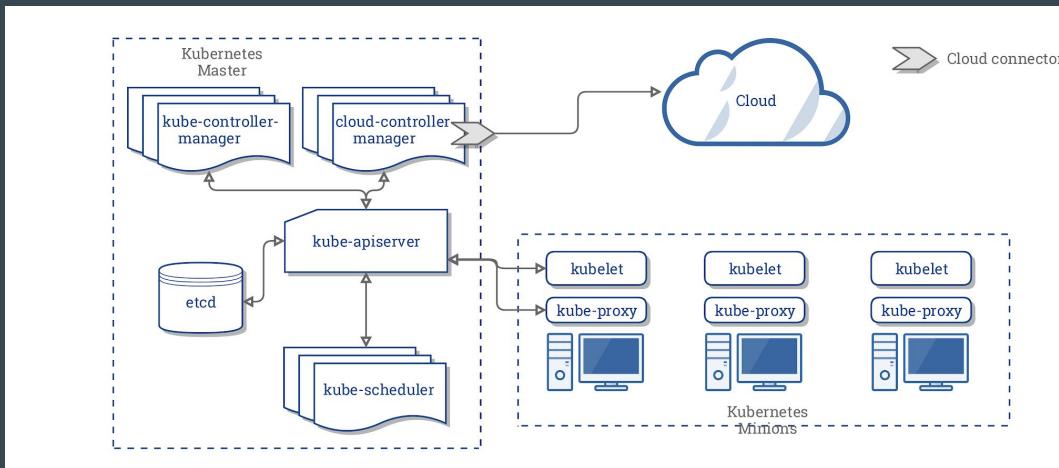


# **Configure Scheduler**

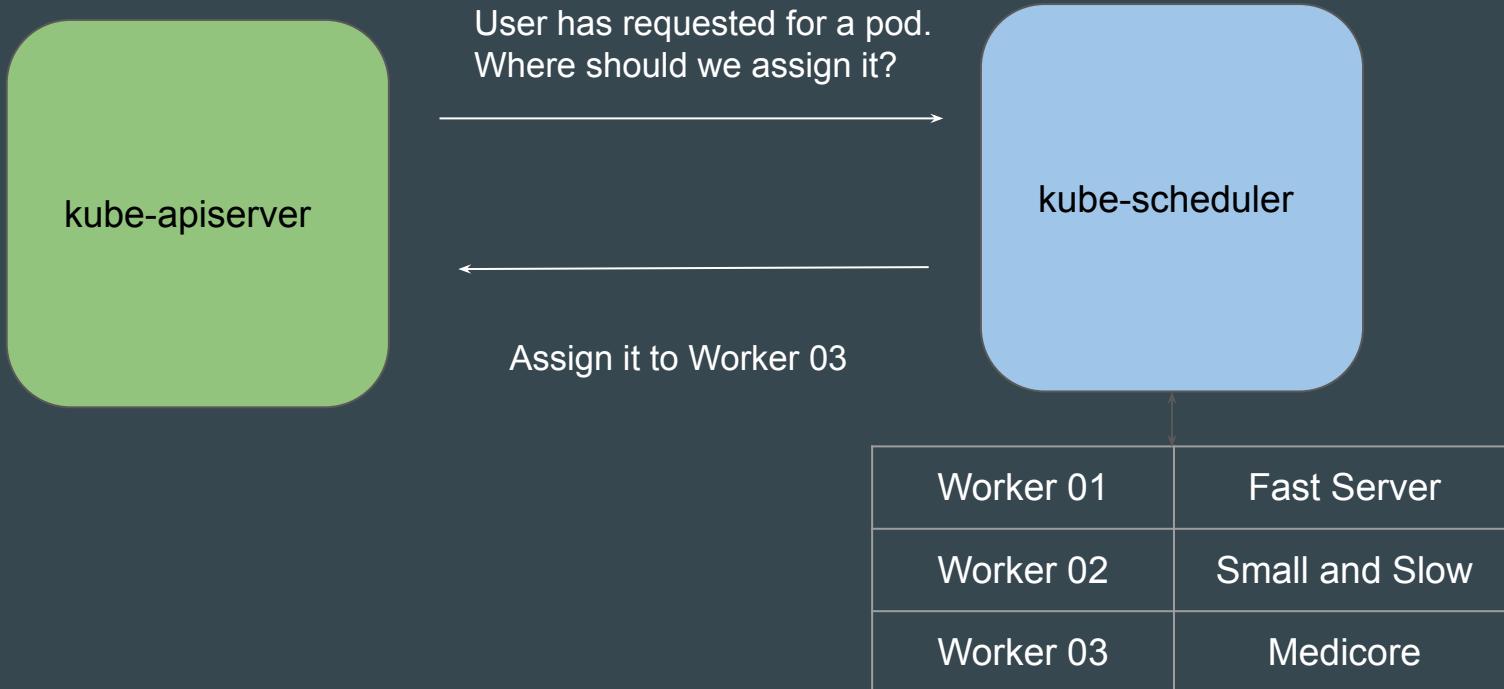
# Setting the Base

A scheduler watches for newly created Pods that have no Node assigned.

For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on.



# Revising the Workflow



# **Validate Cluster Status**

# Setting the Base

Once all important control plane components are configured, validate whether they run successfully.

```
root@control-plane:~# systemctl status kube-apiserver
● kube-apiserver.service - Kubernetes API Server
  Loaded: loaded (/etc/systemd/system/kube-apiserver.service; enabled; preset: enabled)
  Active: active (running) since Sun 2025-02-02 09:30:14 UTC; 17min ago
    Docs: https://github.com/kubernetes/kubernetes
    Main PID: 2871 (kube-apiserver)
       Tasks: 8 (limit: 2320)
      Memory: 158.5M (peak: 174.4M)
        CPU: 23.734s
      CGroup: /system.slice/kube-apiserver.service
              └─2871 /usr/local/bin/kube-apiserver --advertise-address=139.59.68.120 --allow
```

# Get Component Status

The command `kubectl get componentstatus` is used to retrieve information about the health of the core components of a Kubernetes control plane.

Note: The command is deprecated.

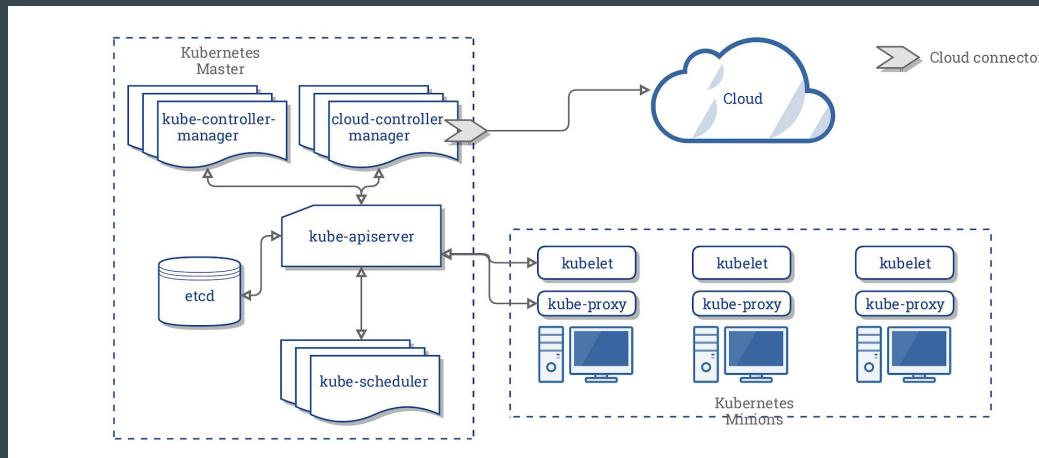
```
root@control-plane:~# kubectl get componentstatus
Warning: v1 ComponentStatus is deprecated in v1.19+
NAME          STATUS   MESSAGE    ERROR
scheduler     Healthy  ok
controller-manager  Healthy  ok
etcd-0        Healthy  ok
```

# **Worker Node Configuration**

# Setting the Base

While setting up worker node, there are three important component to configure:

1. kubelet
2. kube-proxy
3. Container Runtime



# Revising the Basics

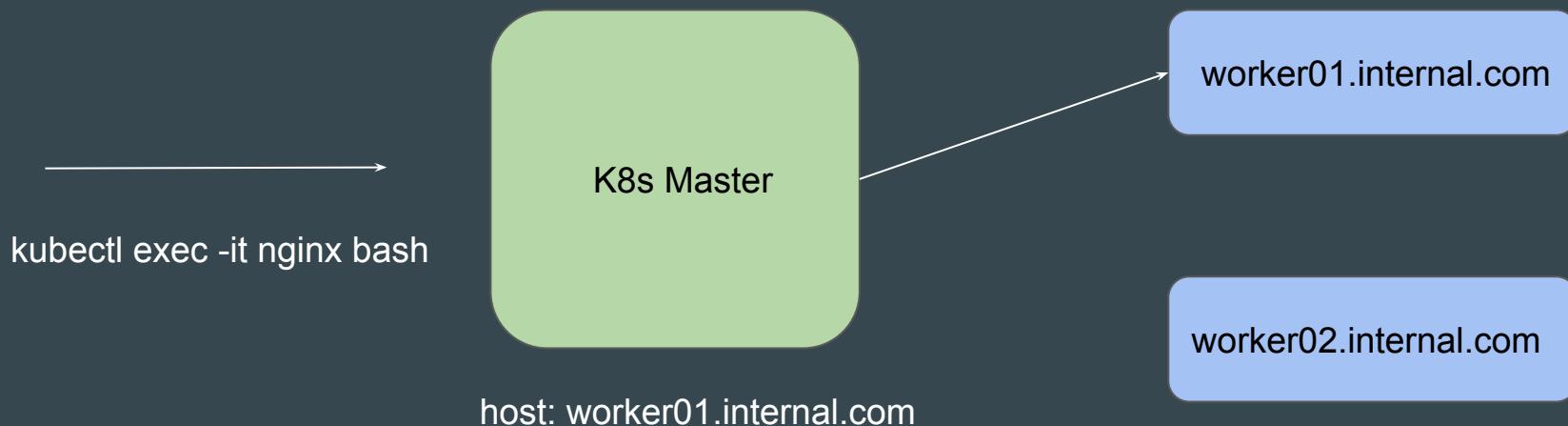
Component	Description
kubelet	An agent that runs on each node and ensures that containers are running
kube-proxy	Maintains network rules on nodes to enable network communication to Pods.
Container Runtime	Software responsible for running containers. Examples include Docker, containerd, and CRI-O.

# Kubelet Preferred Address Types

# Setting the Base

On `kubectl exec`, kube-apiserver initiate the connection with the kubelet running on the host.

In order to communicate, kube-apiserver uses ways like IP addresses, hostnames of the nodes.

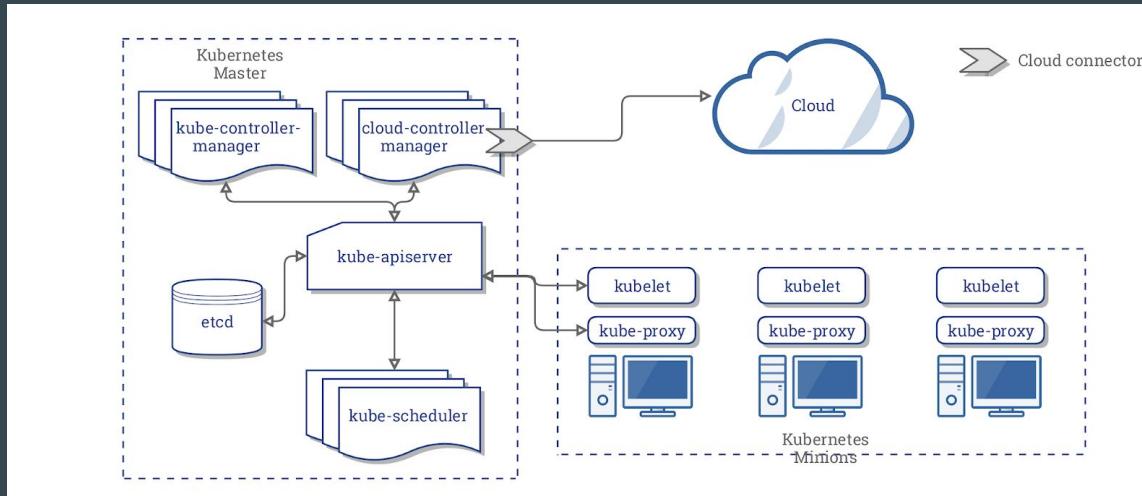


# **Breakdown Learning**

# Setting the Base

Each component in the Kubernetes cluster has an important role.

For testing, we will bring down certain components and see the results.



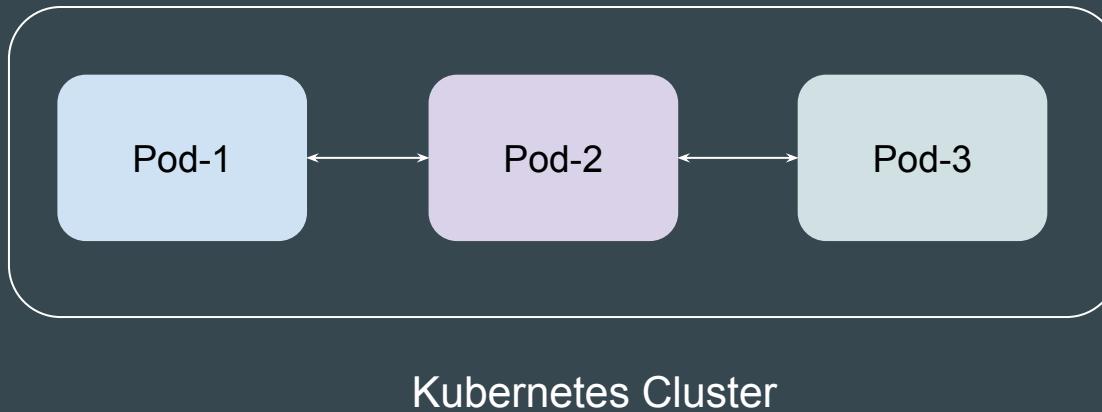
# Bring Down Following Components

Components	Description
Scheduler	Watches for newly created Pods with no assigned node, and selects a node for them to run on.
kubelet	Makes sure that containers are running in a Pod.
Controller Manager	Multiple Controller types, some include:  Service Account & Token controllers: Create default accounts and API access tokens for new namespaces

# **Overview of Network Policies**

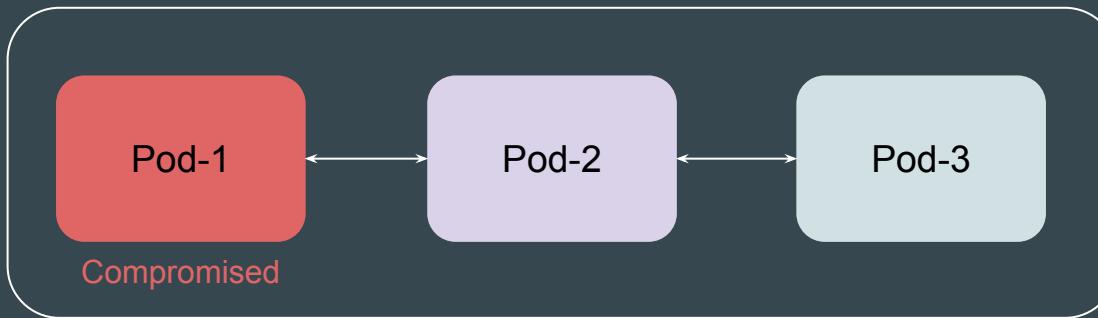
# Understanding the Basics

By default, Kubernetes **allows all traffic between pods within a cluster**. Network Policies help you lock down this open communication.



# Understanding the Challenge

If a application inside any Pod gets compromised, attacker can essentially communicate with all other Pods easily over the network.



Kubernetes Cluster

# Ideal Scenario

You only want Pods that have genuine requirement to connect to other pods to be able to communicate.



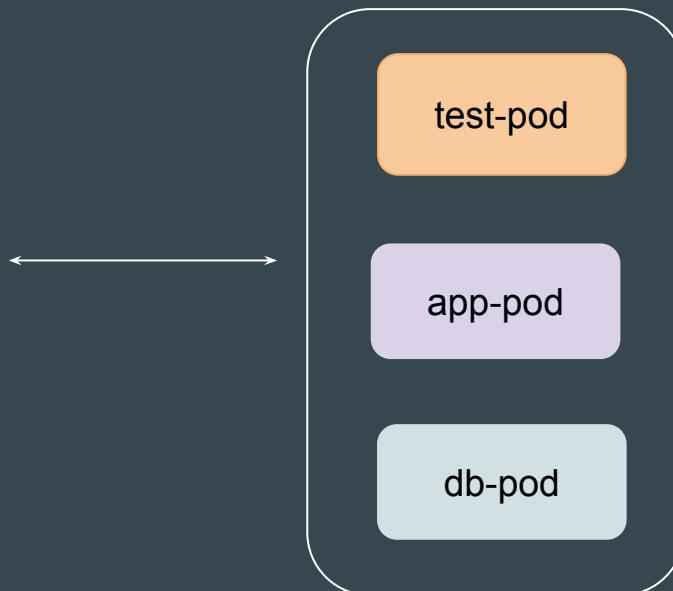
Kubernetes Cluster

# Introducing Network Policies

Network Policies are a **mechanism for controlling network traffic flow** in Kubernetes clusters.

Source	Destination	Effect
app-pod	db-pod	Allow
test-pod	ALL	Deny

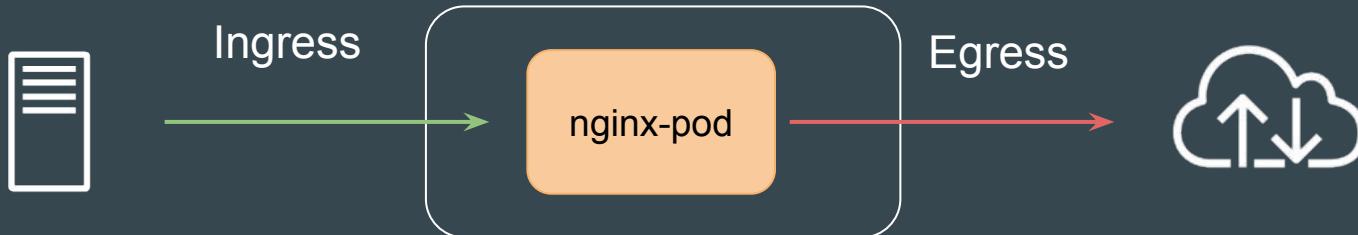
Network Policies



# Types of Rules

There are two types of rules supported as part of Network policies:

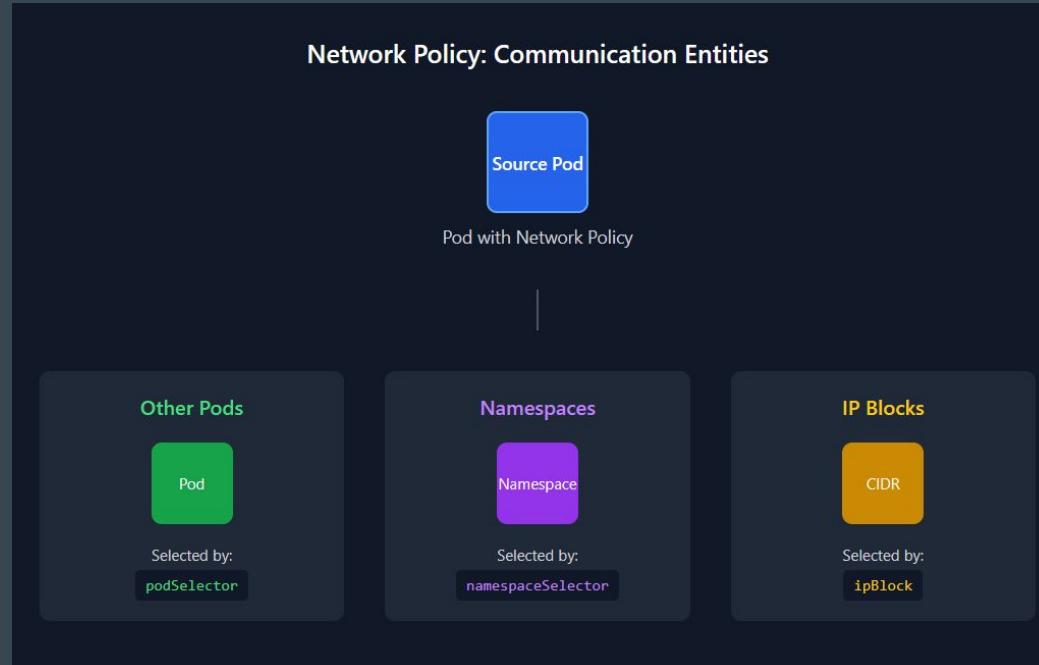
1. Ingress Rules (Inbound Rule)
2. Egress Rules (Outbound Rule)



# Supported Filtering Entities

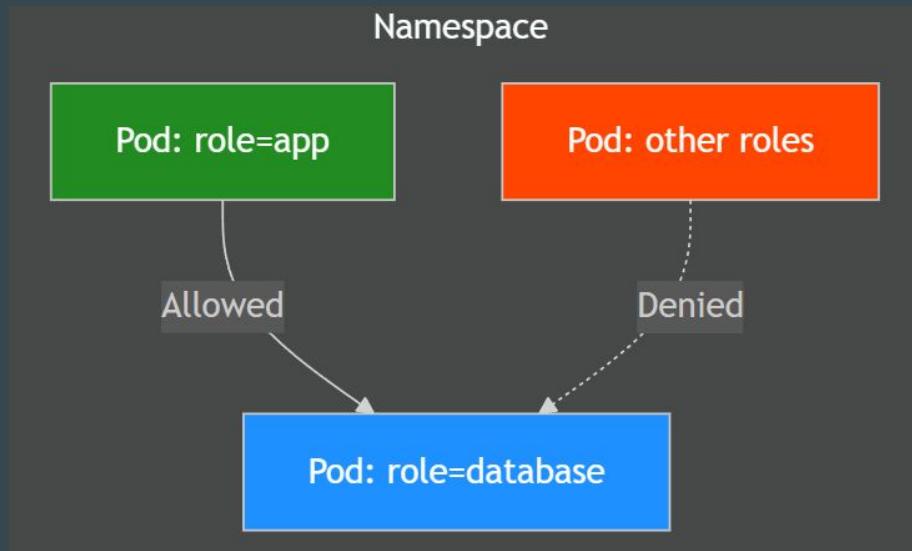
The entities that a Pod can communicate with are identified through a combination of the following three identifiers:

1. Other pods
2. Namespaces
3. IP Blocks



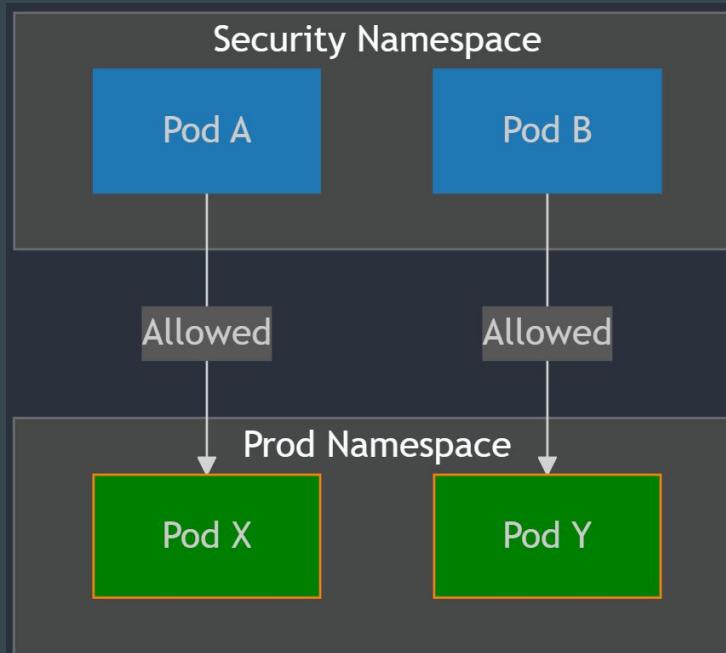
# Example 1 - Pod Selector

Allow pods with label of role=app to connect to pods with labels of role=database



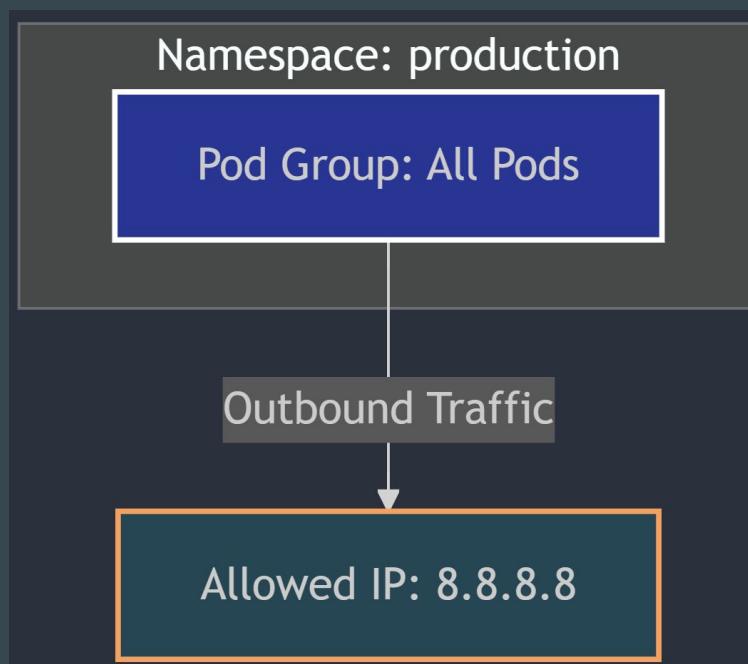
## Example 2 - NameSpace Selector

Allow Pods from Security namespace to connect to Pods in Prod namespace.



## Example 3 - ipBlock

Allow Pods from production namespace to connect to only 8.8.8.8 IP address outbound.



# Support for Network Policy

Not all Kubernetes network plugins (CNI) support NetworkPolicy.

The ability to enforce NetworkPolicies is a feature that must be implemented by the CNI plugin

Some Network Plugins like Calico, Cilium, etc supports Network policy.

Some Network plugins like kubenet, Flannel does NOT support network Policy

# Network Policy and Managed K8s Cluster

Most managed Kubernetes services (like AKS, EKS, GKE) come with a CNI that supports NetworkPolicy by default.

However, it's always a good idea to check the documentation for your specific service to confirm.

# **Structure of Network Policy**

# Sample Network Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

# Basic Mandatory Fields

As with all other Kubernetes config, a NetworkPolicy needs the following fields:

- apiVersion
- kind
- metadata

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
```

# Contents of Spec

NetworkPolicy spec has all the information needed to define a particular network policy in the given namespace.

- podSelector
- policyTypes
- Ingress
- egress

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

# 1 - Pod Selector

Each NetworkPolicy includes a podSelector which **selects the grouping of pods to which the policy applies.**

The example network policy applies to all pods that has label of env=production

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
```

## Point to Note

An empty podSelector selects all pods in the namespace.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector: {}
```

## 2 - Policy Types

Each NetworkPolicy includes a policyTypes list which may include either Ingress, Egress, or both.

The policyTypes field indicates whether or not the given policy applies to inbound traffic to selected pod, outbound traffic from selected pods, or both

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
```

## 3 - Ingress

Inside ingress, you can use various combinations of podSelector, nameSpace selector etc to define the rules.

Inbound traffic for Pods with label of env=production will be allowed from pods with label env=security.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            env: security
```

## 4 - Egress

Inside Egress rule, you can use various combinations of podSelector, namespaceselector, ipBlock etc to define the rules.

Allow Outbound Traffic only to IP address of 8.8.8.8 for Pods having label of env=production



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

# Role of from and to

<b>from</b>	<p>Used in an Ingress rule.</p> <p>Specifies the sources of incoming traffic allowed to the selected pods.</p> <p>Sources can be other pods, namespaces, or IP blocks.</p>
<b>to</b>	<p>Used in an Egress rule.</p> <p>Specifies the destinations of outgoing traffic allowed from the selected pods.</p> <p>Destinations can be other pods, namespaces, or IP blocks</p>

# **Practical - Network Policies**

# Example 1 - Block All Ingress and Egress

Since no specific rules are defined for ingress or egress, Kubernetes denies all traffic by default

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```



## Points to Note - podSelector

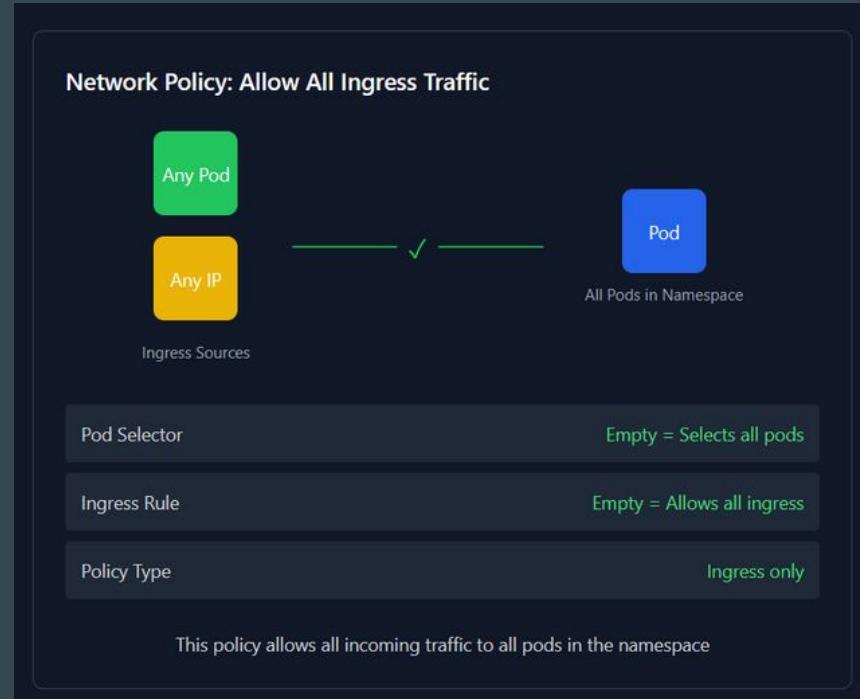
podSelector { }

This means the policy applies to all pods in the namespace because the selector is empty (matches all pods).

## Example 2 - Allow Ingress Traffic

This policy allows all incoming traffic (ingress) to the selected pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```



# Points to Note

ingress:

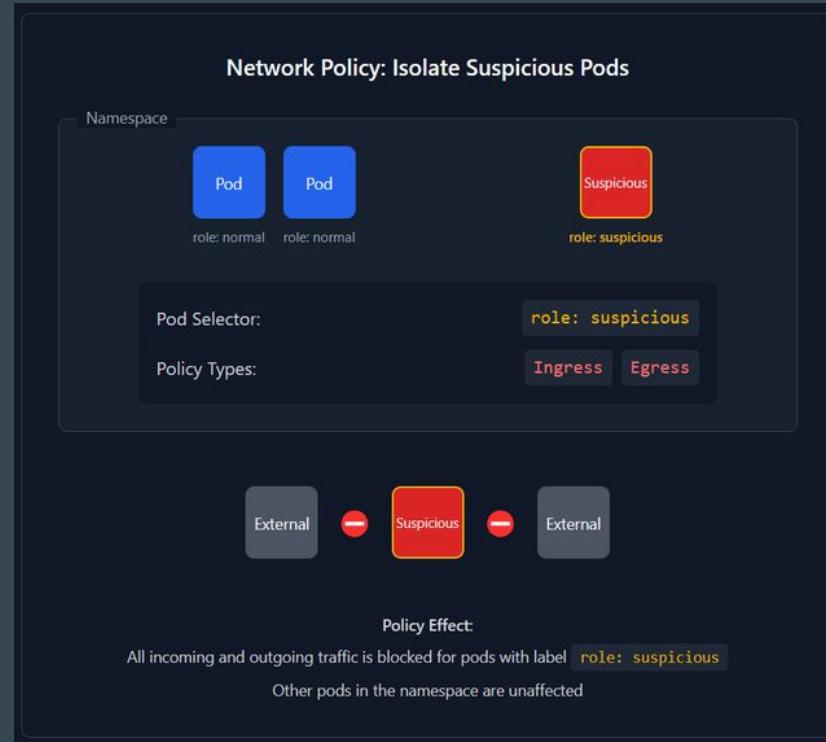
- { }

The empty {} means that there are no restrictions on the source of the traffic (any source is allowed).

# Example 3 - Isolate Suspicious Pod

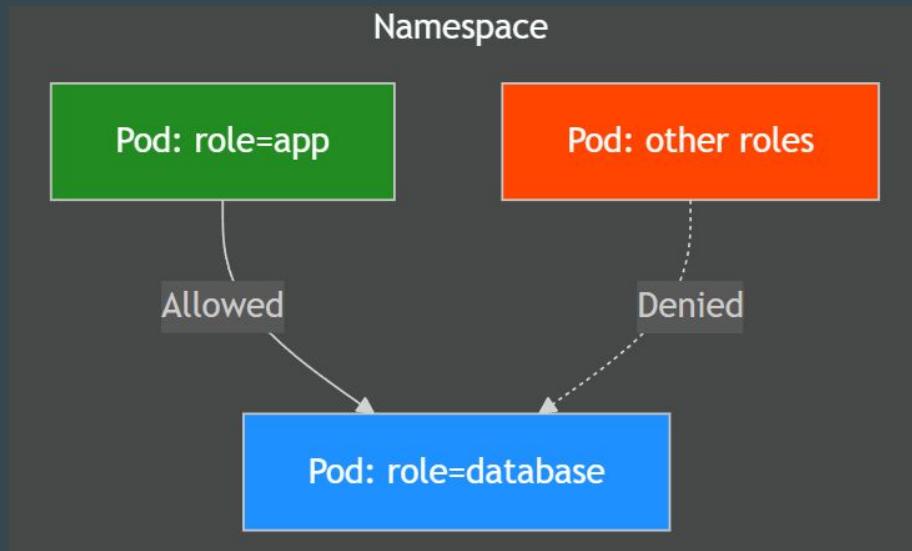
This policy blocks all ingress and egress access to pod with role=suspicious.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: suspicious-pod
spec:
  podSelector:
    matchLabels:
      role: suspicious
  policyTypes:
    - Ingress
    - Egress
```



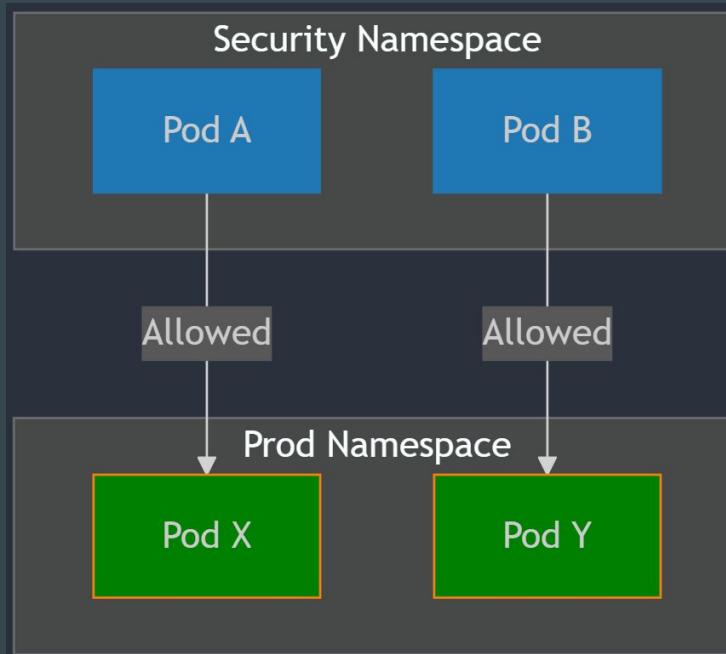
## Example 4 - PodSelector

Allow pods with label of role=app to connect to pods with labels of role=database



## Example 5 - Namespace Selector

Allow Pods from Security namespace to connect to Pods in Prod namespace.

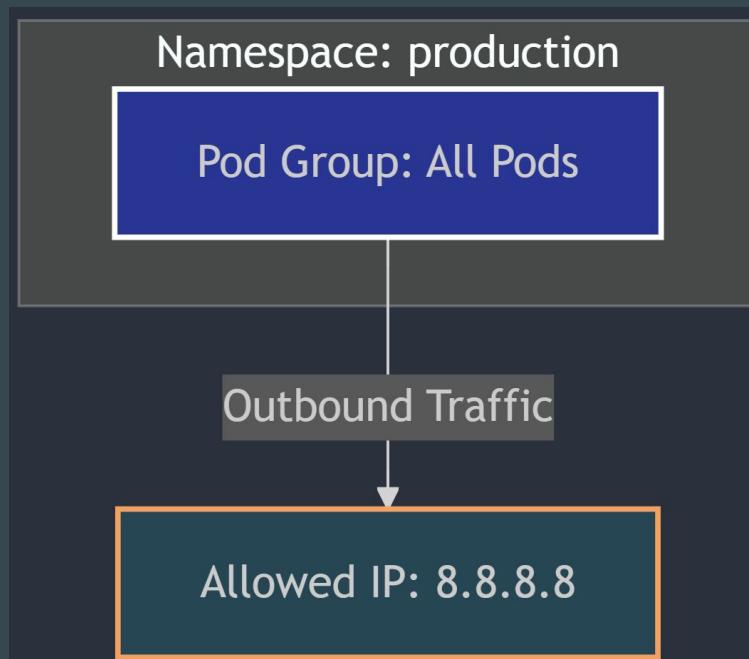


## Example 5 - Reference Code

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: namespace-eselector
  namespace: production
spec:
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: security
  policyTypes:
  - Ingress
```

## Example 6 - ipBlock

Allow Pods from production namespace to connect to only 8.8.8.8 IP address outbound.



## Example 6 - Reference Code

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: outbound-8888
spec:
  podSelector: {}
  egress:
  - to:
    - ipBlock:
        cidr: 8.8.8.8/32
  policyTypes:
  - Egress
```

## **Network Policies - Except, Port and Protocol**

# Except

The `except` field in a Kubernetes NetworkPolicy allows you to define exceptions to a broader rule.

Following policy allows ingress for cidr range of 172.17.0.0/16 except the range of 172.17.1.0/24

```
ingress:  
- from:  
  - ipBlock:  
    cidr: 172.17.0.0/16  
  except:  
  - 172.17.1.0/24
```

# Reference Screenshot - Entire Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: except-ingress
spec:
  podSelector:
    matchLabels:
      role: database
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
  policyTypes:
    - Ingress
```

# Ports and Protocol

When writing a NetworkPolicy, you can target a port or range of ports.

```
egress:  
  - to:  
    - ipBlock:  
        cidr: 10.0.0.0/24  
  ports:  
    - protocol: TCP  
      port: 32000  
      endPort: 32768
```

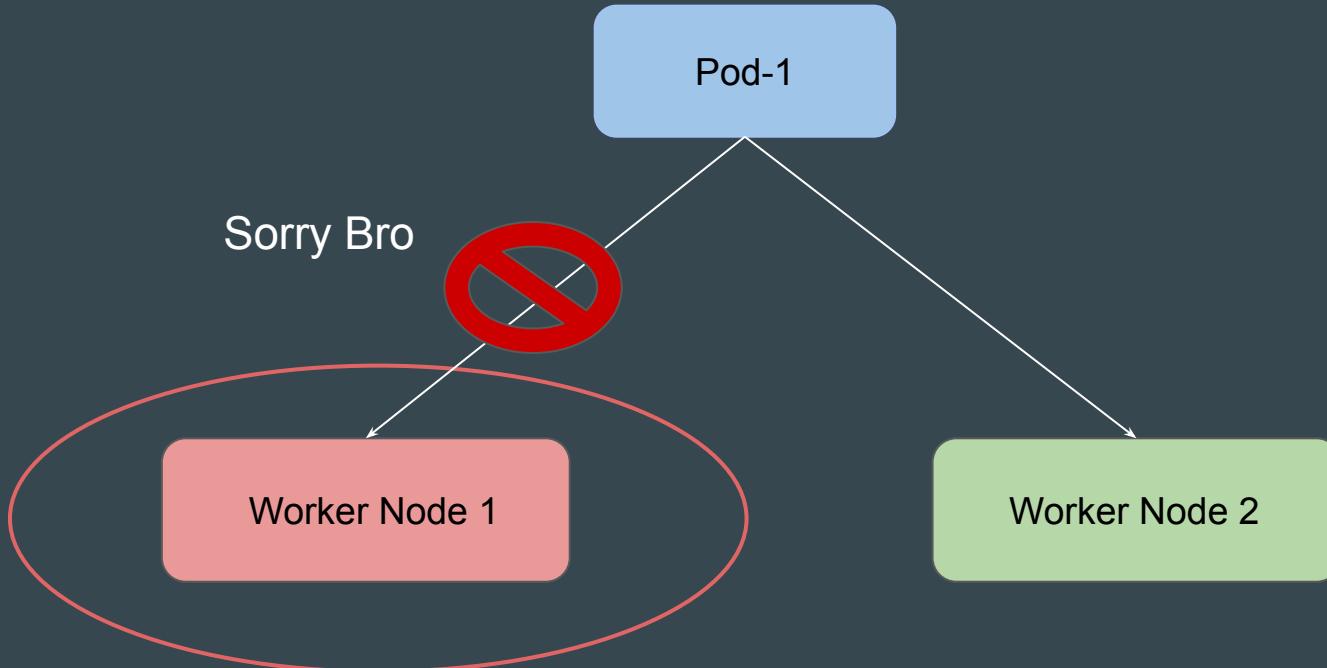
# Reference Screenshot - Entire Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: multi-port-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Egress
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 32000
      endPort: 32768
```

# **Taints and Tolerations**

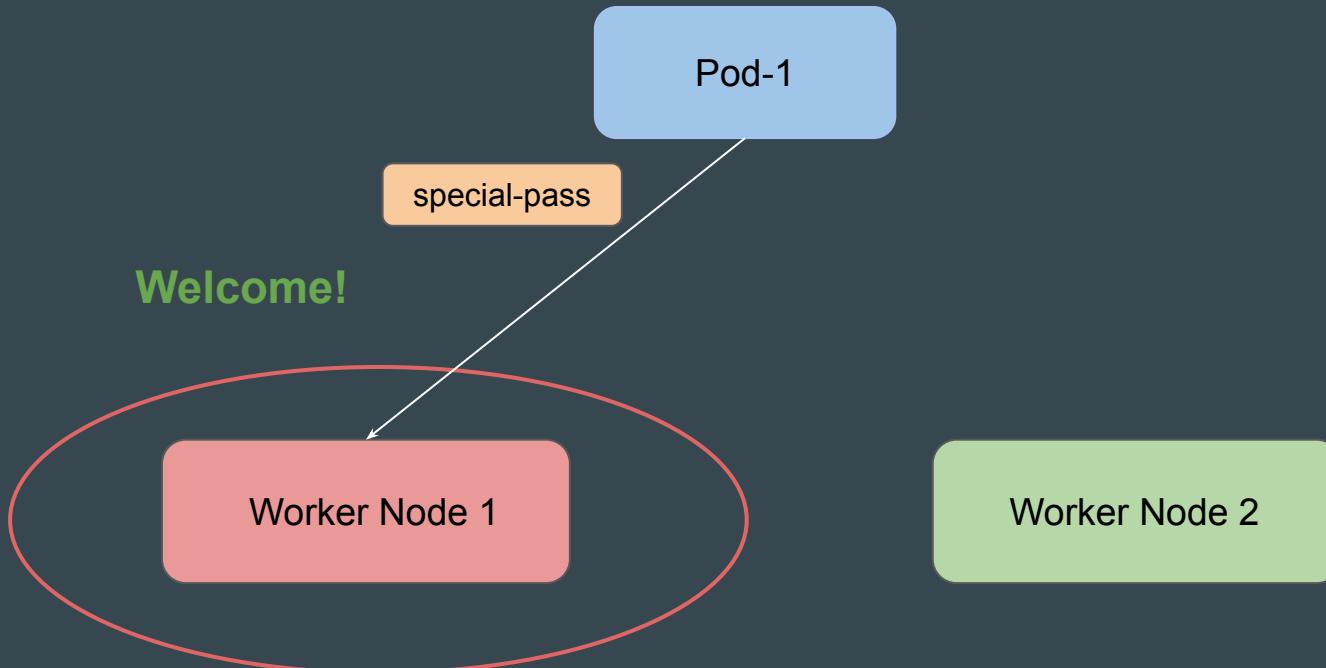
# Understanding Taint

Taint is a property added to a node that repels certain pods



# Understanding Tolerations

In order to schedule into the worker node with taint, you need a **special pass**. This special pass is called **Toleration**.



# Defining a Taint

You can use the `kubectl taint node` command to add a taint to a node.

A taint consists of a key, value (optional), and effect.

```
root@kubeadm-2:~# kubectl taint node worker-01 key=value:NoSchedule  
node/worker-01 tainted
```

# Effects in Taints

Effects	Description
NoSchedule	Prevents scheduling of new pods on the node unless they tolerate the taint.
PreferNoSchedule	Tries to avoid scheduling new pods on the node, but does not enforce it strictly.
NoExecute	Evicts existing pods and prevents new pods from being scheduled on the node.

# Defining Tolerations - Sample Manifest

```
! pod-tolerate.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx
  tolerations:
  - key: "example-key"
    operator: "Exists"
    effect: "NoSchedule"
```

# Defining Toleration - Structure

Component	Description	Important Pointers
key	The taint key that the toleration applies to. This is used to match a taint on a node.	
operator	Defines the relationship between the key and the value.	<ul style="list-style-type: none"><li>- <b>Equal</b>: The toleration matches if the key and value are equal.</li><li>- <b>Exists</b>: The toleration matches if the key exists, regardless of the value.</li></ul>
effect	Specifies the effect of the taint.	
value	The value associated with the key that the toleration applies to.	Any string value that matches the value of the taint

# Revising the Concept

Taints repel pods, and tolerations allow exceptions.

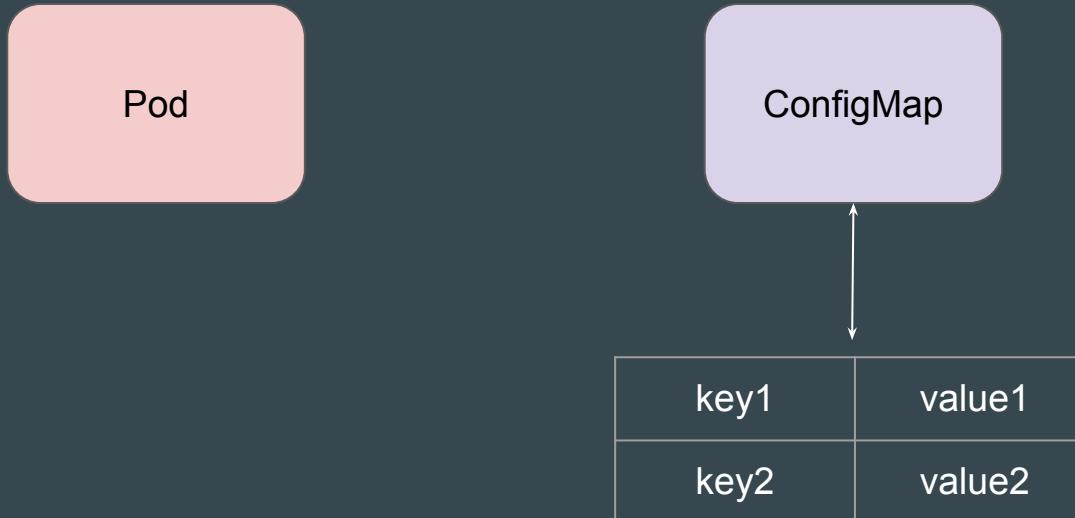
A pod without a toleration for a node's taint will not be scheduled on that node.

Taints are applied to nodes, while tolerations are applied to pods.

# **Editing Existing Kubernetes Resources**

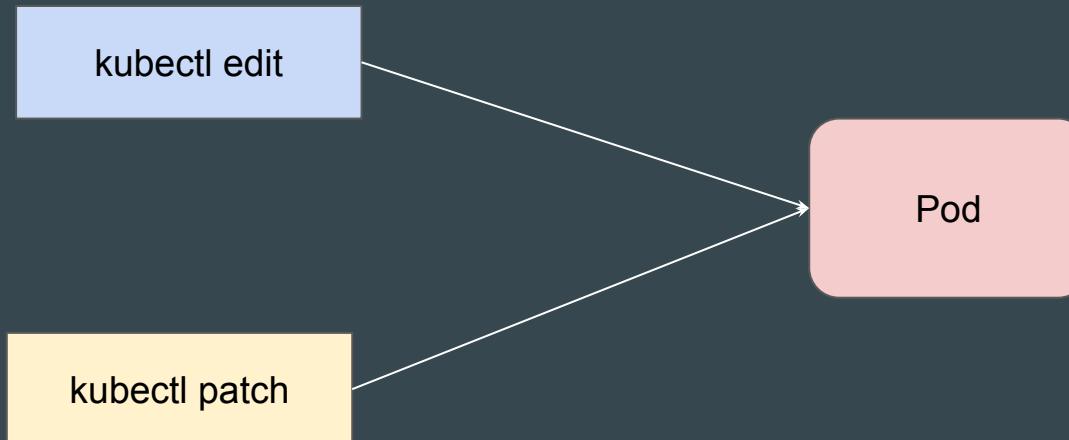
# Setting the Base

You **might need to modify the configuration of existing Kubernetes objects** (such as Pods, Secrets, ConfigMaps) after their creation.



# Two Primary Ways to Edit

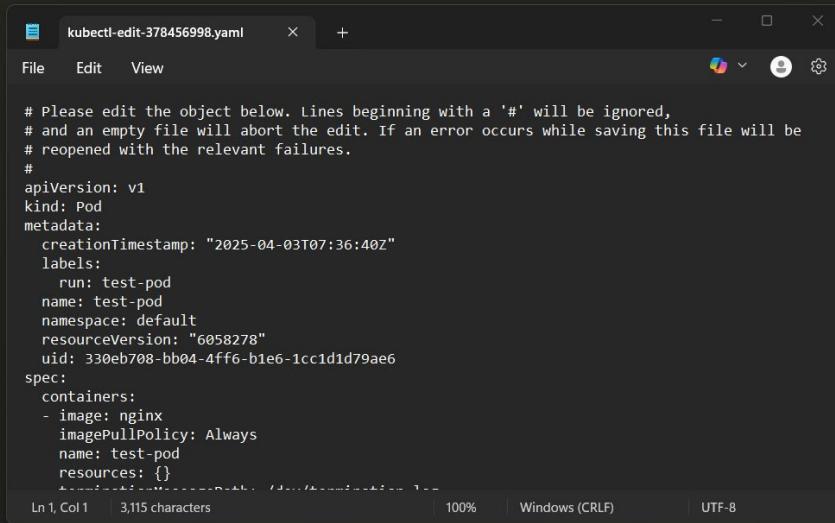
There are two primary ways to edit a resource in Kubernetes.



# Approach 1 - kubectl edit

**kubectl edit** command opens the resource manifest in your default editor (like vim), allowing you to manually edit fields in the YAML definition.

```
C:\kplabs-k8s>kubectl edit pod test-pod
```



A screenshot of a terminal window on a Windows system. The command `kubectl edit pod test-pod` has been run, and the output shows the YAML configuration for a pod named `test-pod`. The configuration includes metadata like creation timestamp, labels, and a single container using the `nginx` image. The terminal window title is `kubectl-edit-378456998.yaml`.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-04-03T07:36:40Z"
  labels:
    run: test-pod
  name: test-pod
  namespace: default
  resourceVersion: "6058278"
  uid: 330eb708-bb04-4ff6-b1e6-1cc1d1d79ae6
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: test-pod
    resources: {}
```

Ln 1, Col 1 | 3,115 characters | 100% | Windows (CRLF) | UTF-8

## Approach 2 - kubectl patch

kubectl patch allows you to update an object non-interactively.

```
root@kubeadm:~# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
test-pod  1/1     Running   0          19m   run=test-pod
root@kubeadm:~# kubectl patch pod test-pod -p '{"metadata":{"labels":{"run":"dev-pod"}}}'
pod/test-pod patched
root@kubeadm:~# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
test-pod  1/1     Running   0          19m   run=dev-pod
```

# Point to Note

Not all the field of a object can be edited.

Depending on the resource, these can be different.

<b>Mutable Field</b>	<b>Immutable Field</b>
metadata.labels	spec.containers[*].image
metadata.annotations	spec.containers[*].name
spec.containers[*].resources	metadata.name
spec.containers[*].env	spec.restartPolicy

## **Capacity, Allocatable, and Allocated**

# Setting the Base

The output of `kubectl describe node <node-name>` provides a detailed view of the node's capacity, allocatable resources, and how much of those resources are currently requested or limited by running Pods.

```
Capacity:  
  cpu:          1  
  ephemeral-storage: 51432064Ki  
  hugepages-2Mi:    0  
  memory:        2014392Ki  
  pods:          110  
  
Allocatable:  
  cpu:           900m  
  ephemeral-storage: 47399790104  
  hugepages-2Mi:    0  
  memory:         1574Mi  
  pods:          110
```

# 1 - Capacity

This represents the **total amount of resources physically available on the node hardware itself**.

```
Capacity:  
  cpu:           1  
  ephemeral-storage: 51432064Ki  
  hugepages-2Mi:    0  
  memory:         2014392Ki  
  pods:           110
```

Resource	Meaning
cpu: 1	The node has 1 CPU core (or virtual CPU).
memory: 2014392Ki	~1.92 GiB of total memory (RAM).
ephemeral-storage: 51432064Ki	~49 GB of local ephemeral storage (temporary disk storage).

## 2 - Allocatable

This section shows the **resources available for Pods** after subtracting what the system (OS, kubelet, etc.) reserves

```
Allocatable:  
cpu: 900m  
ephemeral-storage: 47399790104  
hugepages-2Mi: 0  
memory: 1574Mi
```

Resource	Meaning
cpu: 900m	900 milliCPU (0.9 CPU) available for pods.
ephemeral-storage	~44.1 GB is available to pods.
memory: 1574Mi	~1.54 GiB available to pods (rest is used by system daemons).

# Allocated Resources

This section shows how much of the allocatable resources are currently requested or limited by running pods

Allocated resources:		
(Total limits may be over 100 percent, i.e., overcommitted.)		
Resource	Requests	Limits
-----	-----	-----
cpu	702m (78%)	0 (0%)
memory	720Mi (45%)	640Mi (40%)
ephemeral-storage	0 (0%)	0 (0%)
hugepages-2Mi	0 (0%)	0 (0%)

# Check Total Request and Limits at Pod Level

You can also check how much request each pod in the cluster has used.

NAMESPACE	POD	CPU_REQUEST	MEMORY_REQUEST
kube-system	cilium-8gmj5	300m	300Mi
kube-system	coredns-8655d79b9d-85hk5	100m	70Mi
kube-system	coredns-8655d79b9d-d9hnq	100m	70Mi
kube-system	cpc-bridge-proxy-ebpf-xtlf4	100m	75Mi
kube-system	csi-do-node-kxqbs	<none>	<none>
kube-system	do-node-agent-vbp4b	102m	80Mi
kube-system	hubble-relay-cd64b5586-9vf2x	<none>	<none>
kube-system	hubble-ui-b79497598-bxq9w	<none>	<none>
kube-system	konnectivity-agent-jj5ln	<none>	<none>
kube-system	kube-proxy-ebpf-8z6gk	<none>	20Mi

# **Exercise - Requests and Limits**

# Setting the Base

Create a Pod that requests all the available CPU and memory resources on a specific worker node

Capacity:

```
cpu:          1
ephemeral-storage: 51432064Ki
hugepages-2Mi:    0
memory:         2014392Ki
pods:           110
```

Allocatable:

```
cpu:          900m
ephemeral-storage: 47399790104
hugepages-2Mi:    0
memory:        1574Mi
pods:           110
```

Allocated resources:

```
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource      Requests     Limits
-----       -----
cpu            702m (78%)   0 (0%)
memory         720Mi (45%)  640Mi (40%)
ephemeral-storage 0 (0%)    0 (0%)
hugepages-2Mi   0 (0%)    0 (0%)
```

# Base Analysis

Total Allocatable CPU	900m (900 millicores or 0.9 CPU cores)
Total Allocatable Memory:	1574Mi (1574 Mebibytes)
CPU Already Requested by Existing Pods:	702m (from the Allocated resources: section)
Memory Already Requested by Existing Pods:	720Mi (from the Allocated resources: section)

# Setting the Calculations

Available CPU for Requests

$$\begin{aligned} \text{Allocatable CPU - Requested CPU} \\ = 900m - 702m = 198m \end{aligned}$$

Available Memory for Requests:

$$\begin{aligned} \text{Allocatable Memory - Requested Memory} \\ = 1574Mi - 720Mi = 854Mi \end{aligned}$$

## Final Conclusion

Any new pod you try to schedule on this node must request CPU less than or equal to 198m AND memory less than or equal to 854Mi

If a pod requests more than either of these available amounts, the Kubernetes scheduler will not be able to place it on this node, and the pod will remain in a Pending state.

```
apiVersion: v1
kind: Pod
metadata:
  name: fit-pod
spec:
  containers:
    - name: successful-container
      image: nginx
      resources:
        requests:
          memory: "854Mi"
          cpu: "198m"
        limits:
          memory: "854Mi"
          cpu: "198m"
```

# Important Pointer

Question: Can we launch new Pod after 100% of Request (CPU and Memory) is reached?

Answer:

Yes. A pod without any resources requests is assumed to request 0 CPU and 0 memory.

Although the pod gets scheduled, it can be evicted or throttled if the node becomes resource-starved. This is because it has no guaranteed resources.

# **JSONPath**

# Setting the Base

JSONPath is a **query language for JSON**, similar to XPath for XML.

It allows you to select and extract specific data from a JSON document.

The screenshot shows a JSONPath query editor interface. At the top, there are tabs for "RFC 9535" and "RFC 9535 - JSONPath: Query Expressions for JSON". Below that is a dropdown menu set to "RFC 9535" and a search bar containing the query `$.employees[*].name`. The main area is divided into two sections: "Document" and "Evaluation Results".

**Document:**

```
1 {  
2   "employees": [  
3     {  
4       "id": 1,  
5       "name": "Alice Johnson",  
6       "age": 30,  
7       "department": {  
8         "name": "Engineering",  
9         "location": "Building A"  
10      },  
11      "skills": ["JavaScript", "Python", "AWS"]  
12    },  
13    {  
14      "id": 2,  
15      "name": "Bob Smith",  
16      "age": 45,  
17      "department": {  
18        "name": "Human Resources",  
19        "location": "Building B"  
20      },  
21      "skills": ["Recruiting", "Conflict Resolution"]  
22    }  
23  ]  
24 }
```

**Evaluation Results:**

```
1 [  
2   "Alice Johnson",  
3   "Bob Smith"  
4 ]
```

# Basic JSONPath Syntax

Expressions	Meaning
\$	The root object or array.
.	Child Operator
[index1, index2]	Selects array elements with the specified indexes. Returns a list.
*	Wildcard selects all elements in an object or an array

# Base Scenario

We will create 2 pods and we will use jsonpath to query and filter various fields.

```
C:\kplabs-k8s>kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
pod-1    1/1       Running   0          52m
pod-2    1/1       Running   0          52m
```

# First Step for Analysis and Testing

First, we need the data in JSON format to examine its structure before working on jsonpath expressions.

```
C:\kplabs-k8s>kubectl get pods -o json
{
    "apiVersion": "v1",
    "items": [
        {
            "apiVersion": "v1",
            "kind": "Pod",
            "metadata": {
                "creationTimestamp": "2025-04-07T13:27:07Z",
                "labels": {
                    "run": "pod-1"
                },
                "name": "pod-1",
                "namespace": "default",
                "resourceVersion": "7275874",
                "uid": "993b9cef-f693-464a-bc5c-09af887adbfc"
            },
            "spec": {
                "containers": [
                    {
                        "image": "nginx",
                        "imagePullPolicy": "Always",
                        "name": "pod-1",
                        "resources": {},
                        "terminationMessagePath": "/dev/termination-log",
                        "terminationMessagePolicy": "File",
                        "volumeMounts": [
                            {
                                "mountPath": "/etc/nginx/html",
                                "name": "nginx-volume"
                            }
                        ]
                    }
                ],
                "dnsPolicy": "ClusterFirstWithHostNet",
                "restartPolicy": "Always",
                "schedulerName": "default-scheduler"
            }
        }
    ],
    "version": "v1"
}
```

# Output using JSONPATH

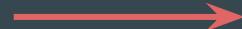
Filter data using relevant JSONPath expressions.

```
root@kubeadm:~# kubectl get pods -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.podIP}{"\n"}{end}'  
pod-1  192.168.45.255  
pod-2  192.168.45.199
```

JSONPath Expressions	Description
\$	\$ refers to the root object of the JSON document.
\$.items[*].metadata.name	List all pod names
\$.items[*].spec.containers[*].image	Get container images for all pods
\$.items[0].metadata.namespace	Get the namespace of the first Pod in the list
\$.items[*].spec.nodeName	Get the node name of all pods
\$.items[*].spec.containers[*].name	Get all container names

# The Base Structure

```
{  
  "apiVersion": "v1",  
  "items": [  
    { ...pod-1... },  
    { ...pod-2... }  
,  
  "kind": "List",  
  "metadata": {  
    "resourceVersion": ""  
  }  
}
```



```
{  
  "apiVersion": "v1",  
  "kind": "Pod",  
  "metadata": { ... },  
  "spec": { ... },  
  "status": { ... }  
}
```

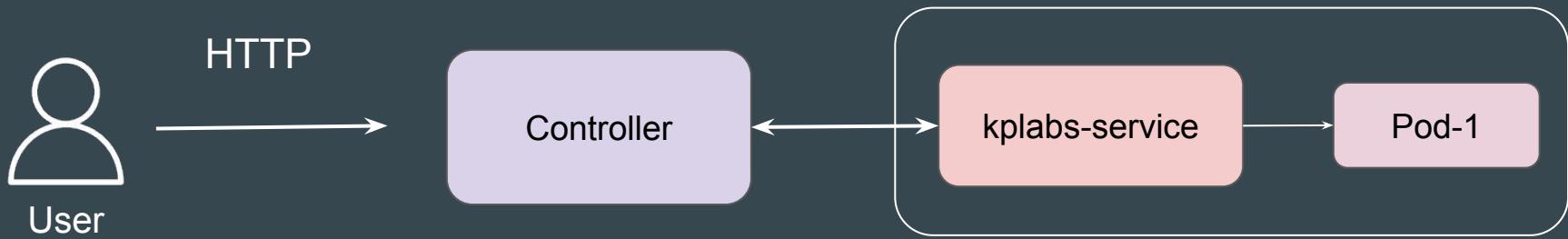
Pod Structure

Document Structure

# Ingress with TLS

# Understanding the Challenge

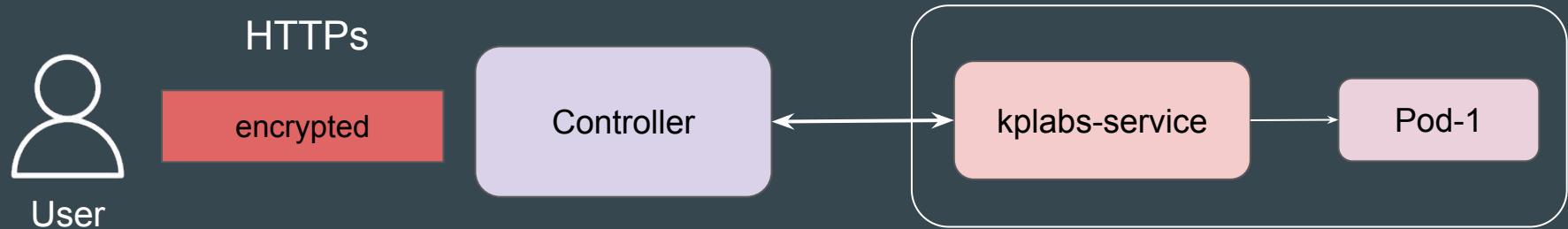
A HTTP based connection to the ingress controller is not a secure.



# Ingress with TLS

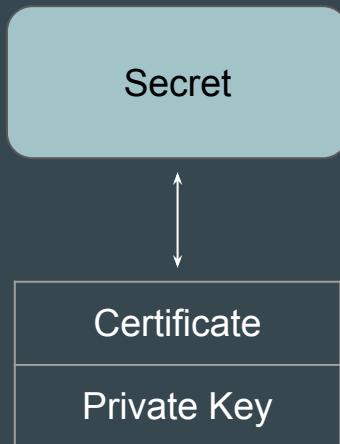
TLS ensures **secure communication** between the client and the server

You can secure the connection by setting up TLS at Ingress level.



# Point to Note

The certificates are stored in Kubernetes as **Secrets**, and the Ingress resource is configured to use these secrets for HTTPS traffic.



```
root@kubeadm:~# kubectl describe ingress tls-ingress
Name:          tls-ingress
Labels:        <none>
Namespace:    default
Address:
Ingress Class:  nginx
Default backend: <default>
TLS:
  tls-cert terminates demo.kplabs.in
Rules:
  Host            Path  Backends
  ----           ----  -----
  demo.kplabs.in      /   example-service:80 (192.168.45.195:80)
```

# Reference Screenshot

```
root@kubeadm:~# kubectl describe secret tls-cert
Name:         tls-cert
Namespace:    default
Labels:       <none>
Annotations: <none>
Type:        kubernetes.io/tls

Data
=====
tls.crt:  2839 bytes
tls.key:  241 bytes
```

```
root@kubeadm:~# kubectl describe ingress tls-ingress
Name:           tls-ingress
Labels:         <none>
Namespace:      default
Address:
Ingress Class: nginx
Default backend: <default>
TLS:
    → tls-cert terminates demo.kplabs.in
Rules:
  Host          Path  Backends
  ----          ---   -----
  demo.kplabs.in
                           /   example-service:80 (192.168.45.195:80)
```

# **Practical - Ingress with TLS**

# High-Level Steps

1. Create a TLS Certificate for your domain.
2. Create a Secret and store certificate and key.
3. Create Ingress Resource that references the Secret for TLS termination
4. Access your application over HTTPS

# **Container Runtime Interface (CRI)**

# Setting the Base

During its early development, Kubernetes had a tight coupling with Docker.

In early versions of Kubernetes, the kubelet interacted with containers primarily through the Docker Engine.



# Setting the Base

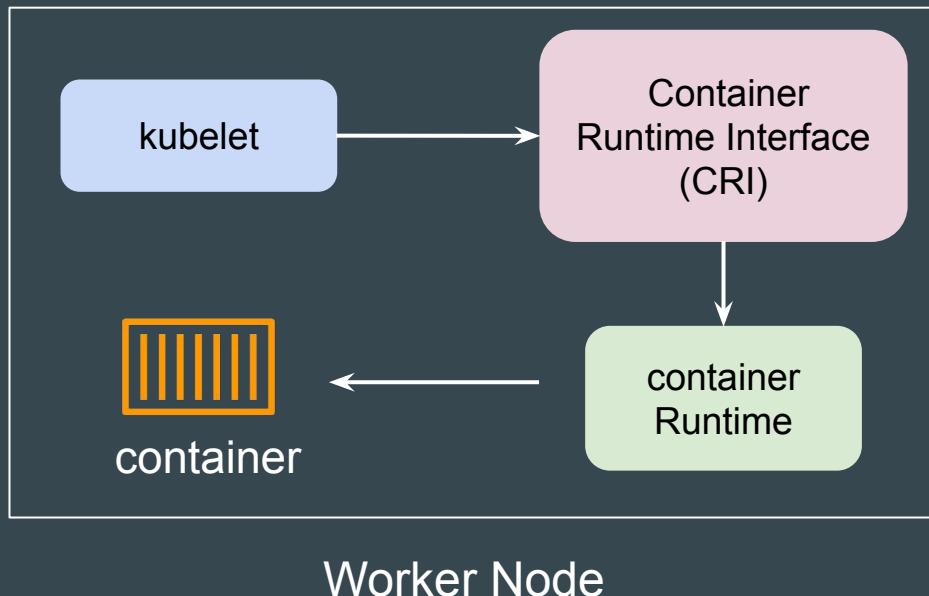
As the container ecosystem evolved, **new runtimes emerged** (like containerd, CRI-O, etc.)

To support them, Kubernetes introduced the CRI in Kubernetes

Popular CRI Compatible Container Runtime
containerd
CRI-O
Docker Engine (using cri-dockerd)
Mirantis Container Runtime

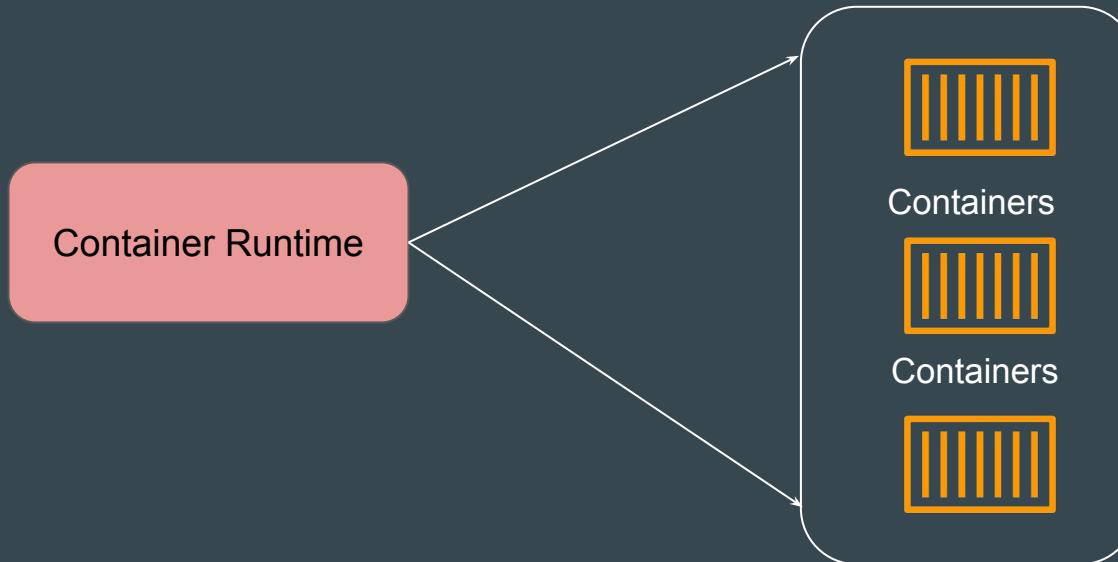
# Introducing CRI

The **Container Runtime Interface** (CRI) is essentially a plugin interface or an API that allows the kubelet to communicate with various container runtimes.



# Container Runtime

A **container runtime** is the foundational software that allows containers to operate within a host system, responsible for tasks like pulling container images, managing their lifecycle, and running them on the system



# cricl

cricl is a tool specifically designed to interact directly with container runtimes that implement the Kubernetes Container Runtime Interface (CRI) specification.

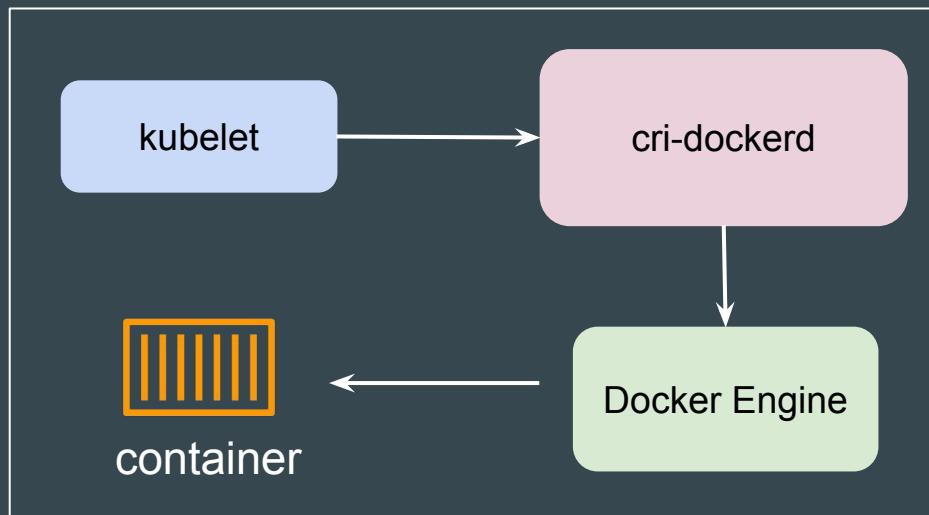
POD ID	CREATED	STATE	NAME	NAMESPACE	ATTEMPT
30c324b414b96	2 days ago	Ready	pod-2	default	0
9856db98fb73e	2 days ago	Ready	pod-1	default	0
ac33596901fc7	10 days ago	Ready	local-path-provisioner-84967477f-xfhpb	local-path-storage	0
2cd9a13b011f0	13 days ago	Ready	calico-apiserver-68c899dd7f-xbjrc	calico-apiserver	15
e6fab95c047e3	13 days ago	Ready	coredns-7c65d6cfc9-q2wct	kube-system	16
e7ff0c5f58758	13 days ago	Ready	coredns-7c65d6cfc9-8g2jj	kube-system	16
4dc3a5c2b97d2	13 days ago	Ready	calico-apiserver-68c899dd7f-prpnw	calico-apiserver	15
1e2afed0e2af3	13 days ago	Ready	calico-kube-controllers-6dd68f89b5-k2124	calico-system	16
cec224290e3bc	13 days ago	Ready	csi-node-driver-c62m2	calico-system	16
0aab7306f878e	13 days ago	Ready	calico-node-jsbzq	calico-system	0
9267d7acd41b4	13 days ago	Ready	calico-typha-5dc67b94dc-rw9jz	calico-system	0
11de21c9da886	13 days ago	Ready	tigera-operator-76c4976dd7-gntht	tigera-operator	0
2e57fff83a872	13 days ago	Ready	kube-proxy-wrpjh	kube-system	0
c6f6536c0d520	13 days ago	Ready	etcd-kubeadm	kube-system	0
173f9728442cd	13 days ago	Ready	kube-controller-manager-kubeadm	kube-system	0
cb603a9ed7a8e	13 days ago	Ready	kube-scheduler-kubeadm	kube-system	0
a804208c327b7	13 days ago	Ready	kube-apiserver-kubeadm	kube-system	0

# **Configuring cri-dockerd**

# Setting the Base

`cri-dockerd` is a shim (adapter) that implements the Kubernetes CRI (Container Runtime Interface) for Docker.

Translates CRI gRPC calls from Kubernetes into Docker API calls.



# Point to Note

The cri-dockerd provides its own socket which kubelet connects to.

/var/run/cri-dockerd.sock

---

# Kubernetes Events

Let's Validate!

---

# Overview of k8s Events

Kubernetes Events are created when other resources have state changes, errors, or other messages that should be broadcast to the system.

It provides insight into what is happening inside a cluster, such as what decisions were made by scheduler or why some pods were evicted from the node.

Events:				
Type	Reason	Age	From	Message
Normal	Scheduled	9m33s	default-scheduler	Successfully assigned default/nginx-7bb7cd8db5-nzpzm to secondary-nodes-btws
Normal	Pulling	9m30s	kubelet, secondary-nodes-btws	Pulling image "nginx"
Normal	Pulled	9m26s	kubelet, secondary-nodes-btws	Successfully pulled image "nginx"
Normal	Created	9m26s	kubelet, secondary-nodes-btws	Created container nginx
Normal	Started	9m26s	kubelet, secondary-nodes-btws	Started container nginx

# Important Pointer

All the events are stored in the master server.

To avoid filling up master's disk, a retention policy is enforced: events are removed one hour after the last occurrence.

To provide longer history and aggregation capabilities, a third party solution should be installed to capture events.

# Events and Namespaces

Events are namespaced.

Hence if you want event of a pod in “kplabs-namespace” then you will have to explicitly specify the --namespace kplabs-namespace.

To see events from all namespaces, you can use the --all-namespaces argument.

---

# Field Selector

Let's Search!

# Overview of Field Selector

Field selectors let you select Kubernetes resources based on the value of one or more resource fields

C:\Users\Zeal Vora>kubectl get pods --all-namespaces --field-selector metadata.namespace!=default					
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	cilium-jg5vs	2/2	Running	0	3d
kube-system	cilium-mdm7m	2/2	Running	0	3d
kube-system	cilium-nj598	2/2	Running	0	11d
kube-system	cilium-operator-69676856c9-nvfjr	1/1	Running	2 (7d15h ago)	11d
kube-system	coredns-566f6cc75f-b4vmx	1/1	Running	0	11d
kube-system	coredns-566f6cc75f-cxsxq	1/1	Running	0	11d
kube-system	cpc-bridge-proxy-bc762	1/1	Running	0	3d
kube-system	cpc-bridge-proxy-svs6d	1/1	Running	0	3d
kube-system	cpc-bridge-proxy-vbzzc	1/1	Running	0	11d
kube-system	csi-do-node-vc7r4	2/2	Running	0	11d

# Default Configuration

By default, no selectors/filters are applied, meaning that all resources of the specified type are selected.

This makes the kubectl queries kubectl get pods and kubectl get pods --field-selector "" equivalent.

---

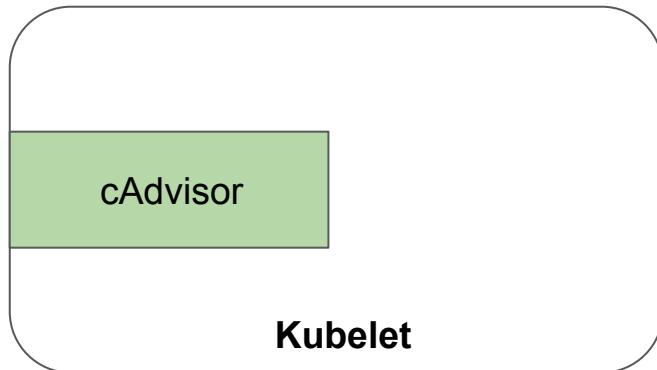
# Monitoring Cluster Components

Let's Monitor!

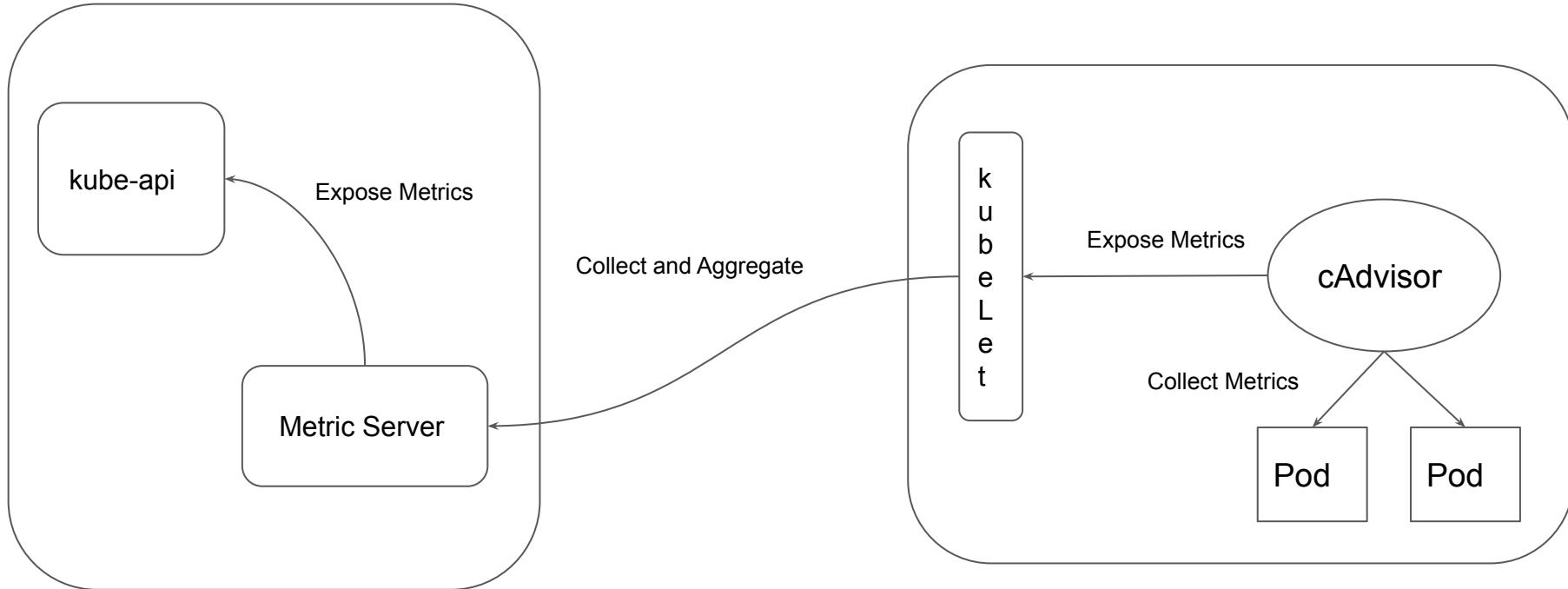
# Overview about cAdvisor

One of the important functionalities of a Kubelet is to retrieve metrics aggregate and expose them through the Kubelet Summary API.

This is achieved with cAdvisor.



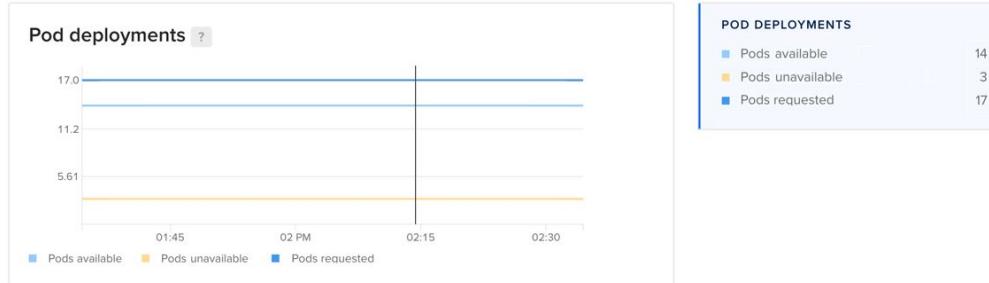
# Overall Workflow



# Overview about kube-state-metrics

kube-state-metrics is a simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.

It is not focused on the health of the individual Kubernetes components, but rather on the health of the various objects inside, such as deployments, nodes and pods.



---

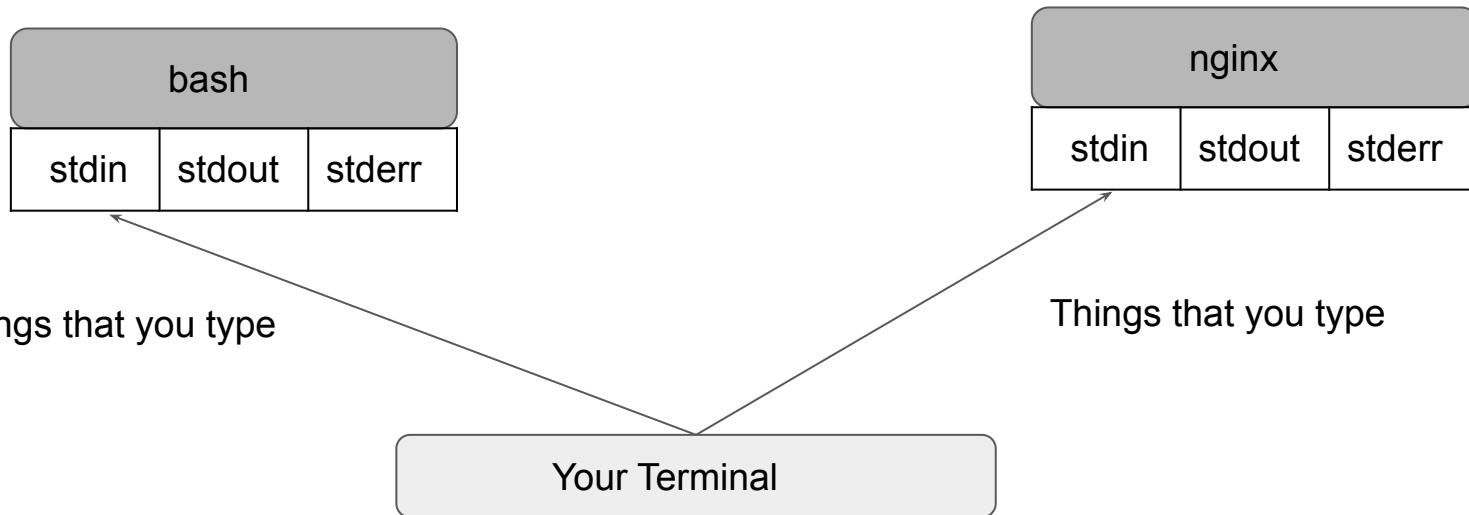
# Logging Drivers

Build once, use anywhere

---

# Getting Started

UNIX and Linux commands typically open three I/O streams when they run, called **STDIN**, **STDOUT**, and **STDERR**



# Logging Drivers in Docker

There are a lot of logging driver options available in Docker, some of these include:

- json-file
- none
- syslog
- local
- journald
- splunk
- awslogs

The docker logs command is not available for drivers other than json-file and journald.

---

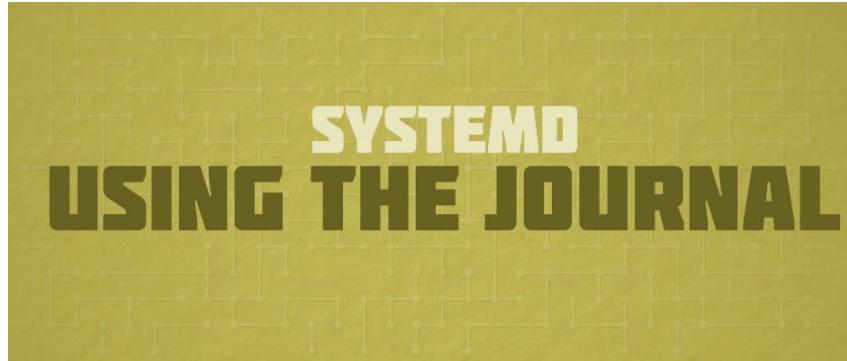
# Cluster Component Logs

Let's Troubleshoot!

# Component Logs

If we are making use of systemd, then we can view the component level logs with `journalctl`

It provides insight into what is happening inside a cluster, such as what decisions were made by scheduler or why some pods were evicted from the node.



---

# Troubleshooting Application Failure

Let's Troubleshoot!

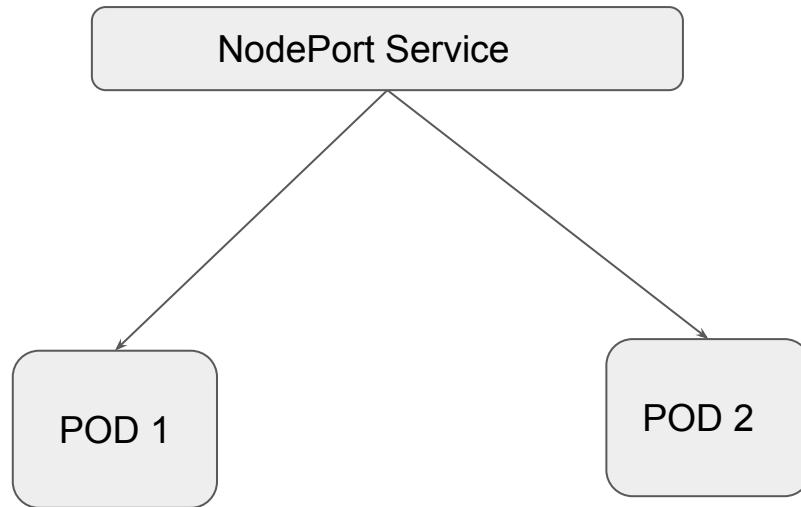
# Troubleshooting Application Failure

An application can make use of various resource objects like:

- Pods
- Services
- Secrets
- ConfigMaps
- Roles, RoleBindings
- Networking Aspect etc

If one of these intermediary things are not working, your application can behave unexpectedly.

# Scenario 1: Fix this up!



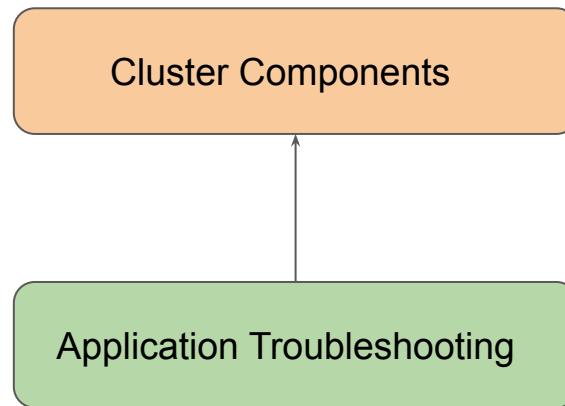
---

# Troubleshooting Control Plane Failure

Let's Troubleshoot!

# Keeping Record Straight

At this step of Cluster Troubleshooting, we assume you have already ruled out your application as the root cause of the problem you are experiencing.



# Step 1: Verify Cluster Components

Verify if all the cluster level components are healthy.

```
[root@kplabs-cka-master ~]# kubectl get componentstatus
NAME           STATUS   MESSAGE             ERROR
scheduler      Healthy  ok
controller-manager  Healthy  ok
etcd-0         Healthy  {"health":"true"}
```

## Step 2: Verify and Dump Cluster Info

Verify if the master and DNS servers are running.

To get more details, run the cluster-info dump to dump lots of relevant info for debugging and diagnosis

```
[root@kplabs-cka-master ~]# kubectl cluster-info  
Kubernetes master is running at https://134.209.158.242:6443  
KubeDNS is running at https://134.209.158.242:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

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

# Step 3: Check Logs of Individual Components

Journalctl allows us to look into the logs for components deployed via systemd.

```
journalctl -l kube-apiserver
```

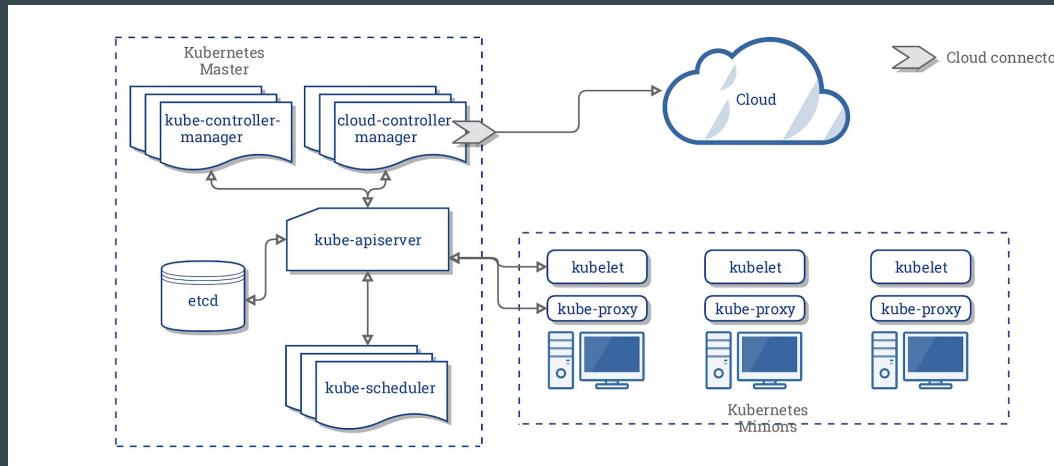
```
-- Logs begin at Sun 2019-09-15 15:24:37 UTC, end at Wed 2019-09-18 13:28:44 UTC. --
kplabs-cka-master systemd[1]: Started Kubernetes API Server.
kplabs-cka-master kube-apiserver[18477]: Flag --insecure-port has been deprecated, This flag will be removed in a
kplabs-cka-master kube-apiserver[18477]: I0915 16:59:34.757359 18477 server.go:560] external host was not speci
kplabs-cka-master kube-apiserver[18477]: I0915 16:59:34.758460 18477 server.go:147] Version: v1.15.0
kplabs-cka-master kube-apiserver[18477]: I0915 16:59:35.736575 18477 plugins.go:158] Loaded 10 mutating admissi
kplabs-cka-master kube-apiserver[18477]: I0915 16:59:35.736616 18477 plugins.go:161] Loaded 6 validating admiss
kplabs-cka-master kube-apiserver[18477]: E0915 16:59:35.739172 18477 prometheus.go:55] Failed to register depth
kplabs-cka-master kube-apiserver[18477]: E0915 16:59:35.739235 18477 prometheus.go:68] Failed to register adds
```

# **Version Skew Policy**

# Setting the Base

In Kubernetes, different components (such as the API server, kubelet, etc) interact with each other across various versions.

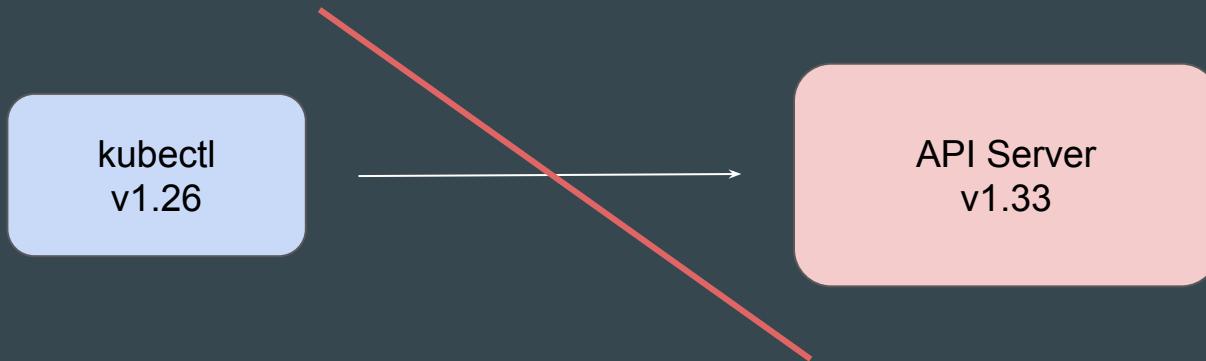
To ensure compatibility and stability, Kubernetes follows a **Version Skew Policy** that defines how versions of these components can differ while still maintaining a functional cluster.



# Understanding the Challenge

Version skew happens whenever two components of a software system communicate, but they aren't running at exactly the same version

While some level of skew is allowed, excessive differences between versions can lead to compatibility issues, failures, or unexpected behaviors.



# Kubernetes Versioning Basics

Kubernetes versions are expressed as **x.y.z**, where x is the major version, y is the minor version, and z is the patch version.

Format: <MAJOR>.<MINOR>.<PATCH> (e.g., v1.32.2)

Major Version	Minor Version	Patch Version
1	32	2

# kube-apiserver and kubelet

Component	Version Skew
kube-apiserver	<p>In highly-available (HA) clusters, the newest and oldest kube-apiserver instances must be within one minor version.</p> <p>If newest kube-apiserver is at 1.32 other kube-apiserver instances are supported at 1.32 and 1.31</p>
kubelet	<p>kubelet must not be newer than kube-apiserver.</p> <p>kubelet may be up to three minor versions older than kube-apiserver</p> <p>Example:</p> <p>kube-apiserver is at 1.32 kubelet is supported at 1.32, 1.31, 1.30, and 1.29</p>

# kube-proxy

Component	Version Skew
kube-proxy	<p>kube-proxy must not be newer than kube-apiserver.</p> <p>kube-proxy may be up to three minor versions older than kube-apiserver</p> <p>Example:</p> <p>kube-apiserver is at 1.32</p> <p>kube-proxy is supported at 1.32, 1.31, 1.30, and 1.29</p>

# Controller Manager, Scheduler, Cloud Controller Manager

Component	Version Skew
Controller Manager	<p>Must not be newer than the kube-apiserver instances they communicate with.</p> <p>They are expected to match the kube-apiserver minor version, but may be up to one minor version older (to allow live upgrades).</p>
Scheduler	
Cloud Controller Manager	<p><b>Example:</b></p> <p>kube-apiserver is at 1.32</p> <p>kube-controller-manager, kube-scheduler, and cloud-controller-manager are supported at 1.32 and 1.31</p>

# kubectl

Component	Version Skew
kubectl	<p>kubectl is supported within one minor version (older or newer) of kube-apiserver.</p> <p>Example:</p> <p>kube-apiserver is at 1.32</p> <p>kubectl is supported at 1.33, 1.32, and 1.31</p>

---

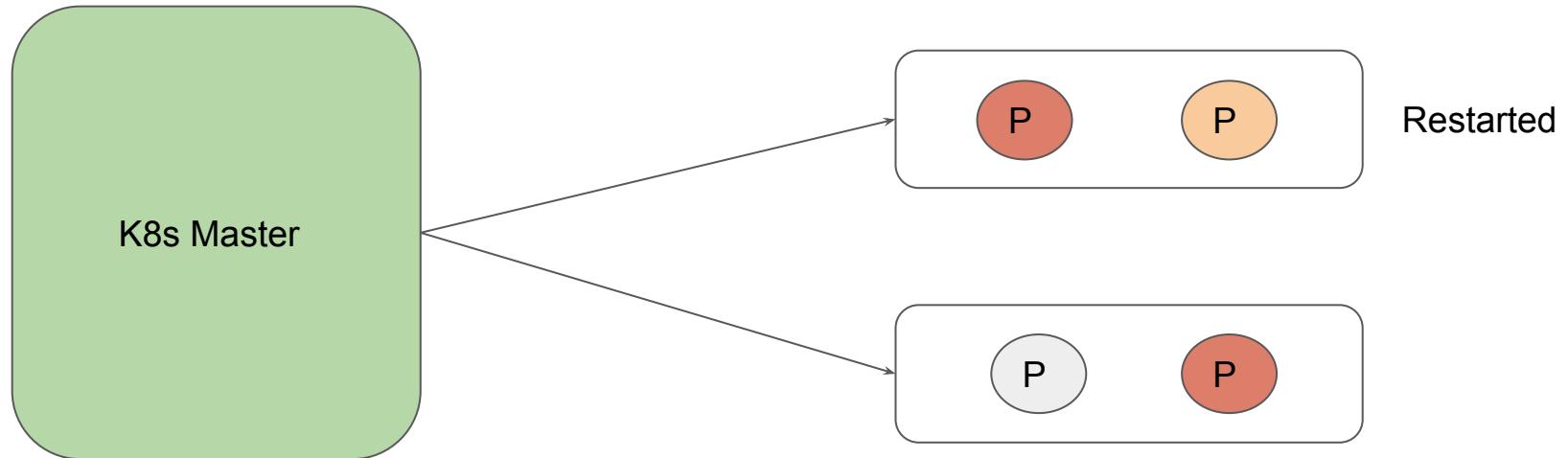
# Taint Based Evictions

Advanced Scheduling

---

## Scenerio 1.1- Immediate Restart

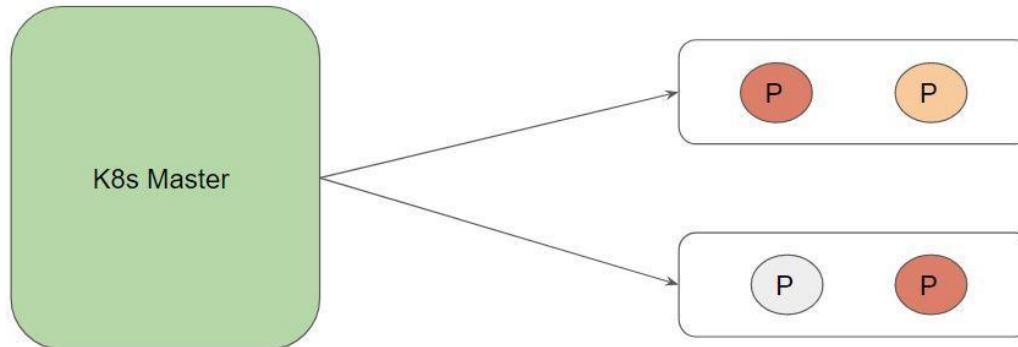
If node starts immediately, the kubelet process will start the pods and node is back online.



## Scenario 1.2 - Brief Downtime

If reboot takes longer (default time is 5 minutes) then the node controller will terminate the pods that are bound to the unavailable node.

If there is a corresponding replica set, then a new copy of the pod will be started on a different node.



# Understanding Taints

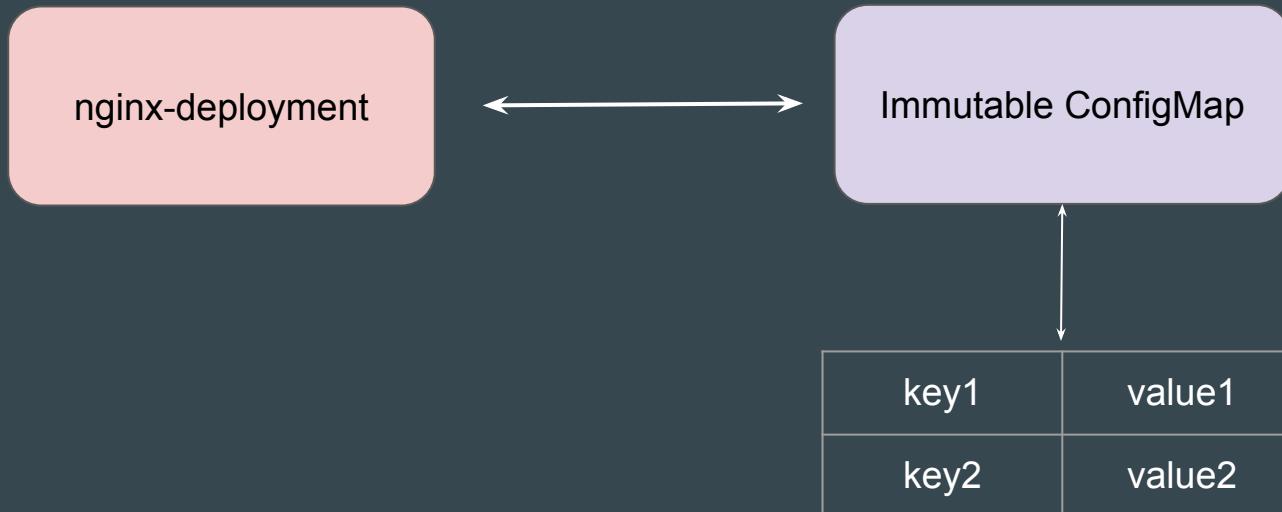
The worker nodes are automatically tainted in Kubernetes based on specific node conditions:

- node.kubernetes.io/not-ready
- node.kubernetes.io/unreachable
- node.kubernetes.io/out-of-disk
- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/network-unavailable
- node.kubernetes.io/unschedulable

# **Exercise - Nginx and Immutable ConfigMap**

# Understanding the Requirement

You need to change the nginx configuration stored as part of an immutable ConfigMap to fulfill the use case requirement.

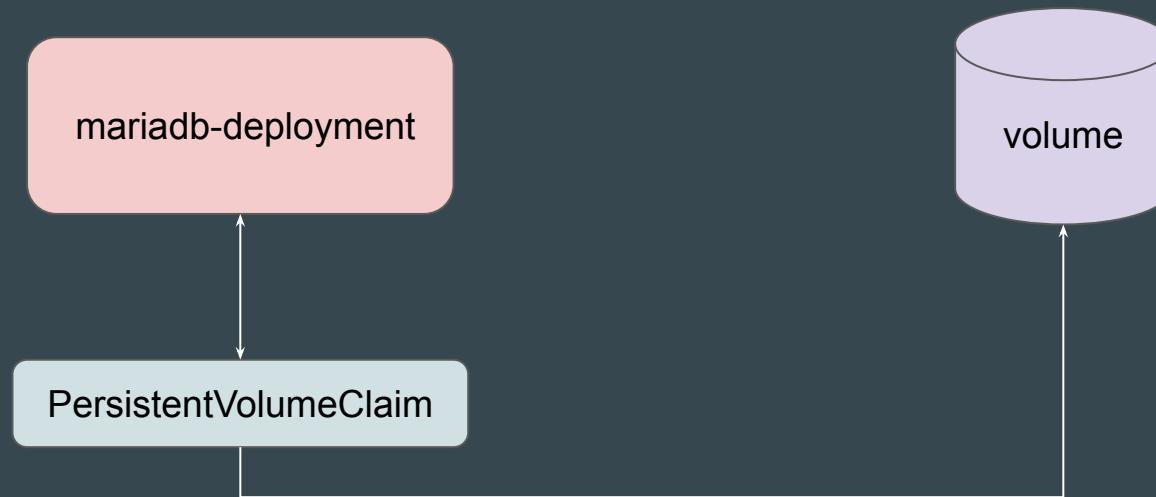


# **Exercise - Persistent Volume and Deployment**

# Understanding the Requirement

Mariadb-deployment associated with a persistent volume was accidentally deleted along with the PVC.

Recreate the deployment and PVC to use existing volume to restore the data.



# **Part 1 - Important Pointers for CKA Exams**

# Priority Class

Priority indicates the importance of a Pod relative to other Pods.

If a Pod cannot be scheduled, the scheduler tries to preempt (evict) lower priority Pods to make scheduling of the pending Pod possible.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: high-priority-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: high-priority-app
  template:
    metadata:
      labels:
        app: high-priority-app
    spec:
      priorityClassName: high-priority
      containers:
        - name: nginx
          image: nginx:latest
```

## **Commands to Know**

```
kubectl create priorityclass high-priority --value=1000
```

```
kubectl create priorityclass low-priority --value=100
```

# Horizontal Pod Autoscaler

Be very familiar with HPA and Stabilization Window

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
      target:
        type: Utilization
        averageUtilization: 50
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 60
```

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
  scaleUp:
    stabilizationWindowSeconds: 0
```

## **Commands to Know**

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=3
```

# Helm

You should be comfortable with basic commands related to Helm.

Helm Command	Description
helm search <repo/hub> <keyword>	Provides the ability to search for Helm charts in the various places they can be stored including the Artifact Hub and repositories you have added.
helm repo add <name> <url>	Adds a Helm chart repository (e.g. nginx)
helm install <release> <chart>	Installs a chart as a release into Kubernetes
helm uninstall <release>	Uninstalls (deletes) a Helm release
helm list	Lists all installed Helm releases
helm template <release> <chart>	Renders Kubernetes manifests locally without installing

# Skipping CRDs

Depending on the chart, there can be a different way to skip crds. One common way is through `--skip-crds` (might not always work). `--set crds.install=false`

```
{-{ if .Values.crds.install }}  
apiVersion: apiextensions.k8s.io/v1  
kind: CustomResourceDefinition  
metadata:  
  annotations:  
    {{- if .Values.crds.keep }}  
    "helm.sh/resource-policy": keep  
    {{- end }}  
    {{- with .Values.crds.annotations }}  
      {{- toYaml . | nindent 4 }}  
    {{- end }}  
  labels:  
    app.kubernetes.io/name: applications.argoproj.io  
    app.kubernetes.io/part-of: argocd  
    {{- with .Values.crds.additionalLabels }}  
      {{- toYaml . | nindent 4 }}  
    {{- end }}  
  name: applications.argoproj.io
```

```
root@kubeadm:~# helm template my-argo-cd argo/argo-cd --set crds.install=false --version 7.8.23 > argo-no-crd.yaml  
root@kubeadm:~# cat argo-no-crd.yaml | grep CustomResourceDefinition  
root@kubeadm:~# |
```

# Ingress

You should be familiar with Ingress as well as Ingress with TLS.

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

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
    - hosts:
        - https-example.foo.com
      secretName: tls-secret
  rules:
    - host: https-example.foo.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: service1
                port:
                  number: 80
```

# Gateway API

You should be comfortable with basic of Gateway API

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-tls-gateway
  namespace: default
spec:
  gatewayClassName: nginx
  listeners:
  - name: https
    protocol: HTTPS
    port: 443
    tls:
      mode: Terminate
      certificateRefs:
      - kind: Secret
        name: example-tls
```

# Ingress to Gateway API Migration

Be comfortable with migrating Ingress to Gateway API (TLS, Multiple Hosts etc)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - example.com
    secretName: my-tls-secret
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
```



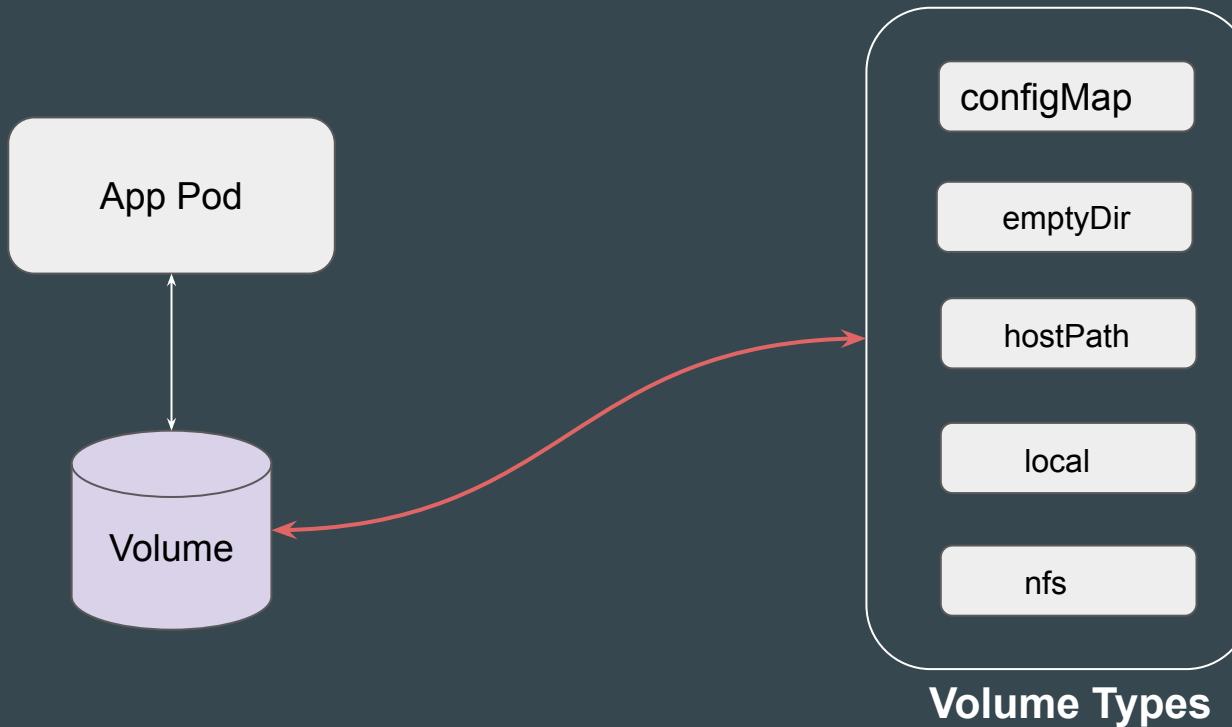
```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: nginx
  listeners:
  - name: https
    protocol: HTTPS
    port: 443
    hostname: example.com
    tls:
      mode: Terminate
      certificateRefs:
      - name: my-tls-secret
```



```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: apache-route
  namespace: default
spec:
  parentRefs:
  - name: nginx-gateway
  rules:
  - matches:
    - path:
      type: PathPrefix
      value: /
    backendRefs:
    - name: apache-service
      port: 80
```

# Kubernetes Volumes

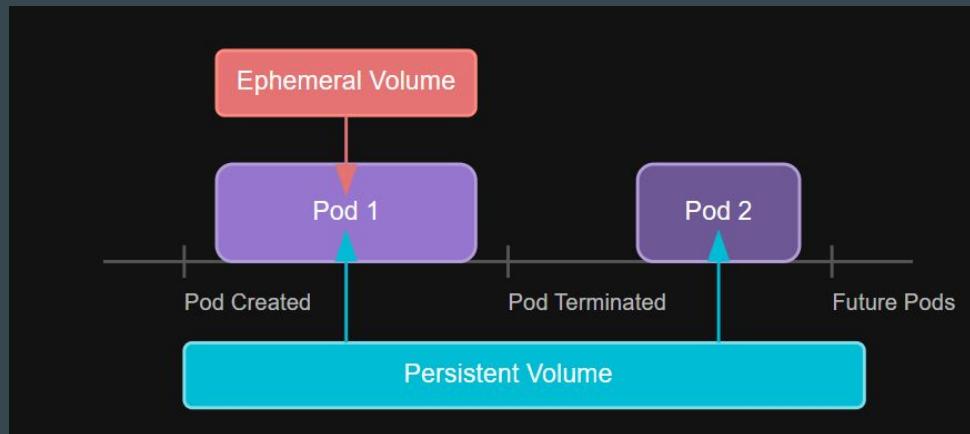
Kubernetes offers many volume types depending on the use-case.



# Ephemeral and Persistent Volumes

Ephemeral volume types have a lifetime linked to a specific Pod, BUT persistent volumes exist beyond the lifetime of any individual pod.

When a pod ceases to exist, Kubernetes destroys ephemeral volumes; however, Kubernetes does not destroy persistent volumes.



# Volume and Volume Mounts

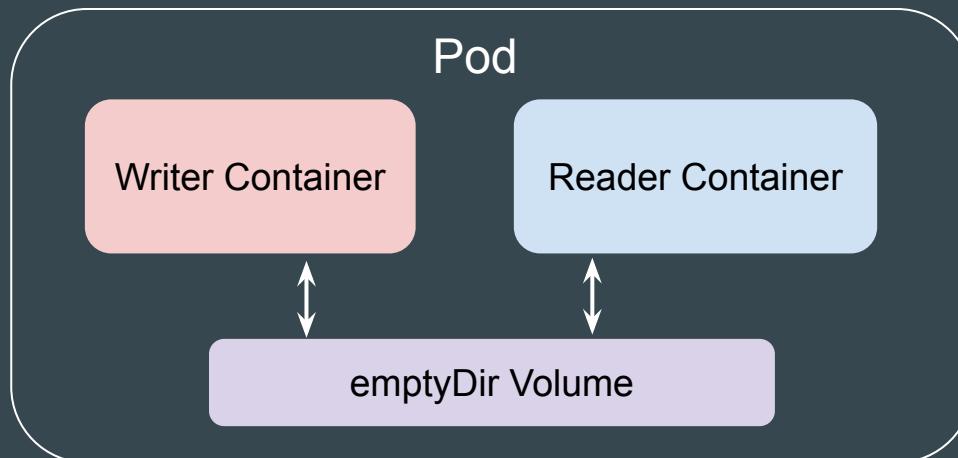
To use a volume, specify the volumes to provide for the Pod in `.spec.volumes` and declare where to mount those volumes into containers in `.spec.containers[*].volumeMounts`.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: nginx:latest
      name: test-container
      volumeMounts:
        - mountPath: /my-data
          name: data-volume
  volumes:
    - name: data-volume
      emptyDir:
        sizeLimit: 500Mi
```

# emptyDir volume

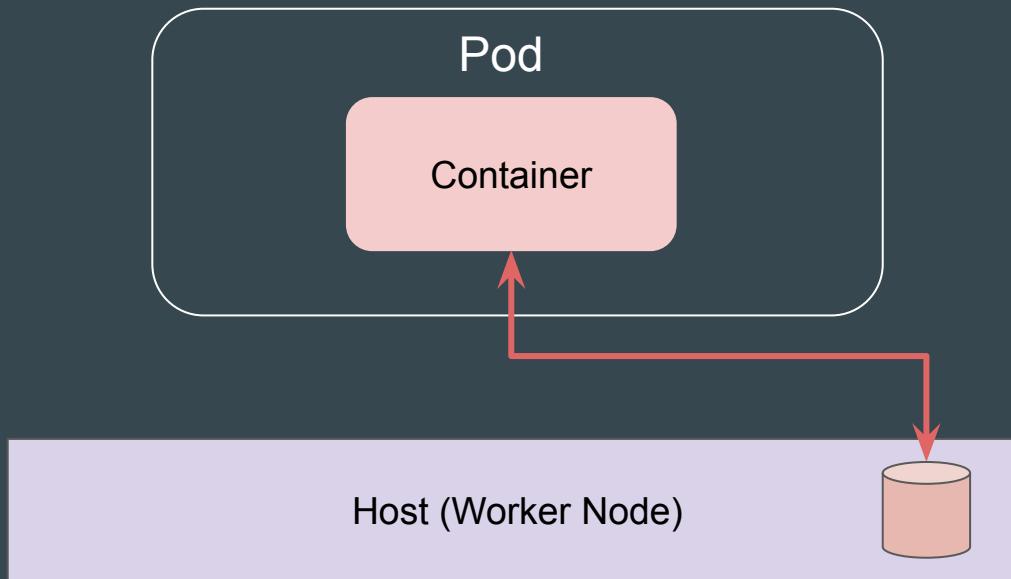
An emptyDir volume is a **temporary storage directory**.

All containers in the Pod can read and write the same files in the emptyDir volume.



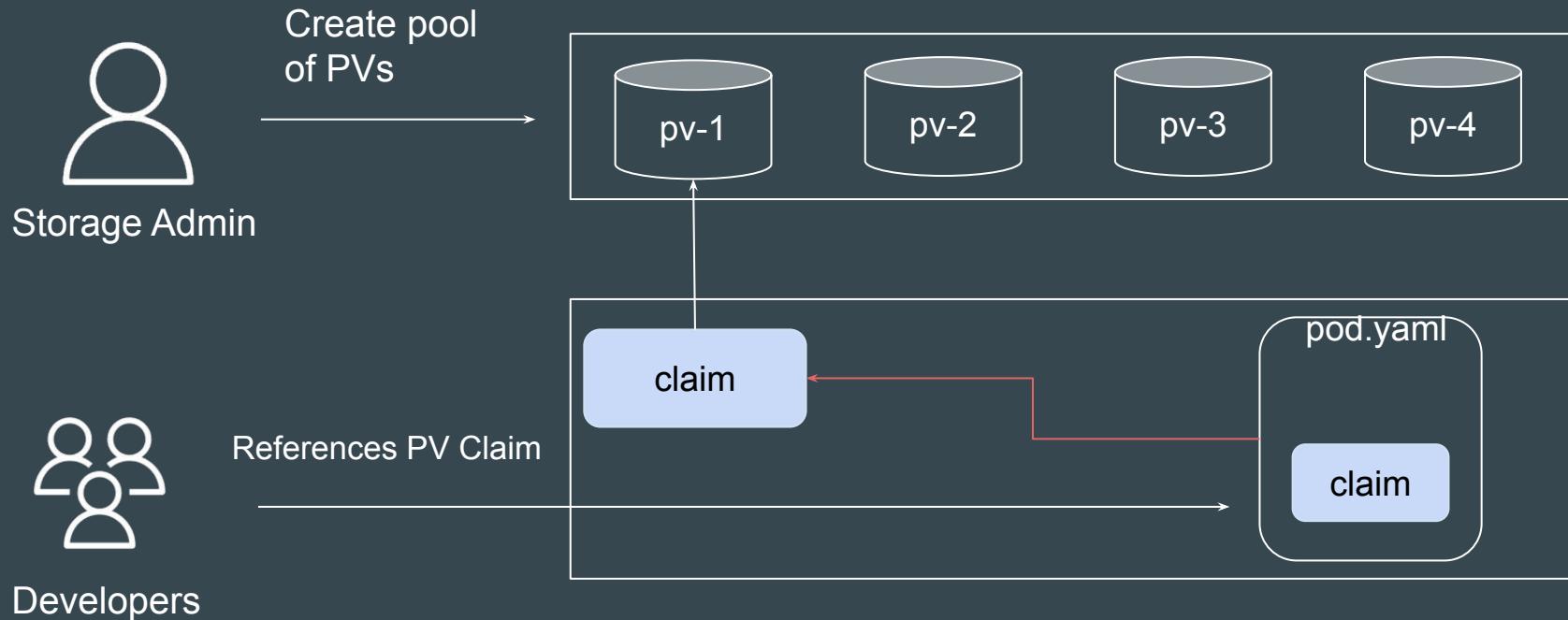
# hostPath Volume

A hostPath volume mounts a file or directory from the host node's filesystem into your Pod.



# PV and PVC

Be comfortable with the workflow of PV and PVC.



# Storage Class and Dynamic Provisioners

Be very familiar with Storage Class and Dynamic Provisioners.

Be comfortable with Default Class, Provisioner and volumeBindingMode.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: low-latency
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-driver.example-vendor.example
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

# Immutable ConfigMaps

Be familiar with workflow of Immutable ConfigMaps + Deployments.

What happens if you want to change value of Immutable ConfigMap. The entire deployment update workflow so new values reflect in Pods.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
immutable: true
data:
  setting1: "value1"
  setting2: "value2"
```

deployment

Commands to Know	Description
kubectl rollout restart deployment/nginx	Restart a deployment

# Service

You should know on how you can create Service and associate it with existing set of Pods and Deployments.

Service Type	Key Features	Use-Cases
ClusterIP	Default Service type. Accessible only within the cluster.	Internal microservices communication
NodePort	Exposes service on a static port (30000-32767) on each Node.	Development testing, demo applications

Commands to Know	Description
kubectl create deployment my-dep --image=busybox --replicas=3 --port=80	Create a deployment named my-dep that runs the busybox image and expose port 5701
kubectl create service nodeport my-ns --tcp=5678:8080	Create NodePort Service
kubectl expose deployment hello-world --type=NodePort --name=example-service	Create NodePort service that exposes the deployment.

# Requests and Limits

Requests and Limits are two ways in which we can control the amount of resource that can be assigned to a pod (resource like CPU and Memory)

```
! request-limit.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kplabs-pod
spec:
  containers:
  - name: kplabs-container
    image: nginx
    resources:
      requests:
        memory: "128Mi"
        cpu: "0.5"
      limits:
        memory: "500Mi"
        cpu: "1"
```

# Capacity, Allocatable, Allocated

Based on capacity of worker node, you should know how to calculate appropriate request/limits for deployments to schedule

```
Capacity:  
  cpu:                1  
  ephemeral-storage: 51432064Ki  
  hugepages-2Mi:     0  
  memory:             2014392Ki  
  pods:               110  
Allocatable:  
  cpu:                900m  
  ephemeral-storage: 47399790104  
  hugepages-2Mi:     0  
  memory:              1574Mi  
  pods:               110
```

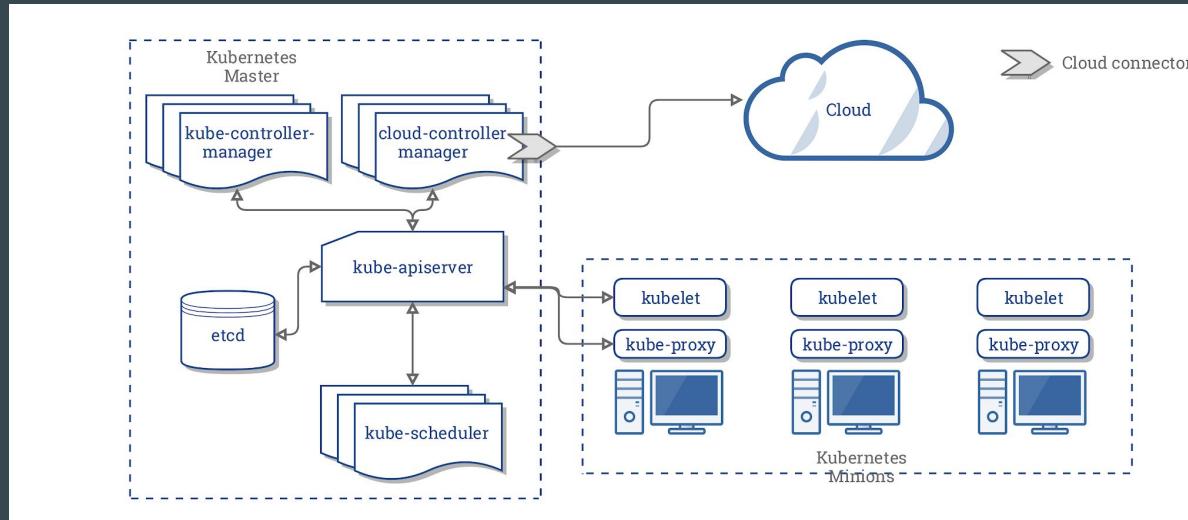
```
Allocated resources:  
(Total limits may be over 100 percent, i.e., overcommitted.)  
Resource      Requests     Limits  
-----  
cpu           702m (78%)   0 (0%)  
memory        720Mi (45%)  640Mi (40%)  
ephemeral-storage 0 (0%)    0 (0%)  
hugepages-2Mi 0 (0%)    0 (0%)  
Events:       <none>
```

## **Part 2 - Important Pointers for CKA Exams**

# Troubleshooting

The exam can present you with the broker Kubernetes cluster scenarios and you will have to fix the cluster.

Practice Kubernetes Cluster from Scratch videos



# Important Cluster Troubleshooting Pointers

<code>systemctl status &lt;kubelet / kube-apiserver, etcd&gt;</code>	Check the status of individual components of Kubernetes.
<code>journalctl -u kubelet, kube-apiserver, etcd</code>	Check logs to understand the issue
<code>/etc/kubernetes/manifests</code>	Path to static K8s component pods for kubeadm based environments.
<code>cd /var/log/containers</code>	Log directory for all important pods

# cricl

cricl is a tool specifically designed to interact directly with container runtimes that implement the Kubernetes Container Runtime Interface (CRI) specification.

POD ID	CREATED	STATE	NAME	NAMESPACE	ATTEMPT
30c324b414b96	2 days ago	Ready	pod-2	default	0
9856db98fb73e	2 days ago	Ready	pod-1	default	0
ac33596901fc7	10 days ago	Ready	local-path-provisioner-84967477f-xfhpb	local-path-storage	0
2cd9a13b011f0	13 days ago	Ready	calico-apiserver-68c899dd7f-xbjrc	calico-apiserver	15
e6fab95c047e3	13 days ago	Ready	coredns-7c65d6cfc9-q2wct	kube-system	16
e7ff0c5f58758	13 days ago	Ready	coredns-7c65d6cfc9-8g2jj	kube-system	16
4dc3a5c2b97d2	13 days ago	Ready	calico-apiserver-68c899dd7f-prpnw	calico-apiserver	15
1e2afed0e2af3	13 days ago	Ready	calico-kube-controllers-6dd68f89b5-k2124	calico-system	16
cec224290e3bc	13 days ago	Ready	csi-node-driver-c62m2	calico-system	16
0aab7306f878e	13 days ago	Ready	calico-node-jsbzq	calico-system	0
9267d7acd41b4	13 days ago	Ready	calico-typha-5dc67b94dc-rw9jz	calico-system	0
11de21c9da886	13 days ago	Ready	tigera-operator-76c4976dd7-gntht	tigera-operator	0
2e57fff83a872	13 days ago	Ready	kube-proxy-wrpjh	kube-system	0
c6f6536c0d520	13 days ago	Ready	etcd-kubeadm	kube-system	0
173f9728442cd	13 days ago	Ready	kube-controller-manager-kubeadm	kube-system	0
cb603a9ed7a8e	13 days ago	Ready	kube-scheduler-kubeadm	kube-system	0
a804208c327b7	13 days ago	Ready	kube-apiserver-kubeadm	kube-system	0

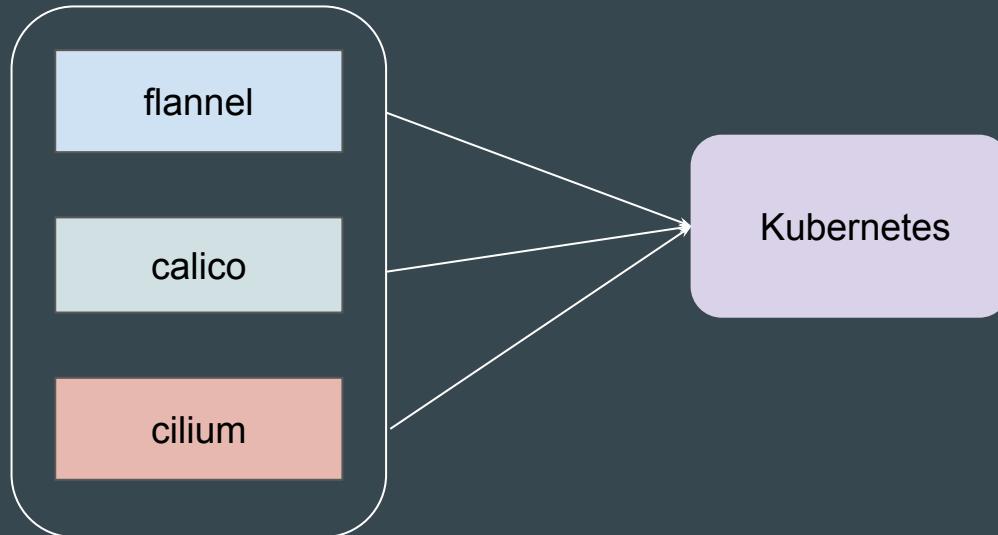
# Debugging Kubernetes nodes with crictl

Commands	Description
crictl pods	List All Pods
crictl ps -a	List All Containers
crictl logs 87d3992f84f74	Get all container logs:
crictl exec -i -t 1f73f2d81bf98 ls	Execute a command in a running container

# Network Policies

Be comfortable in **reading, analyzing, and writing** Network Policies.

CNI Plugins like Calico, Cilium supports Network Policy while Flannel does not.



# Sample Network Policy for Reference

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            env: security
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            env: security
  egress:
    - to:
      - ipBlock:
          cidr: 8.8.8.8/32
```

# Configuring Networking for cluster

You should know on how to configure Networking

Example:

1. Installing Tigera Operator with Calico.
2. You should be able to change the CIDR based on kubeadm cidr

```
# This section includes base Calico installation configuration.
# For more information, see: https://docs.tigera.io/calico/latest/reference/
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  name: default
spec:
  # Configures Calico networking.
  calicoNetwork:
    ipPools:
      - name: default-ipv4-ippool
        blockSize: 26
        cidr: 192.168.0.0/16
        encapsulation: VXLANCrossSubnet
        natOutgoing: Enabled
        nodeSelector: all()
```

# Configuring CNI Plugins

You should know on how to install Networking

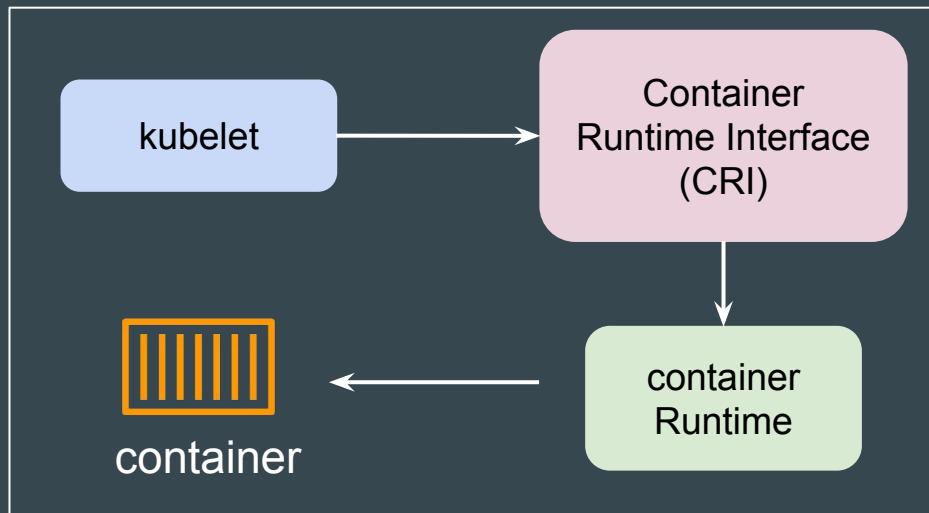
Example:

1. Installing Tigera Operator with Calico.
2. You should be able to change the CIDR based on kubeadm cidr

```
# This section includes base Calico installation configuration.
# For more information, see: https://docs.tigera.io/calico/latest/reference/
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  name: default
spec:
  # Configures Calico networking.
  calicoNetwork:
    ipPools:
      - name: default-ipv4-ippool
        blockSize: 26
        cidr: 192.168.0.0/16
        encapsulation: VXLANCrossSubnet
        natOutgoing: Enabled
        nodeSelector: all()
```

# Container Runtime Interface

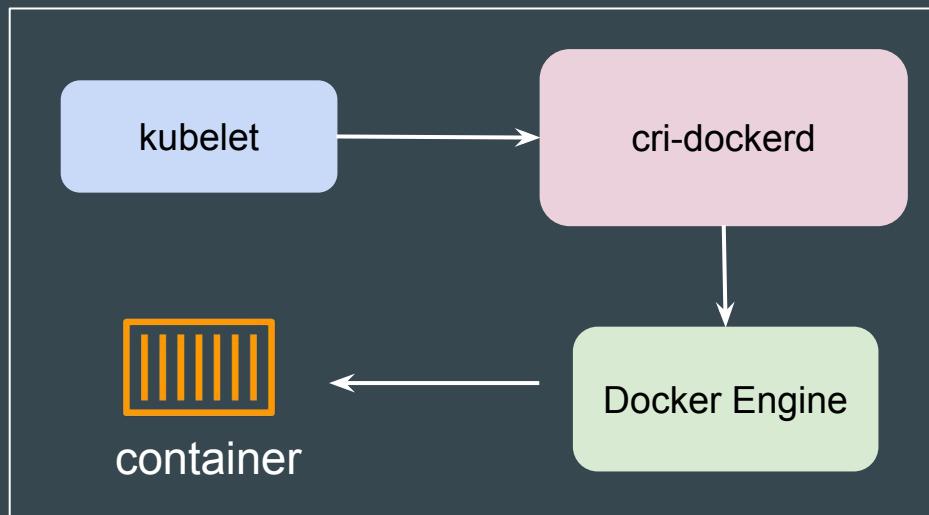
The Container Runtime Interface (CRI) is essentially a plugin interface or an API that allows the kubelet to communicate with various container runtimes.



# Configuring cri-dockerd

cri-dockerd is a shim (adapter) that implements the Kubernetes CRI (Container Runtime Interface) for Docker.

Be aware of installing cri-dockerd through dpkg.



# sysctl - Kernel Parameters Permanent Modification

To ensure your kernel parameter changes survive a reboot, **you need to store them in configuration files** that the system reads during boot.

Configuration Files	Description
/etc/sysctl.conf	The traditional main configuration file. You can add your parameter settings here.
/etc/sysctl.d/*.conf	The modern, preferred approach  You should place your custom settings in a file here, typically named something like 99-custom.conf

# sysctl - Applying the Changes

After editing /etc/sysctl.conf or files in /etc/sysctl.d/, the changes won't take effect until the next boot unless you apply them manually.

You can run the `sysctl --system` to load the parameters from these configuration files again.

```
root@kubeadm:~# sysctl --system
* Applying /usr/lib/sysctl.d/10-apparmor.conf ...
* Applying /etc/sysctl.d/10-bufferbloat.conf ...
* Applying /etc/sysctl.d/10-console-messages.conf ...
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
* Applying /etc/sysctl.d/10-magic-sysrq.conf ...
* Applying /etc/sysctl.d/10-map-count.conf ...
* Applying /etc/sysctl.d/10-network-security.conf ...
* Applying /etc/sysctl.d/10-ptrace.conf ...
* Applying /etc/sysctl.d/10-zero-page.conf ...
```

# Editing Resources

**kubectl edit** command opens the resource manifest in your default editor (like vim), allowing you to manually edit fields in the YAML definition.

**kubectl patch** allows you to update an object non-interactively.

```
root@kubeadm:~# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
test-pod  1/1     Running   0          19m   run=test-pod
root@kubeadm:~# kubectl patch pod test-pod -p '{"metadata":{"labels":{"run":"dev-pod"}}}'
pod/test-pod patched
root@kubeadm:~# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
test-pod  1/1     Running   0          19m   run=dev-pod
```

# Point to Note - Editing Immutable Fields

Not all the field of a object can be edited.

Possible Workflow in this case:

1. kubectl get deployment test-deployment -o yaml > deployment.yaml
2. kubectl delete deployment test-deployment
3. Modify the deployment.yaml
4. kubectl create -f deployment.yaml

# Multi-Container Pod Configuration

To create a multi-container based Pods, you can defined additional details in container definition.

Exam might ask you to add sidecar container for shared volume access.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx
```



```
! multi-container-pods.yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
    - name: redis-container
      image: redis
```

# JSONPath

JSONPath is a query language for JSON, similar to XPath for XML.

```
root@kubeadm:~# kubectl get pods -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.podIP}{ "\n"}{end}'  
pod-1  192.168.45.255  
pod-2  192.168.45.199
```

# kubectl explain

You can also use **kubectl explain** to describe the fields associated with each supported API resource. Keep a note of **--recursive** flag

```
C:\>kubectl explain pods.metadata
KIND:     Pod
VERSION:  v1

FIELD: metadata <ObjectMeta>

DESCRIPTION:
  Standard object's metadata. More info:
  https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
  ObjectMeta is metadata that all persisted resources must have, which
  includes all objects users must create.

FIELDS:
  annotations  <map[string]string>
    Annotations is an unstructured key value map stored with a resource that may
    be set by external tools to store and retrieve arbitrary metadata. They are
    not queryable and should be preserved when modifying objects. More info:
    https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations.html

  creationTimestamp      <string>
    CreationTimestamp is a timestamp representing the server time when this
    object was created. It is not guaranteed to be set in happens-before order
    across separate operations. Clients may not set this value. It is
    represented in RFC3339 form and is in UTC.

    Populated by the system. Read-only. Null for lists. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#when-created

  deletionGracePeriodSeconds   <integer>
```

```
C:\Users\zealv>kubectl explain pod.spec --recursive
KIND:     Pod
VERSION:  v1

FIELD: spec <PodSpec>

DESCRIPTION:
  Specification of the desired behavior of the pod. More info:
  https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#pod
  PodSpec is a description of a pod.

FIELDS:
  activeDeadlineSeconds <integer>
  affinity            <Affinity>
  nodeAffinity        <NodeAffinity>
  preferredDuringSchedulingIgnoredDuringExecution  <[]PreferredSchedulingTerm>
    preference      <NodeSelectorTerm> -required-
    matchExpressions <[]NodeSelectorRequirement>
      key <string> -required-
      operator       <string> -required-
      enum: DoesNotExist, Exists, Gt, In, ...
      values         <[]string>
  matchFields        <[]NodeSelectorRequirement>
    key <string> -required-
    operator       <string> -required-
    enum: DoesNotExist, Exists, Gt, In, ...
    values         <[]string>
    weight <integer> -required-
  requiredDuringSchedulingIgnoredDuringExecution  <NodeSelector>
  nodeSelectorTerms  <[]NodeSelectorTerm> -required-
    matchExpressions <[]NodeSelectorRequirement>
```

# Custom Resource Definition

A **Custom Resource Definition** allows you to extend the Kubernetes API by defining your own custom resource types.

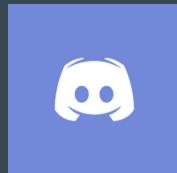
```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: databases.kplabs.internal
spec:
  group: kplabs.internal
  names:
    kind: Database
    listKind: DatabaseList
    plural: databases
    singular: database
  scope: Namespaced
  versions:
    - name: v1
      served: true
      storage: true
      schema:
```

# Deployment Commands

Commands	Description
kubectl create deployment my-dep --image=nginx --replicas=3 --port=5701	Create deployment with 3 replicas and expose port 5701
kubectl set image deployment/frontend www=image:v2	Rolling update "www" containers of "frontend" deployment, updating the image
kubectl rollout restart deployment/frontend	Rolling restart of the "frontend" deployment
kubectl scale --replicas=3 deployment/foo	Scale deployment to 3 replicas
kubectl expose deployment hello-world --type=NodePort --name=example-service	Create a NodePort Service that exposes the deployment:

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)