

# **Certified Kubernetes Application Developer**

# 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.



# PPT Version

Release Date = 4th Feb 2025

# What this Course is All About?

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

This is a perfect course for CKAD 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. Application Design and Build
2. Application Deployment
3. Application Observability and Maintenance
4. Application Environment, Configuration and Security
5. Services and Networking

# The Exciting Part

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

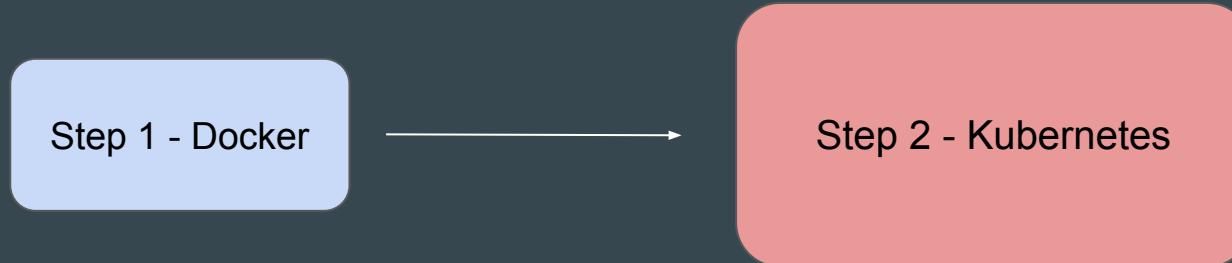
We also have practice tests available as part of the course.



# Point to Note

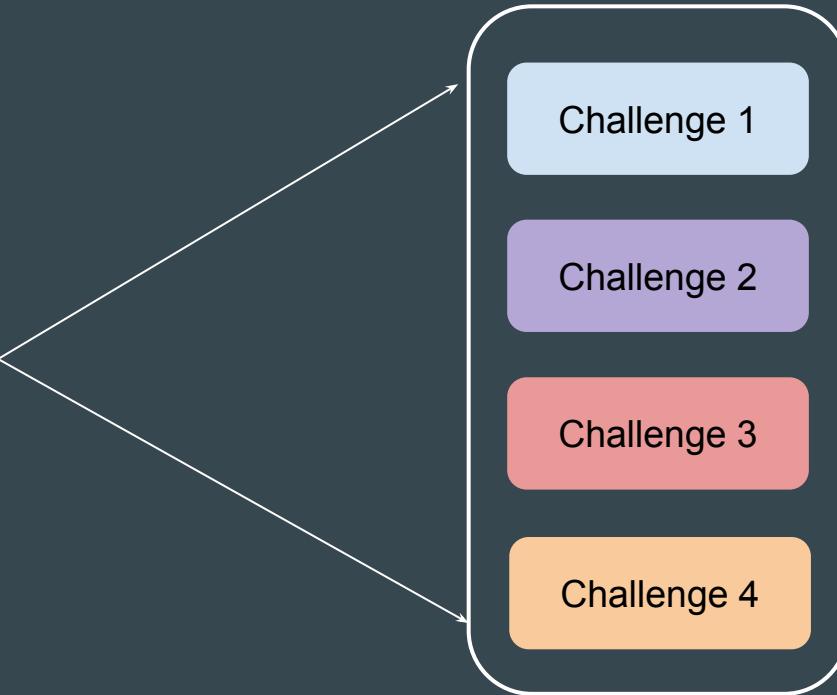
We start our journey on Kubernetes from absolute scratch.

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



# Lab Based Exams

CKAD 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.

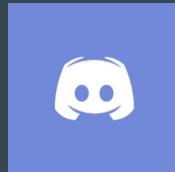
Something about me :-



- 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

# Join us in our Adventure

Be Awesome



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



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

# **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.

The slide displays a vertical stack of five numbered steps on the left side:

- 1 Certified Kubernetes Application Developer
- 2 Setting the Base
- 3 Kubernetes is one of the most popular container orchestration tools in the industry.
- 4 It is used extensively in many medium to large-scale organizations.
- 5 ING, JD.COM, NATE, Nav, NOKIA, NORDSTROM, The New York Times, PingCAP, OpenAI, Pear Deck, Fiverr, and Pinterest logos.

The right side of the slide is a large, solid gray rectangular area.

**Certified Kubernetes Application Developer**

**Setting the Base**

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

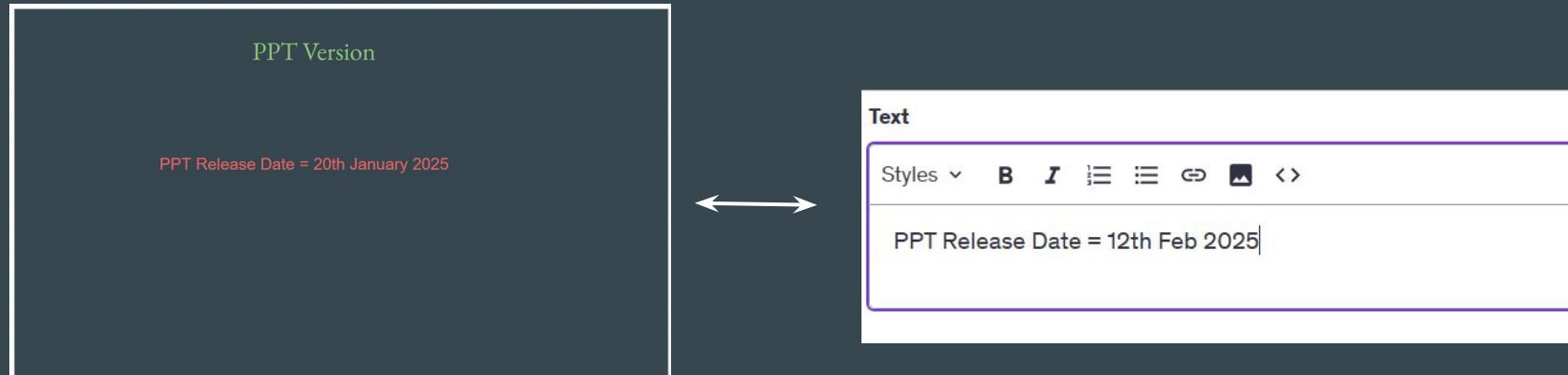
It is used extensively in many medium to large-scale organizations.

ING, JD.COM, NATE, Nav, NOKIA, NORDSTROM, The New York Times, PingCAP, OpenAI, Pear Deck, Fiverr, and Pinterest

## 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.

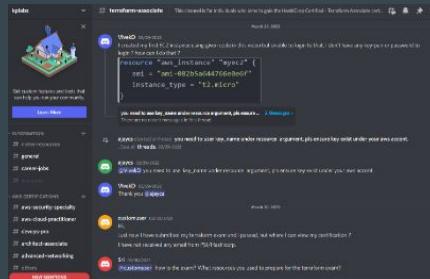


## 4 - Our Community (Optional)

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: #ckad



# 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.

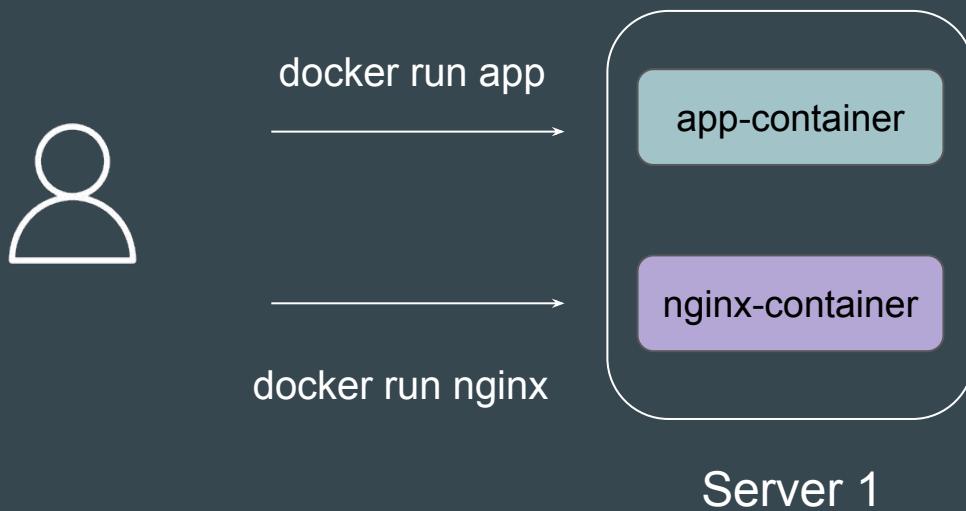


```
C: > kplabs-k8s > ! lb-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: simple-service
spec:
  type: LoadBalancer
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 80
```

# **Overview of Container Orchestration**

# Setting the Base

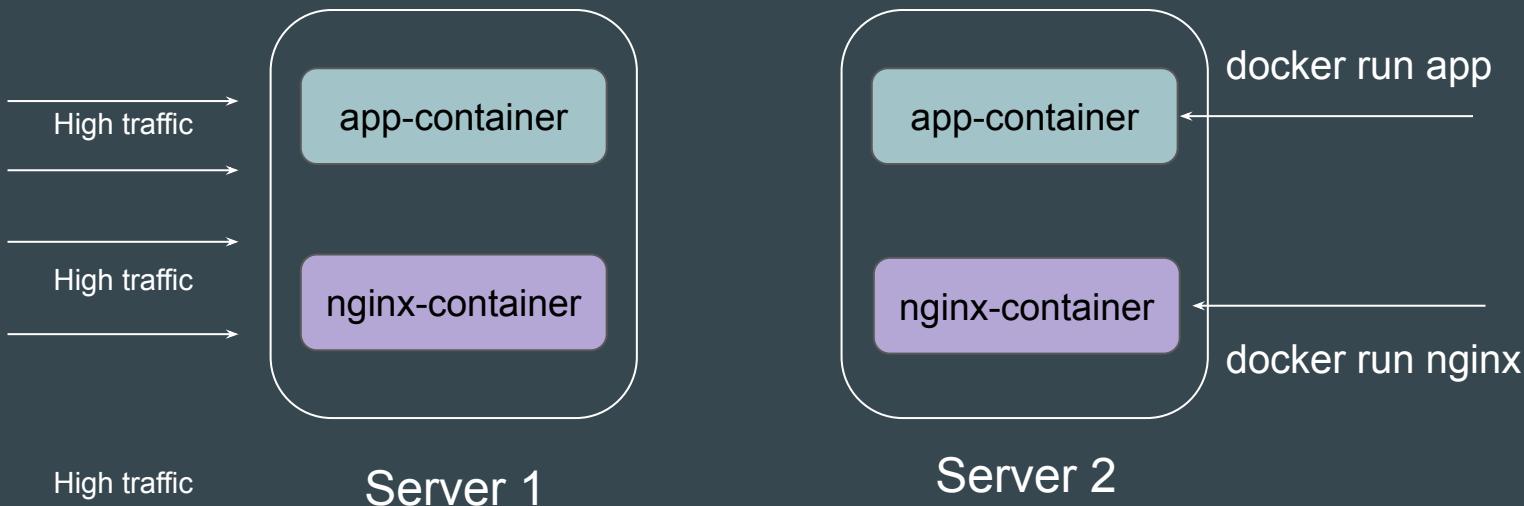
During the initial times, **users relied on manual** container management or basic scripts to handle tasks like deployment, scaling, networking, and monitoring.



# Challenge 1 - Manual Scaling

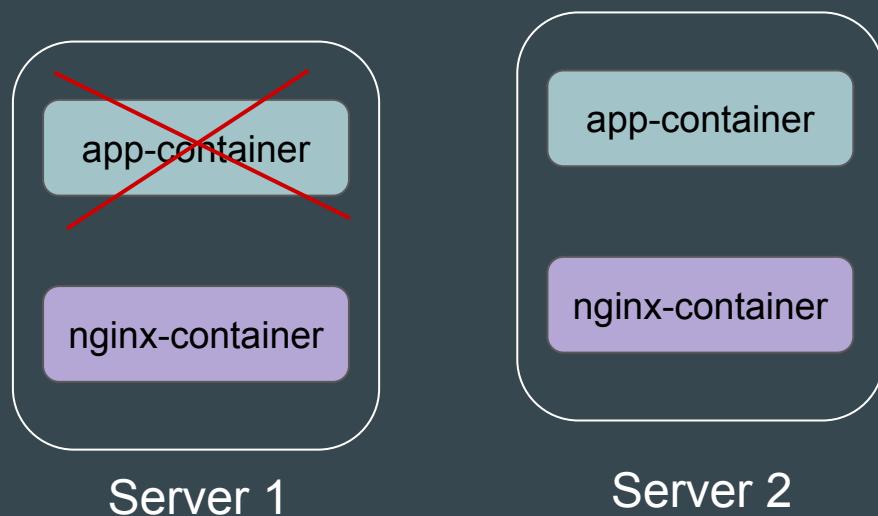
If an application needed to handle higher traffic, developers had to manually start additional containers.

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



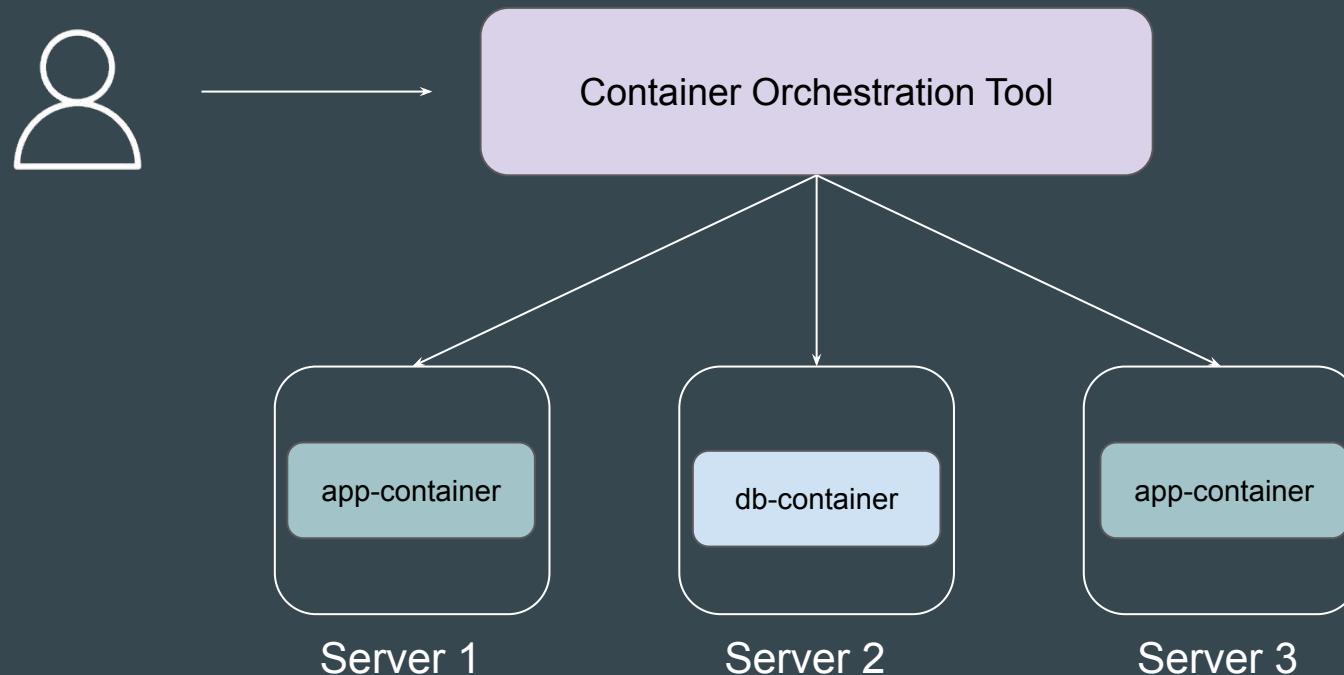
## Challenge 2 - Lack of Fault Tolerance

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.



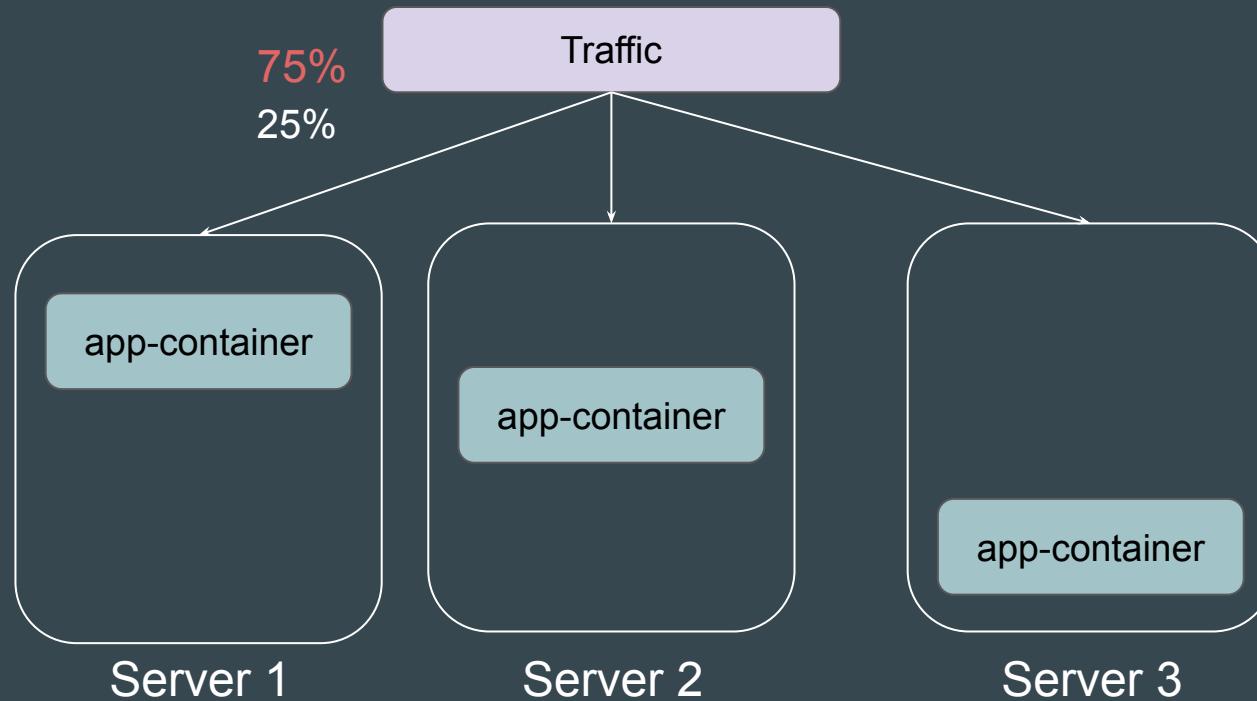
# Introducing Container Orchestration

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



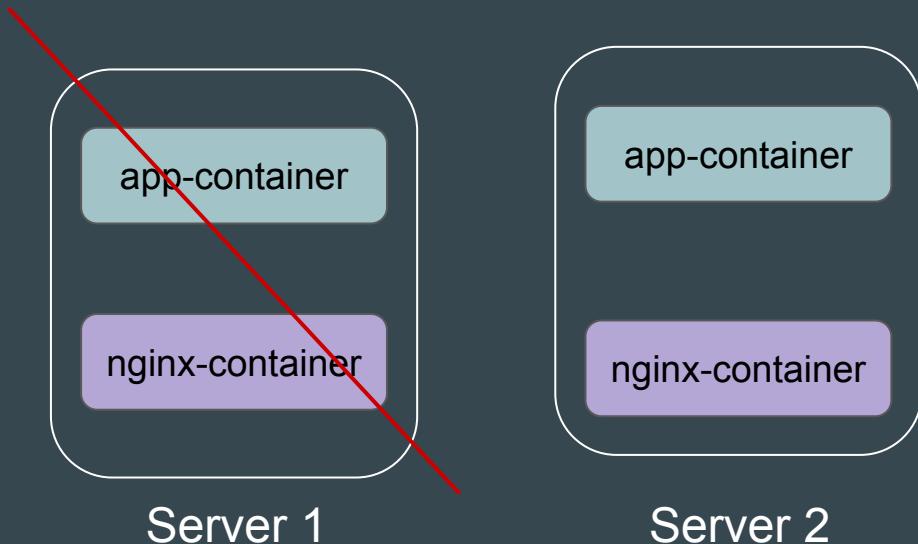
# Solving Challenge - Scaling

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

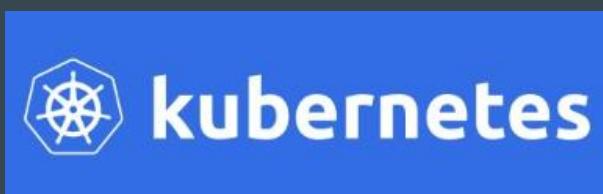
- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Load balancing and traffic routing
- Monitoring container health
- Keeping interactions between containers secure

# 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.

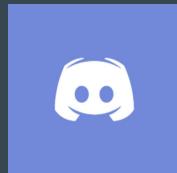
Docker Swarm



Amazon ECS

# Join us in our Adventure

Be Awesome



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



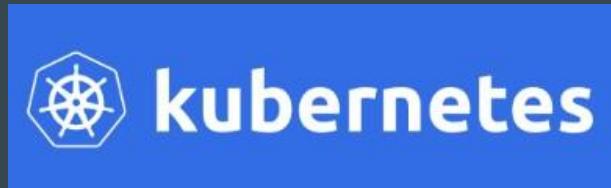
[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Introduction to Kubernetes**

# Setting the Base

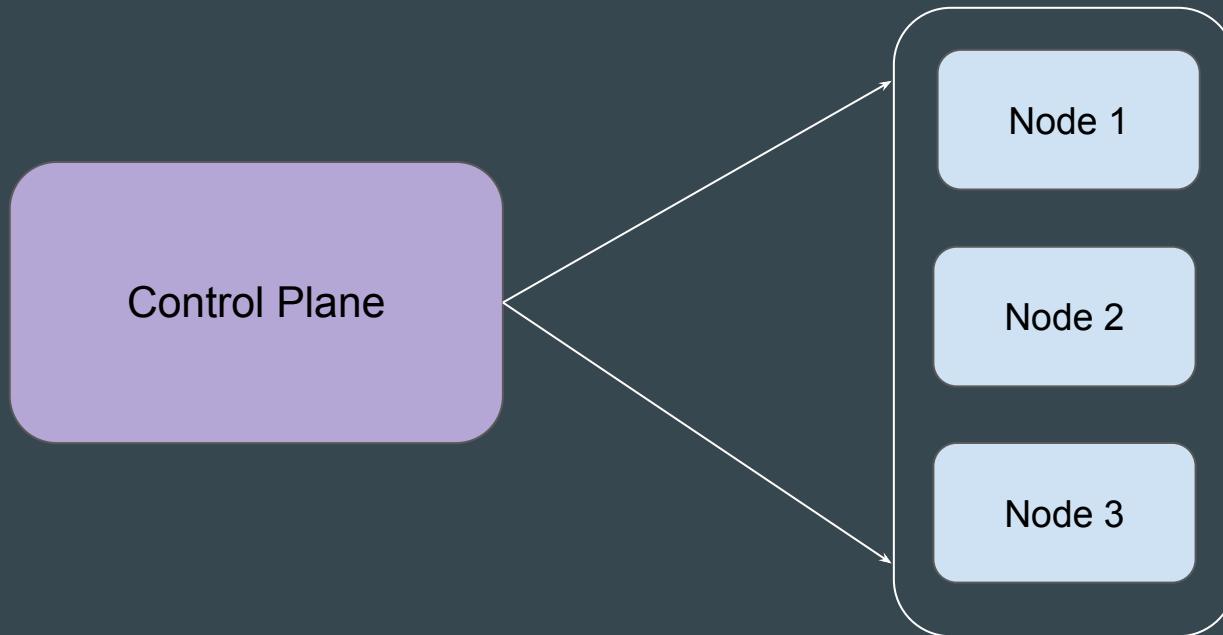
Kubernetes (K8s) is an open-source **container orchestration engine** developed by Google.

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



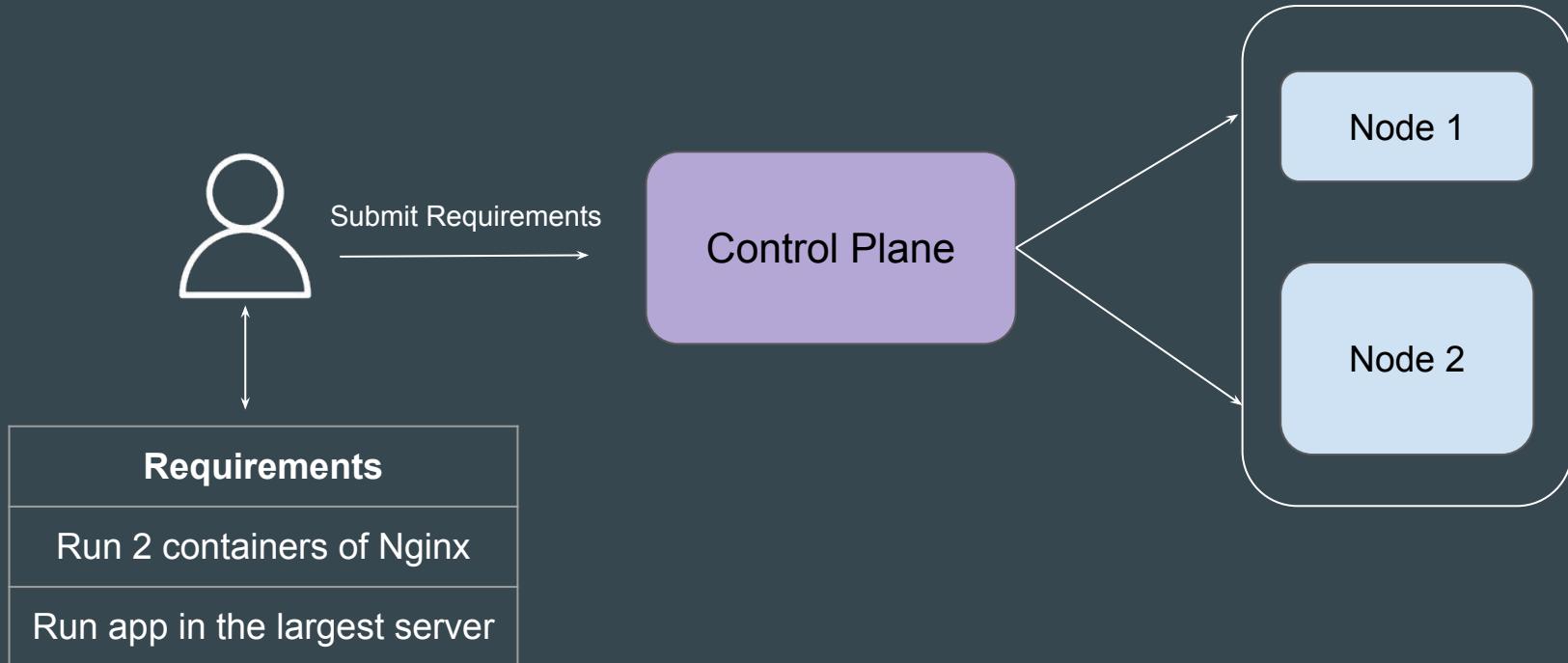
# Architecture of Kubernetes

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



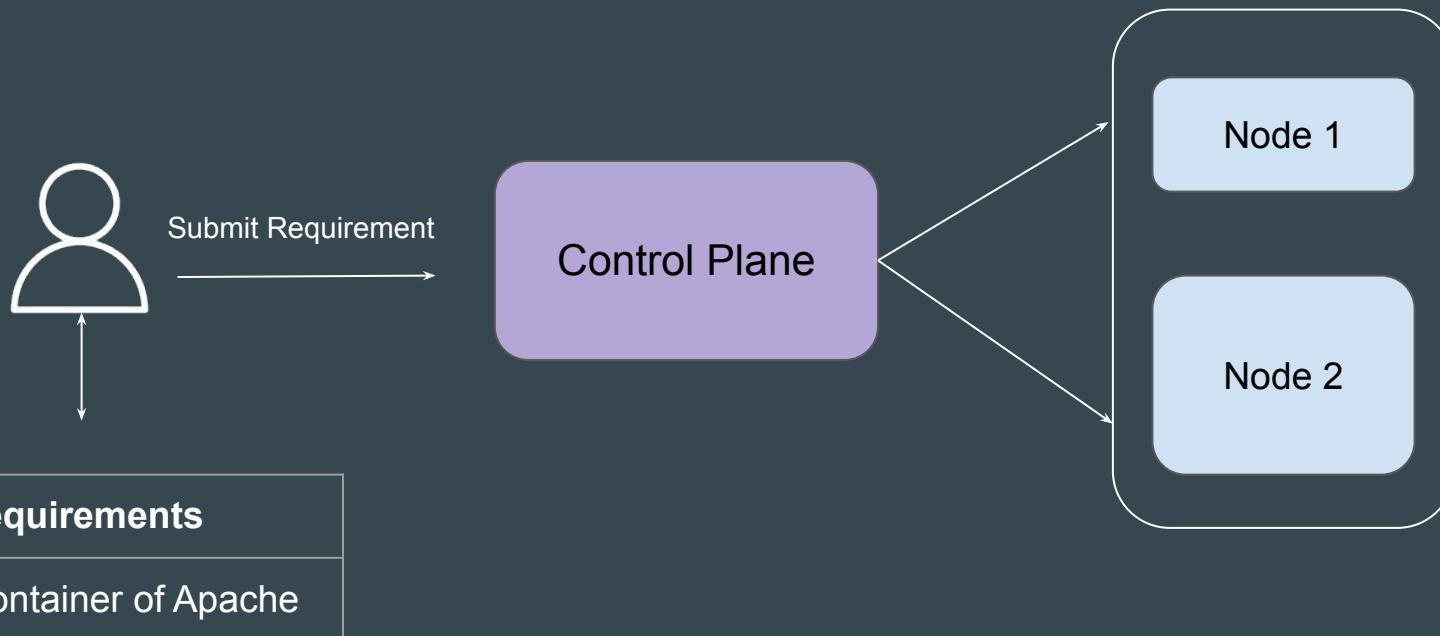
# Basic Workflow

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.

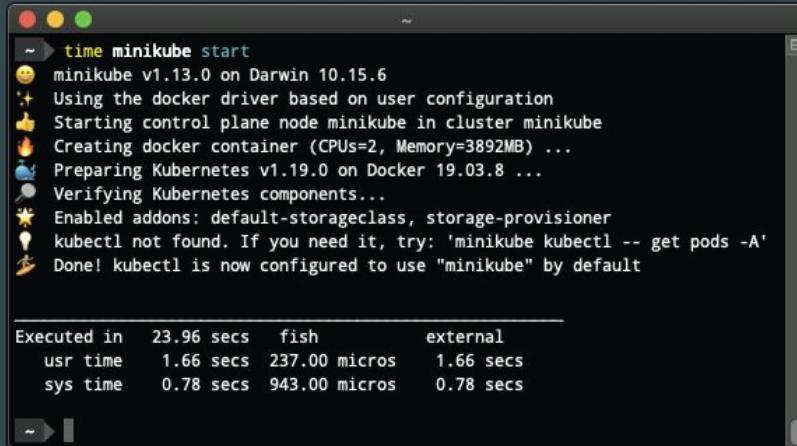
Analogy: Order Ready-Made Food from Store.



# 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.

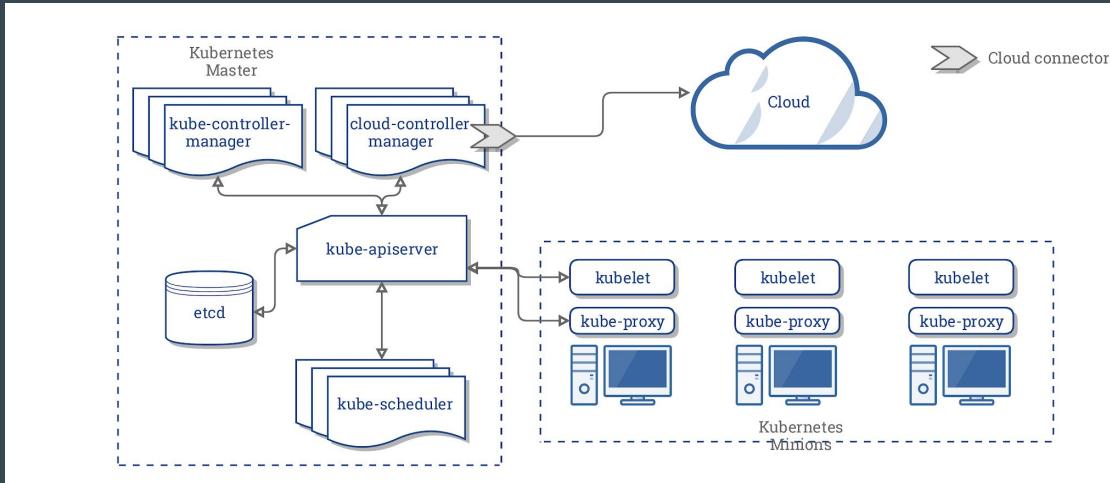


```
~ ➔ time minikube start
😊 minikube v1.13.0 on Darwin 10.15.6
💡 Using the docker driver based on user configuration
👍 Starting control plane node minikube in cluster minikube
🔥 Creating docker container (CPUs=2, Memory=3892MB) ...
🔥 Preparing Kubernetes v1.19.0 on Docker 19.03.8 ...
🔥 Verifying Kubernetes 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

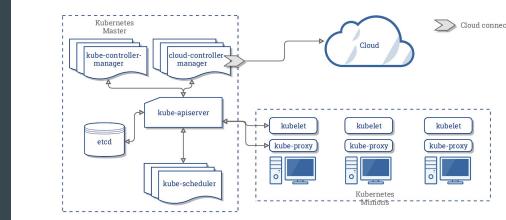
There are **three primary ways** to configure Kubernetes.

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.



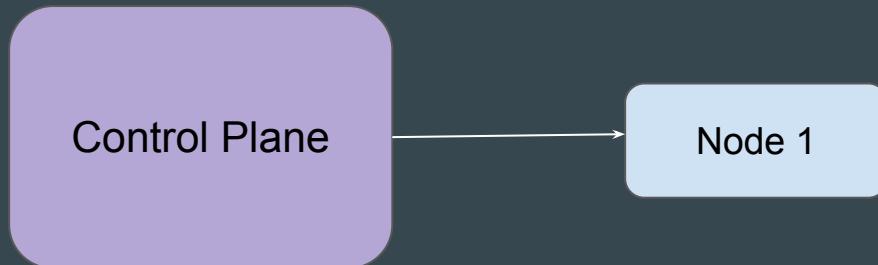
**Kubernetes at scale managed for  
you**

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.

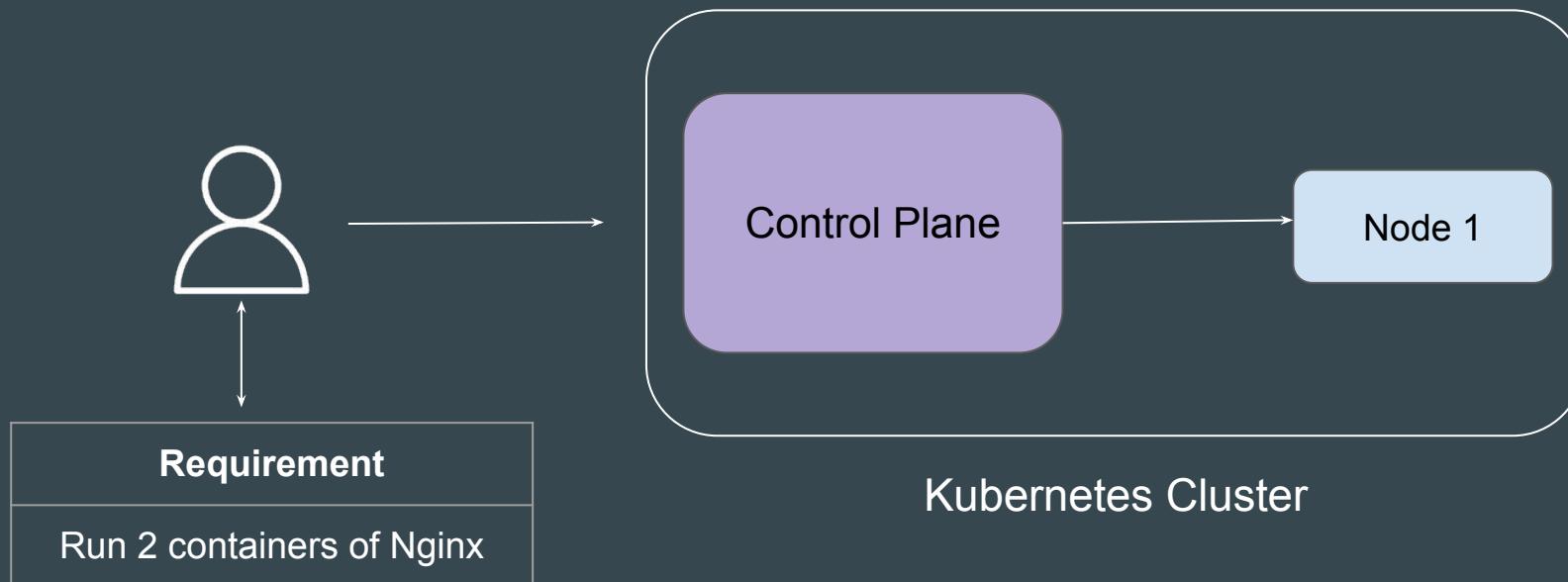
You can literally complete this entire course and ALL practicals for free.



# **Connectivity Options for Kubernetes**

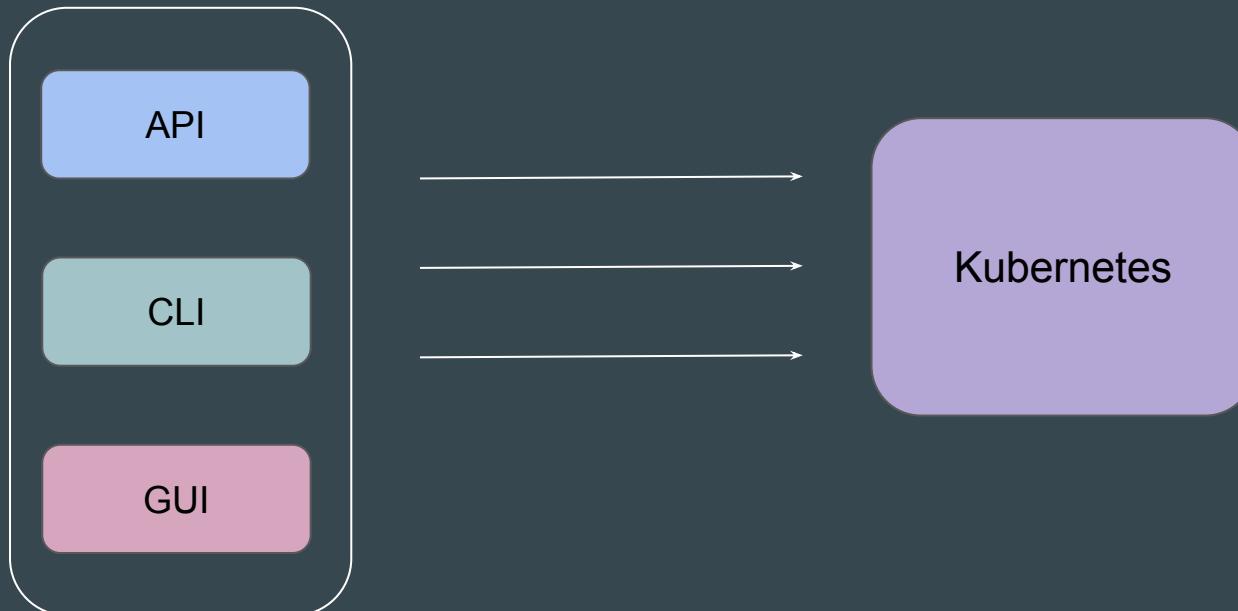
# First Important Step - Connect to Cluster

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.



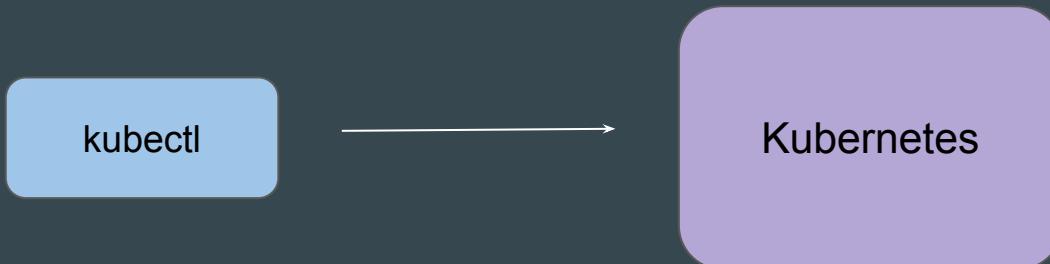
# Option 1 - API

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**

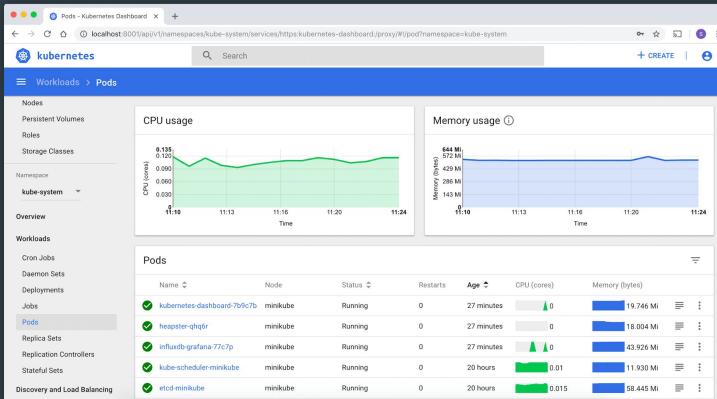


```
root@kubeadm-master:~# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          82s
```

# Option 3 - GUI

Dashboard is a web-based Kubernetes user interface.

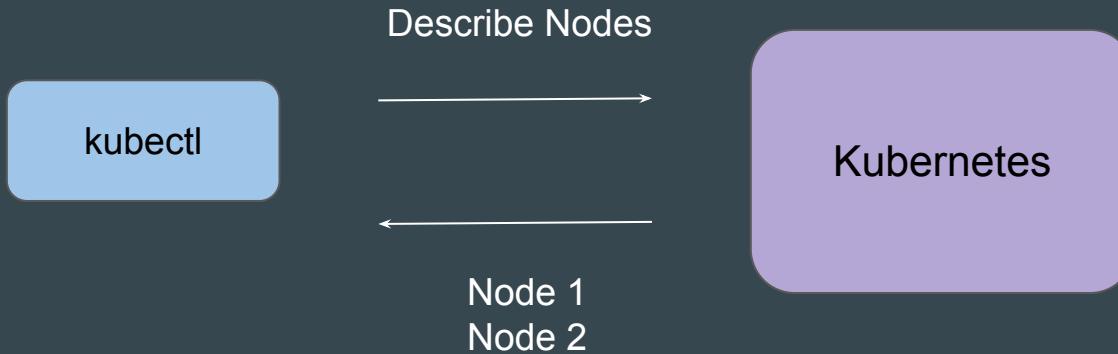
You can use Dashboard to deploy, troubleshoot containerized applications and manage the cluster resources



# **Overview of kubectl**

# Basics of Kubectl

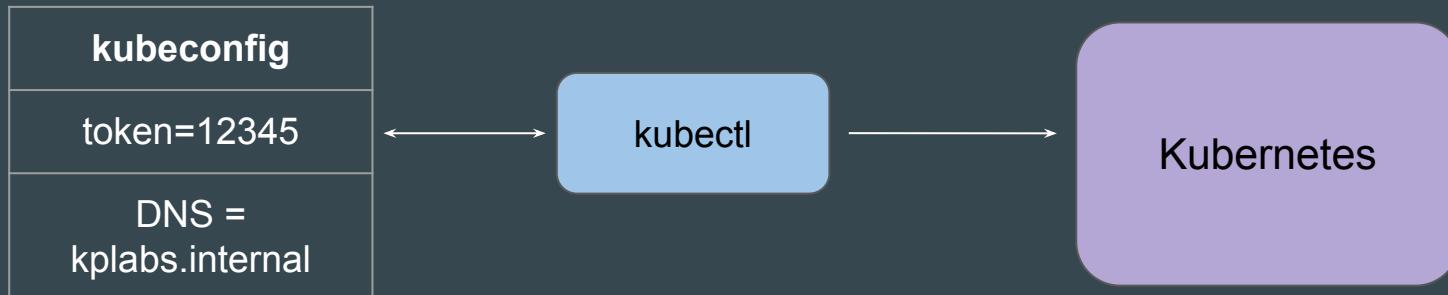
The Kubernetes command-line tool, **kubectl**, allows you to run commands against Kubernetes clusters.



# 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

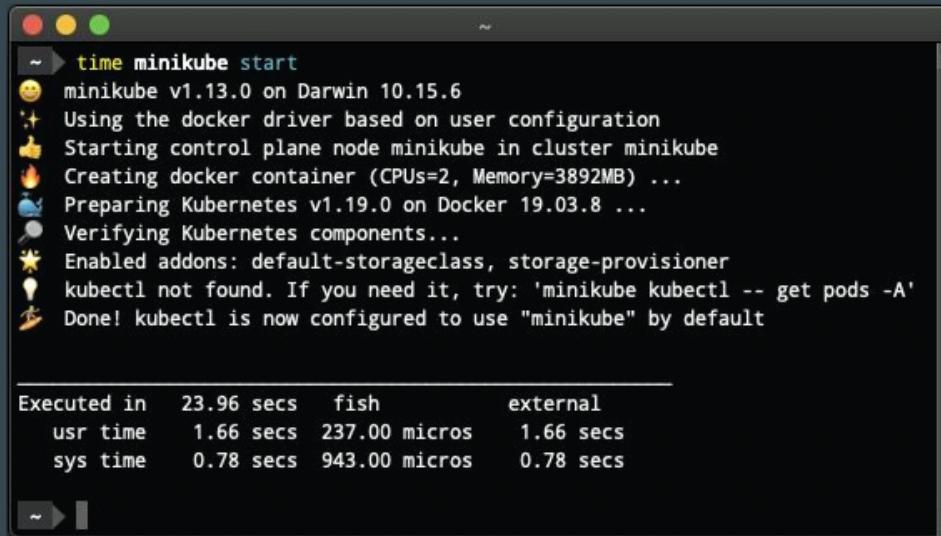
You can also refer to custom config file using the `--kubeconfig` flag

```
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?



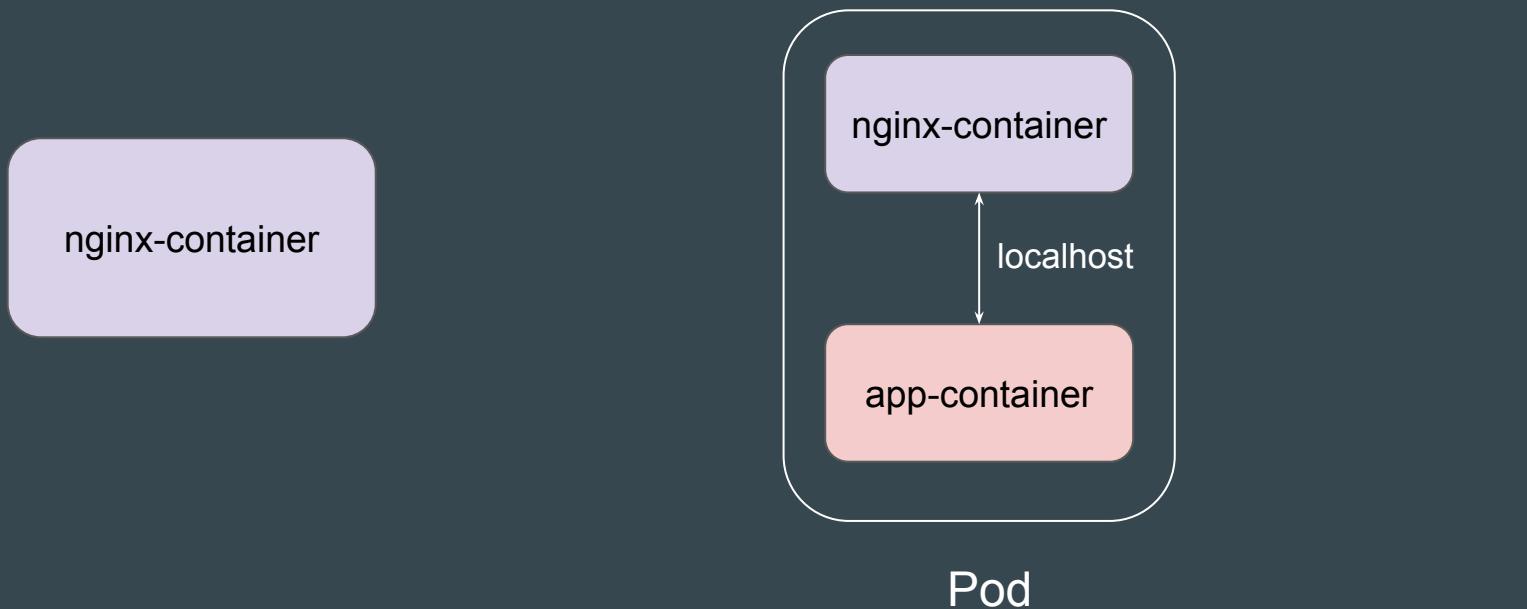
kubectl run nginx --image=nginx



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**

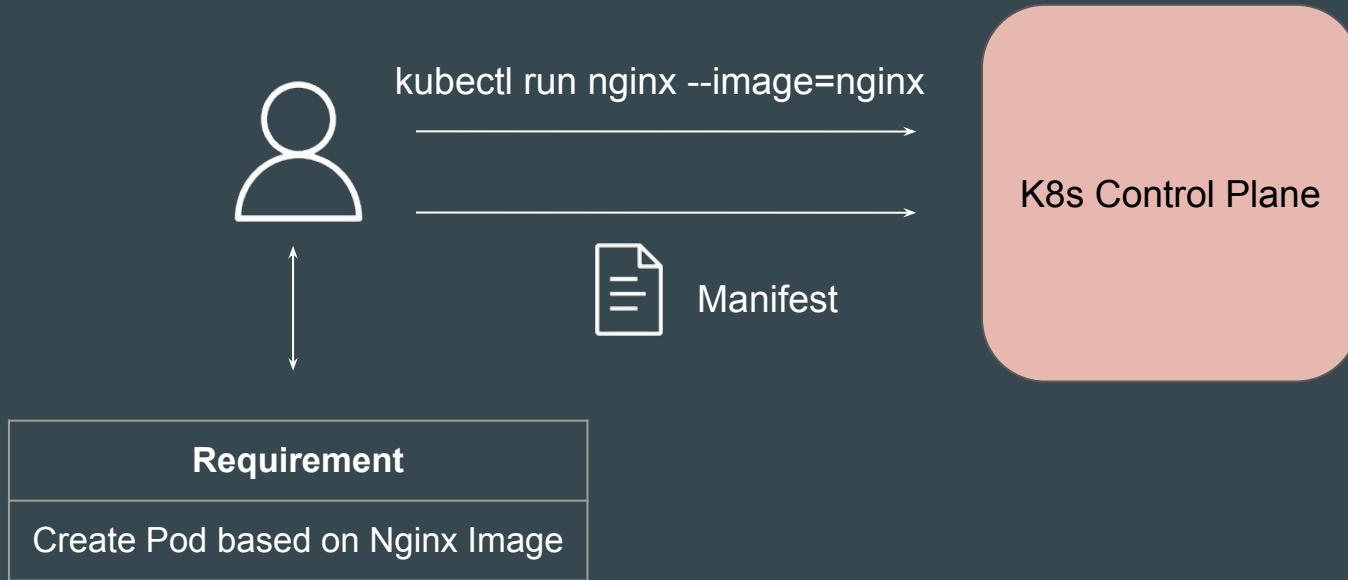
# Setting the Base

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.



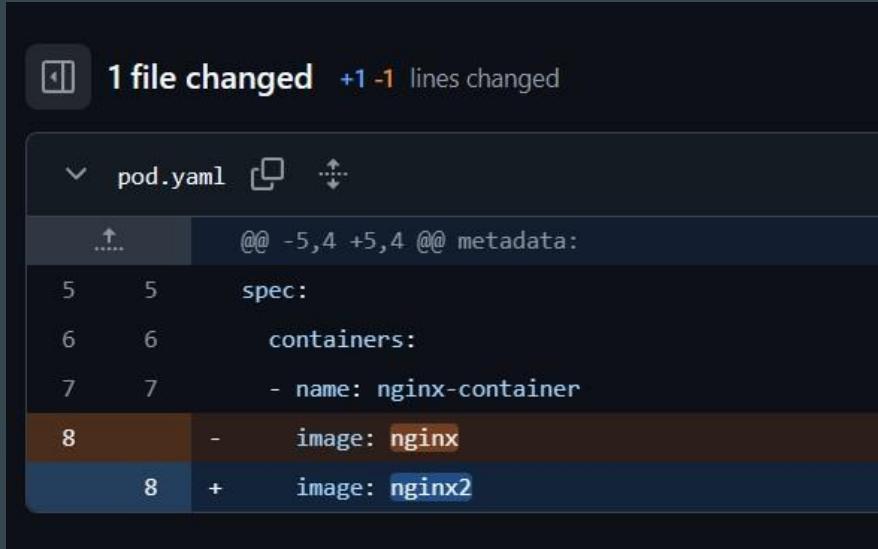
# Manifest File Example

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

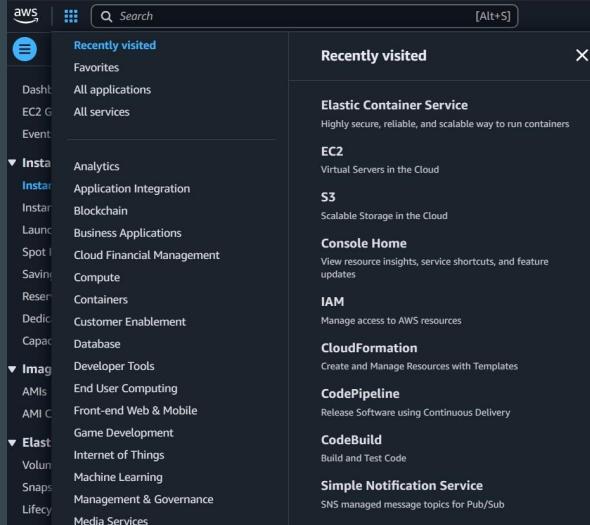
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
```

# **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.

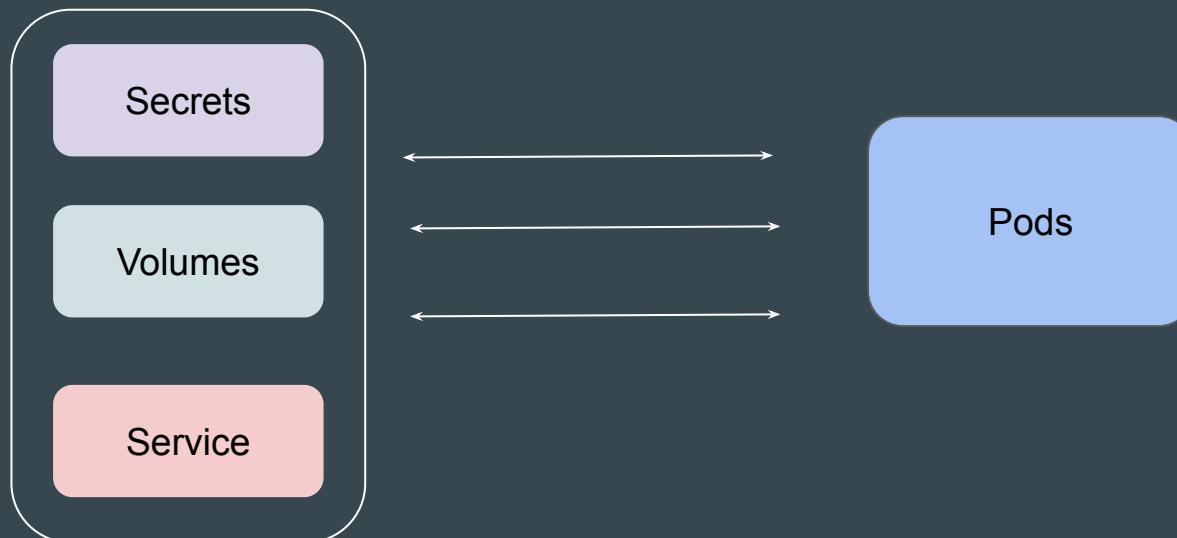
Nowadays, there are 100s of services offered by Cloud providers that aid in the entire lifecycle management of applications and custom use-cases of organizations.



# 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

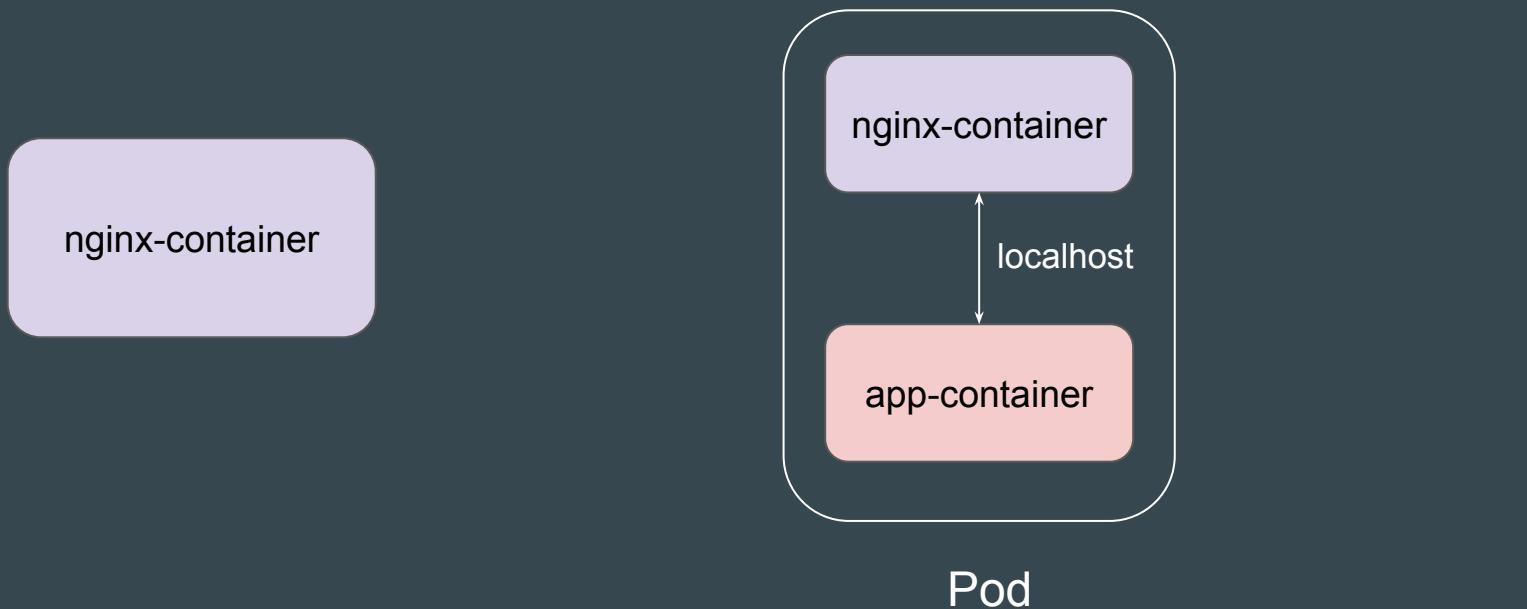
When `--dry-run=client` is combined with `-o yaml`, it will generate manifest file for you associated with the command without actually deploying the object in Kubernetes.

```
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#deletion-timestamps

  deletionGracePeriodSeconds    <integer>
```

---

# EXPOSE

Build once, use anywhere

---

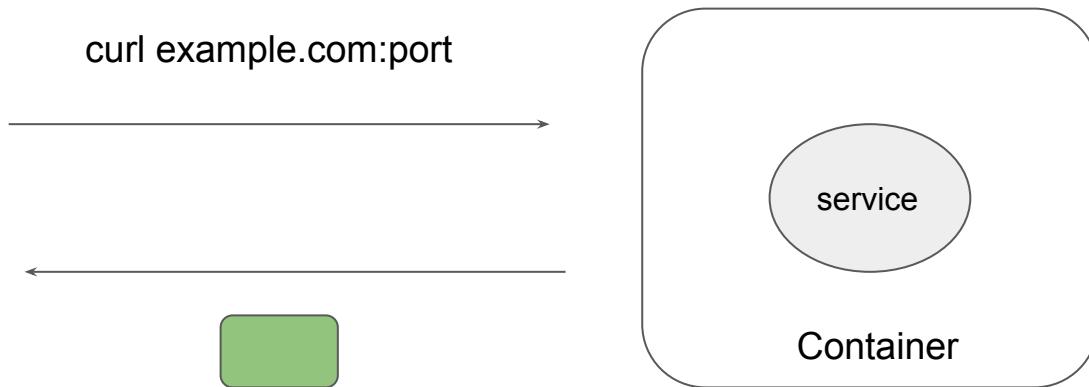
# Overview of EXPOSE instruction

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

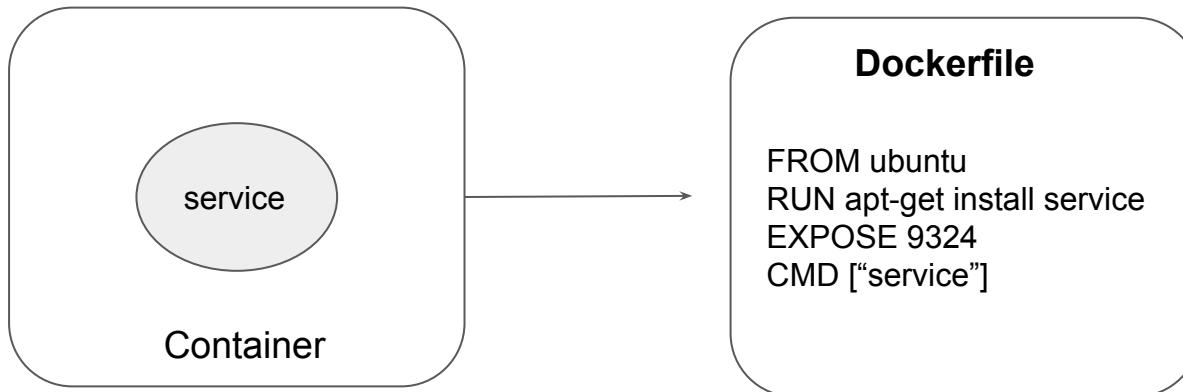
The EXPOSE instruction does not actually publish the port.

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



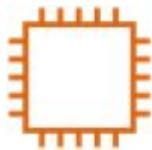
---

# Labels and Selectors

Let's get started

# Overview of Labels

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



Server



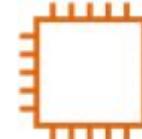
Database



Load Balancer



Load Balancer

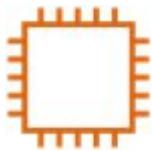


Server



Database

# Adding Labels to Resource



name: kplabs-gateway  
env: production



name: kplabs-db  
env: production



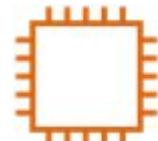
name: kplabs-elb  
env: production



name: kp-elb  
env: dev



name: kp-db  
env: dev



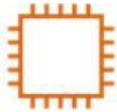
name: kp-gateway  
env: dev

# 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:

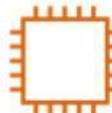
2. Show me all the objects which has label where **env: dev**



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.

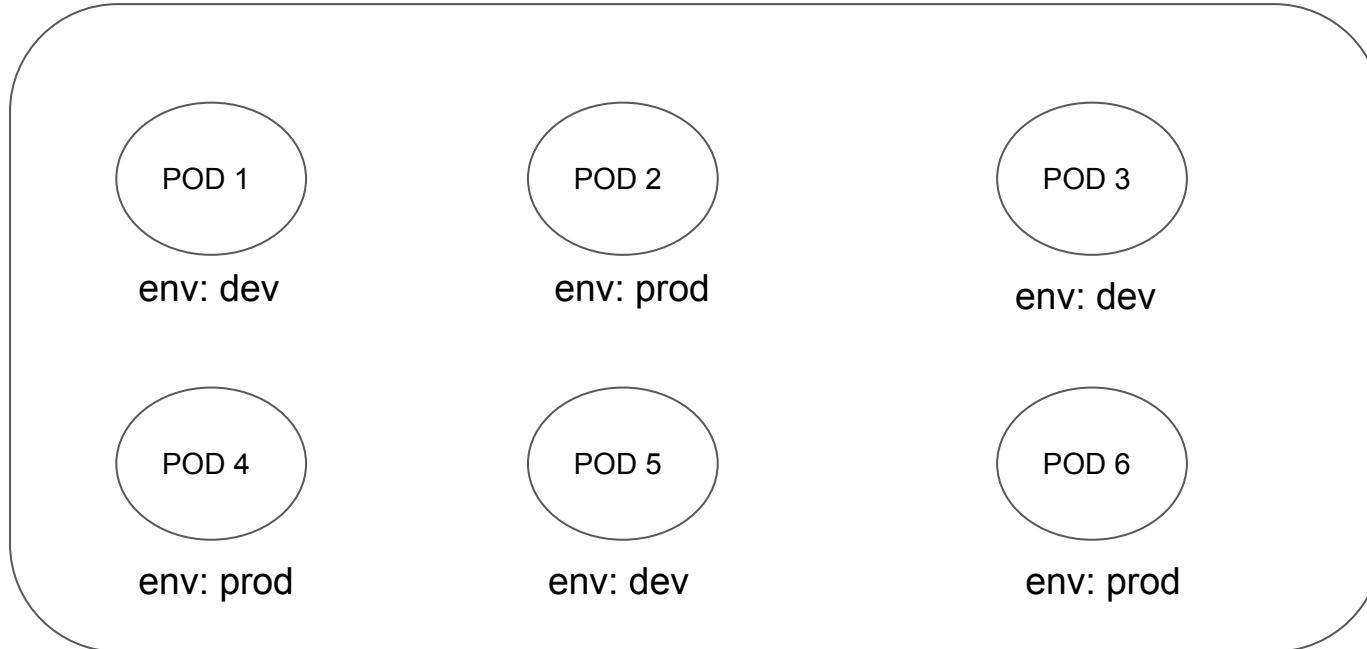
Some of the objects are as follows:

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

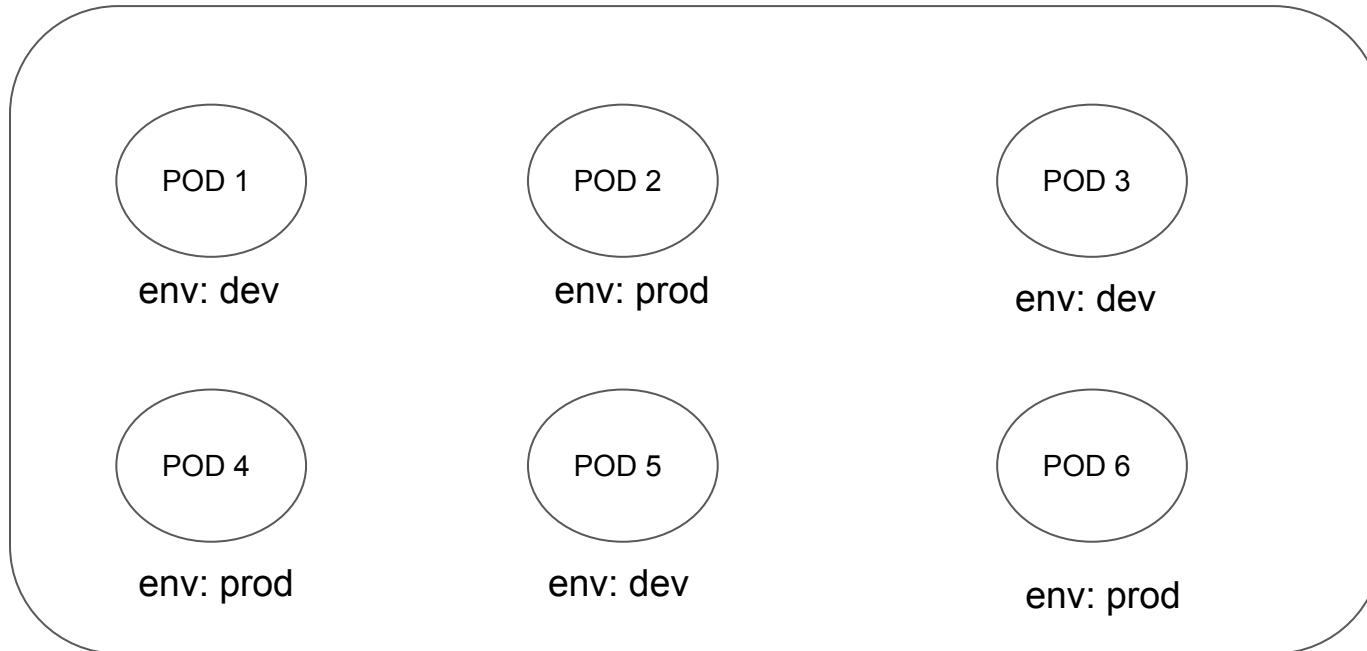
# Kubernetes Perspective



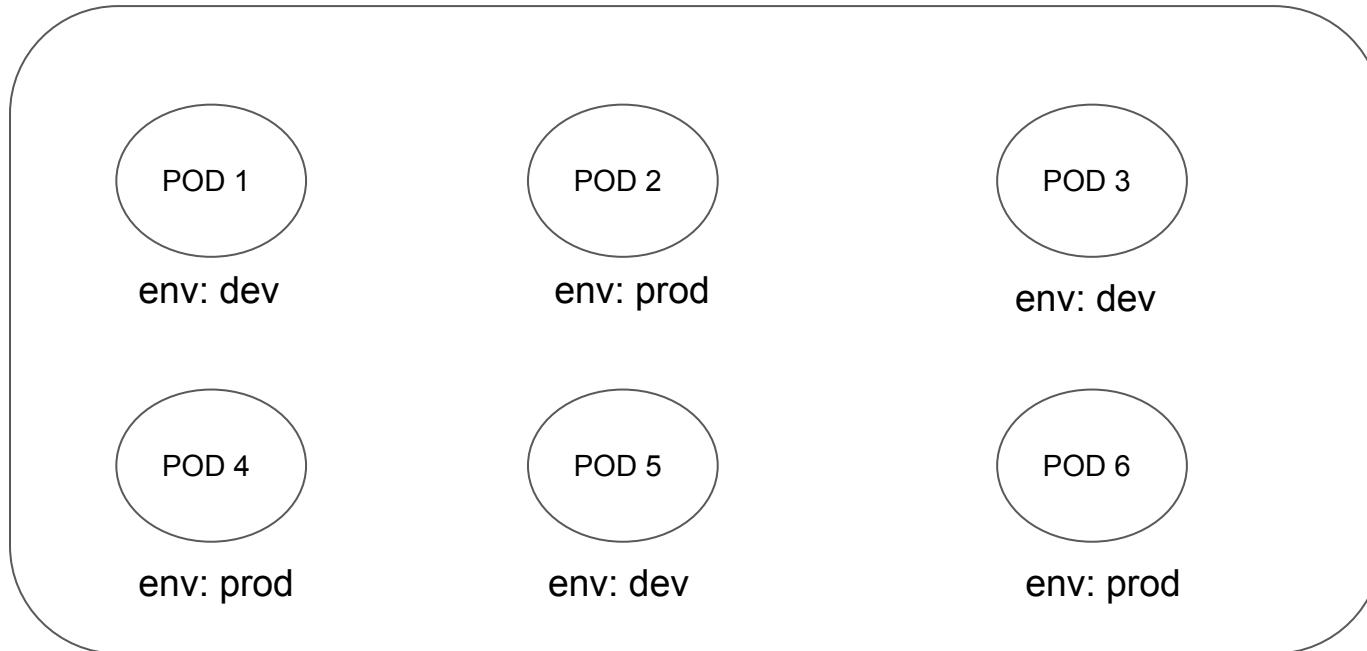
# Assigning Labels to Kubernetes Objects



# Selector: List All Pods from Dev Environment



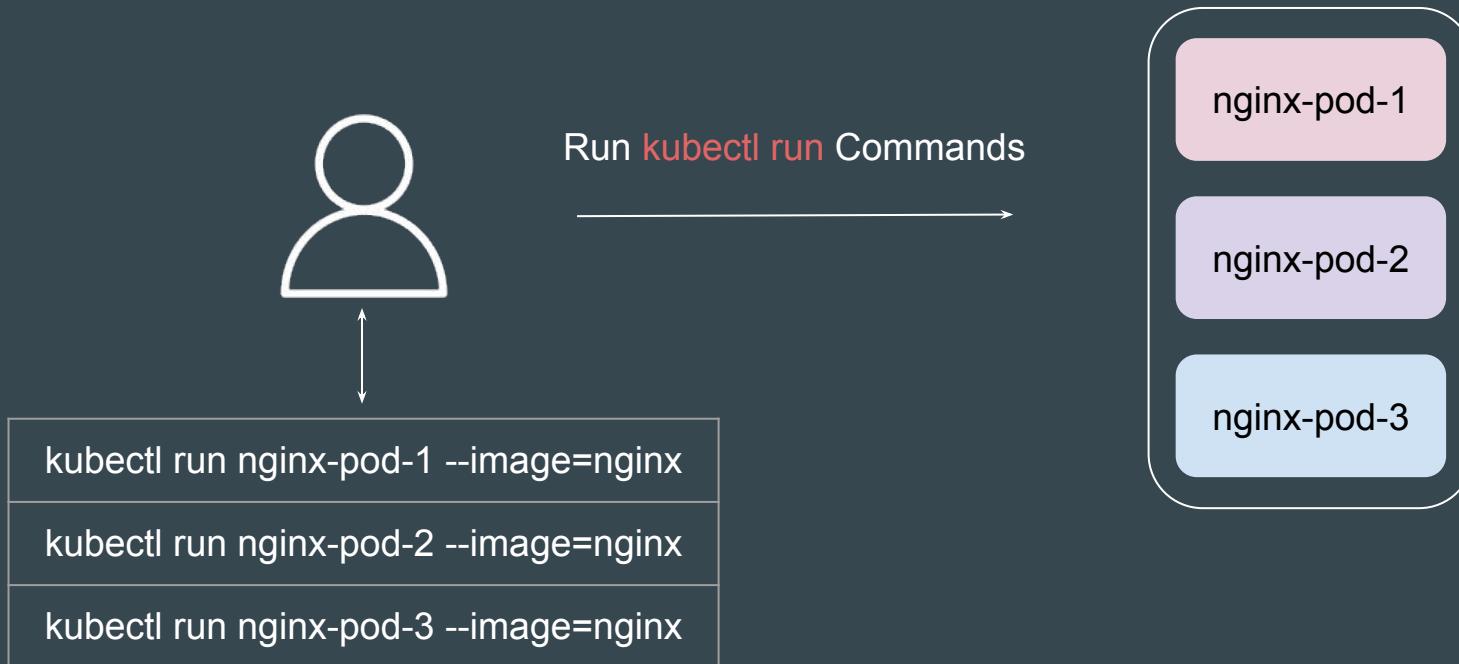
# Selector: List All Pods from Production Environment



# **ReplicaSet**

# Setting the Base

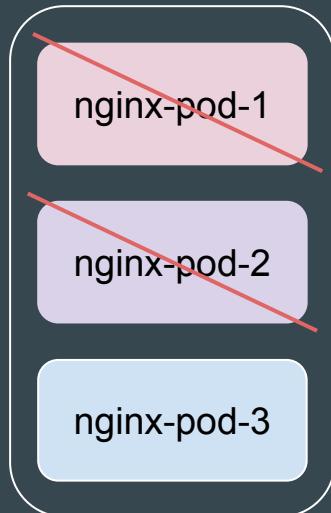
There is a requirement to run 3 Pods based on Nginx image all the time.



# 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

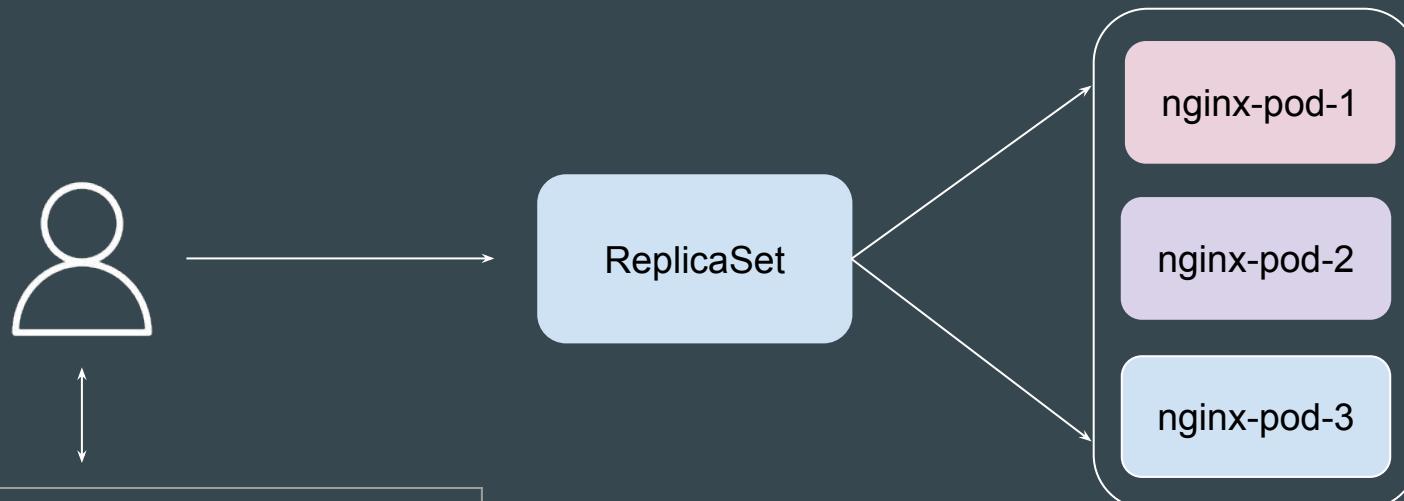
If you need to scale to more or fewer Pods, you would have to manually create or delete Pod definitions and apply the changes.

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.



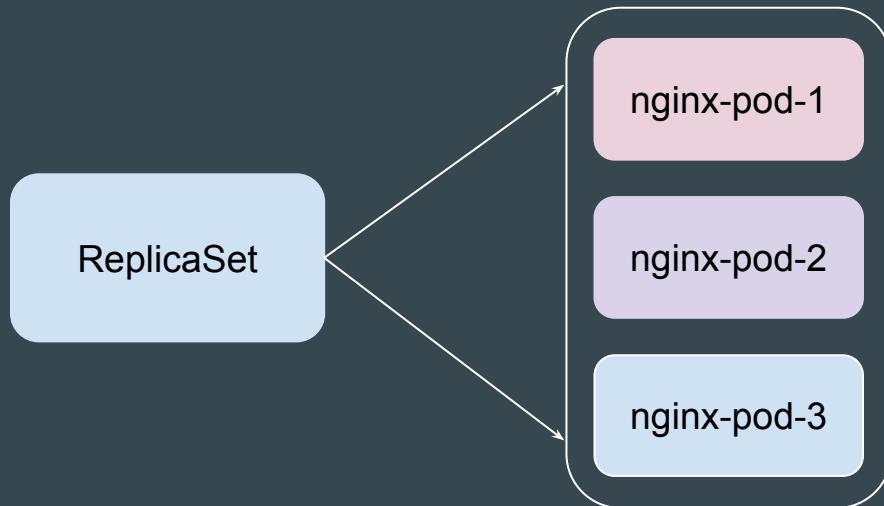
**Requirement**

I need 3 Pods of Nginx Run all the Time

# **Challenges with ReplicaSets**

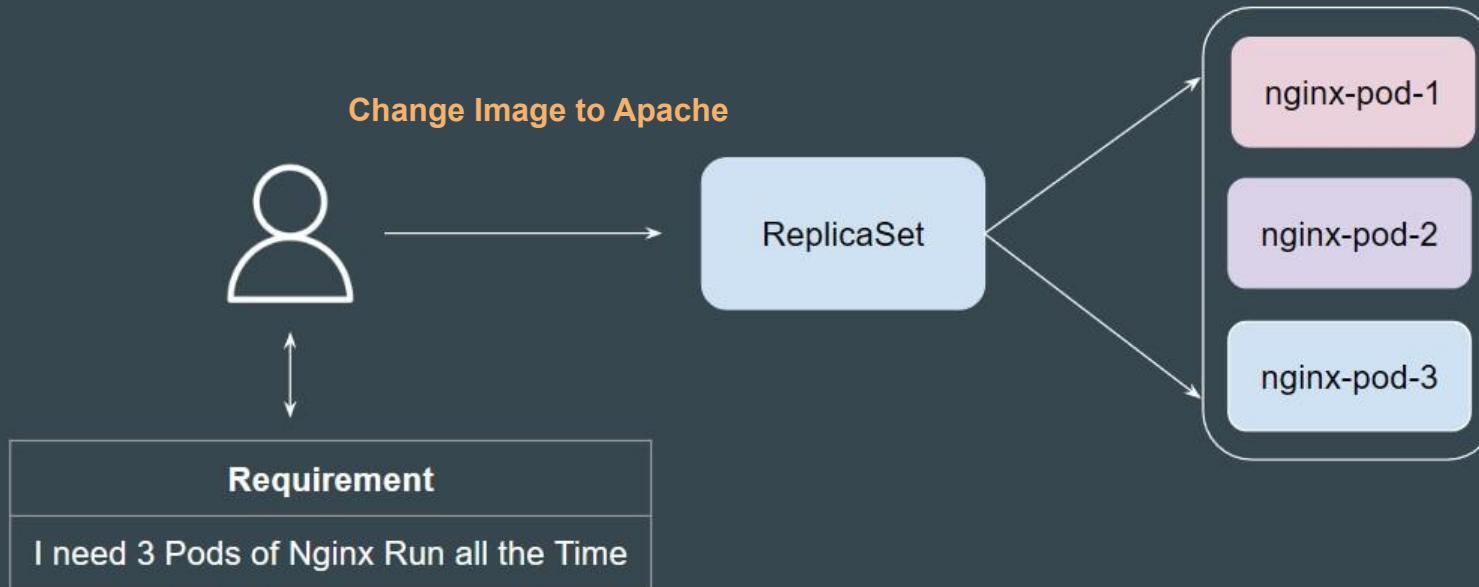
# Setting the Base

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



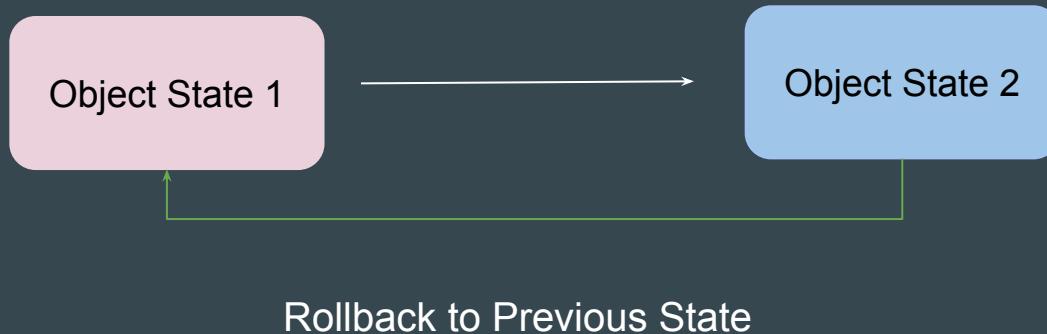
# Challenge 1 - Updating Container Image

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



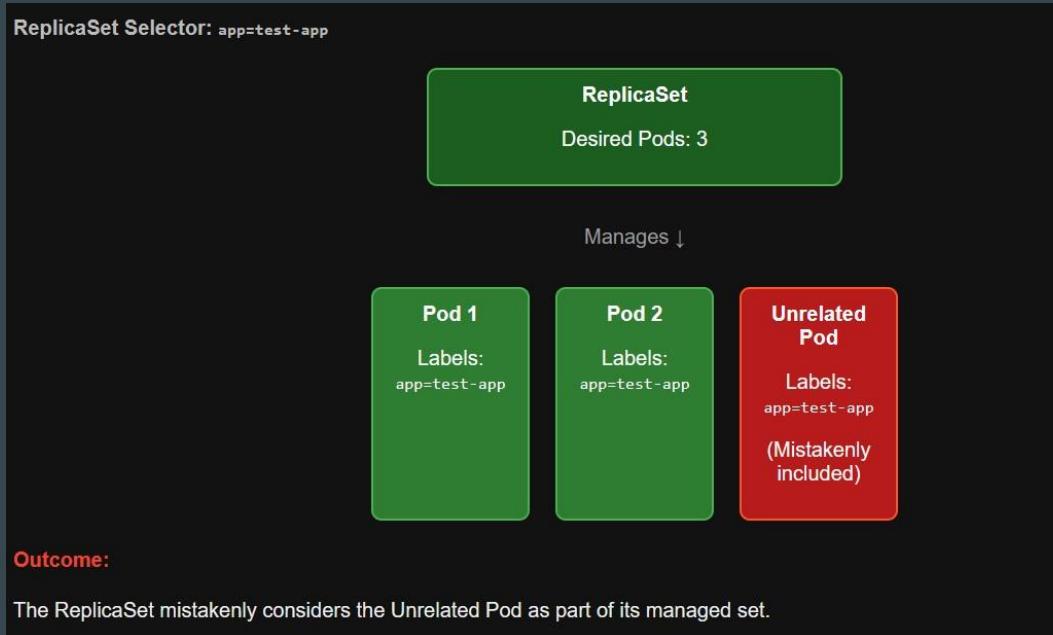
## Challenge 2 - No Built-In Rollback Mechanism

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.

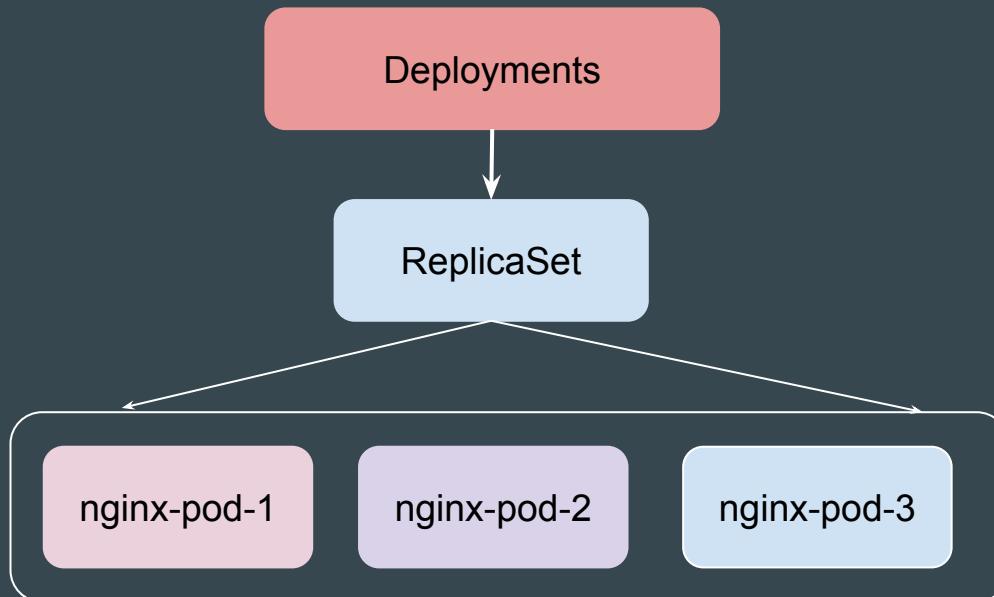
It doesn't just sit on top of the camera; it enhances and extends the camera's functionality, allowing for new perspectives.



# Setting the Base

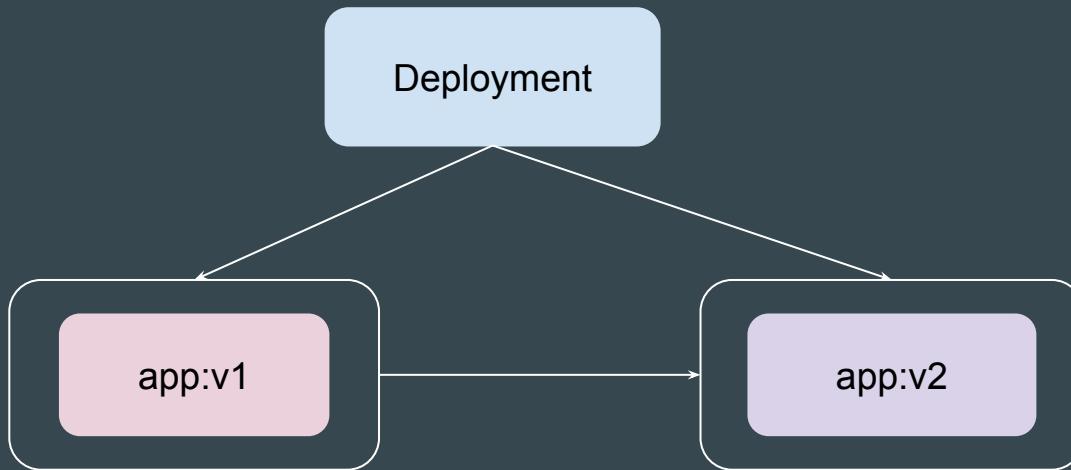
A Deployment is a higher-level abstraction built on top of ReplicaSets.

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



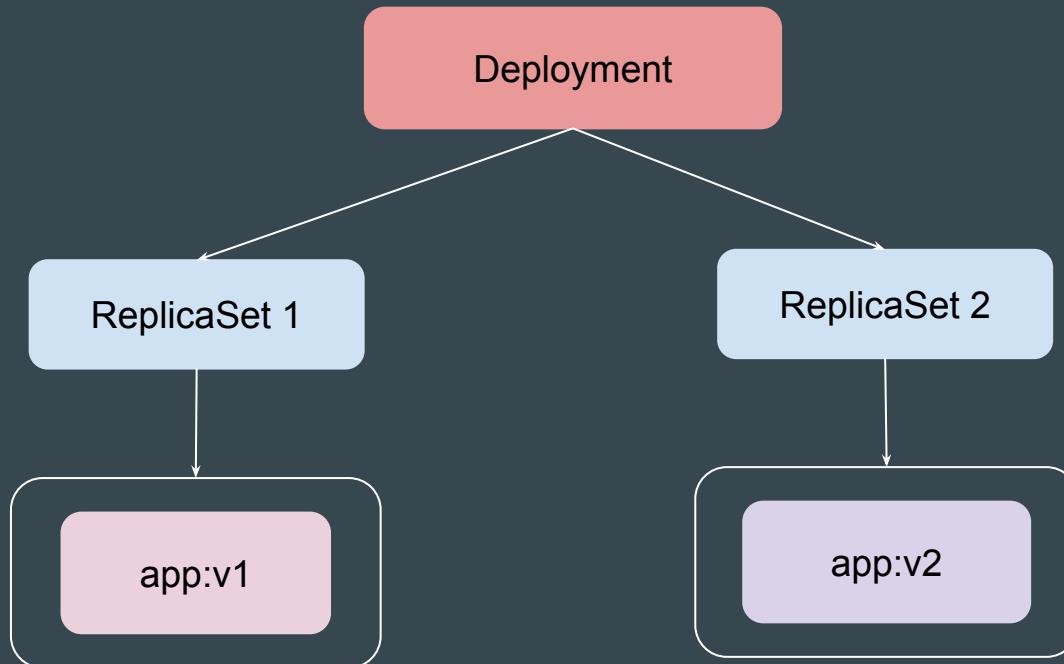
# Use-Case: Update the Container Image

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

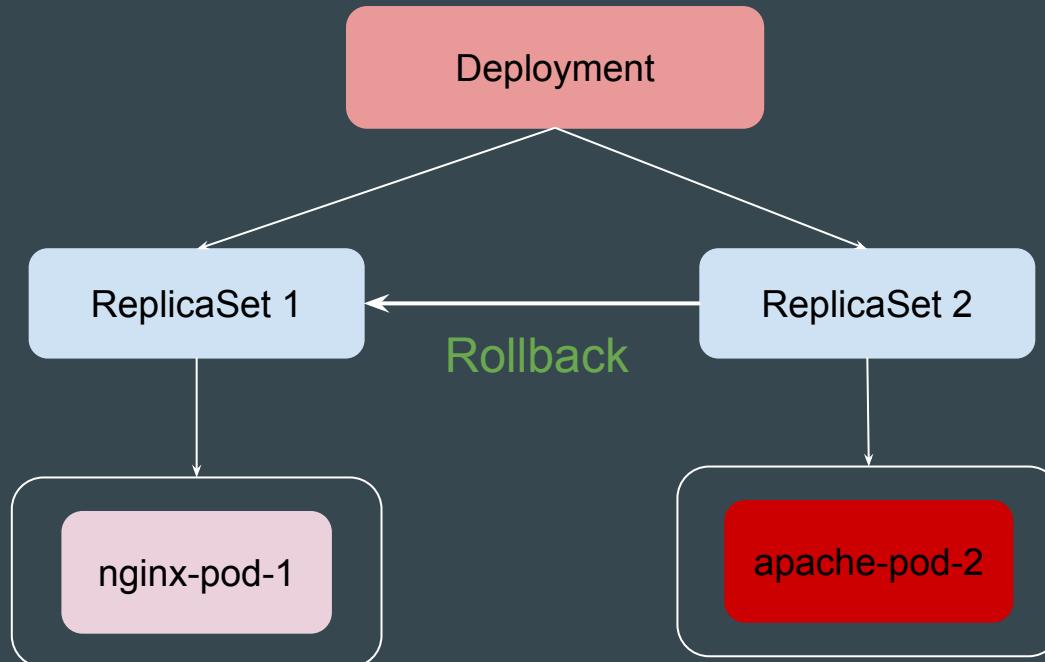
Deployment allows us to inspect the history of your deployments.

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.

---

# maxSurge and maxUnavailable

Deployment Related Options

---

# Overview of Deployment Configuration

While performing a rolling update, there are two important configuration to know.

Configuration Parameter	Description
maxSurge	Maximum Number of PODS that can be scheduled above original number of pods.
maxUnavailable	Maximum number of pods that can be unavailable during the update

# Example Use-Case

Total PODS in Deployment 8.

maxSurge	maxUnavailable
25%	25%

maxSurge = 2 PODS

maxUnavailable = 2 PODS

At Most of 10 PODS (8 current pods + 2 maxSurge pods)

At Least of 6 PODS (6 current pods - 2 maxUnavailable pods)

# Note

You can define the maxSurge and maxUnavailable in both the numeric and percentage terms;

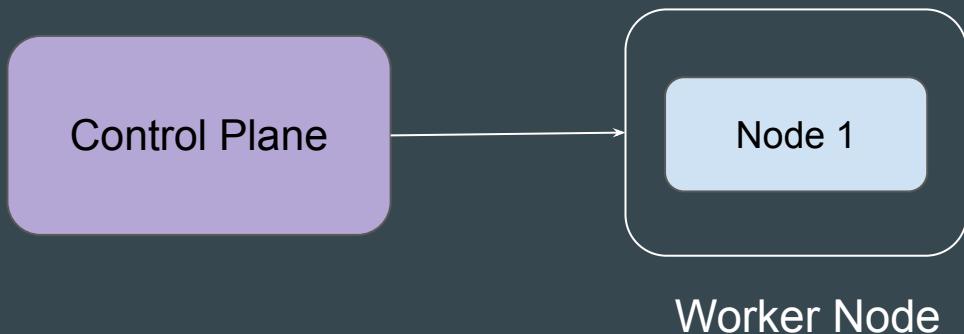
maxUnavailable: 0

maxSurge: 10%

# **Multiple Worker Nodes for Kubernetes**

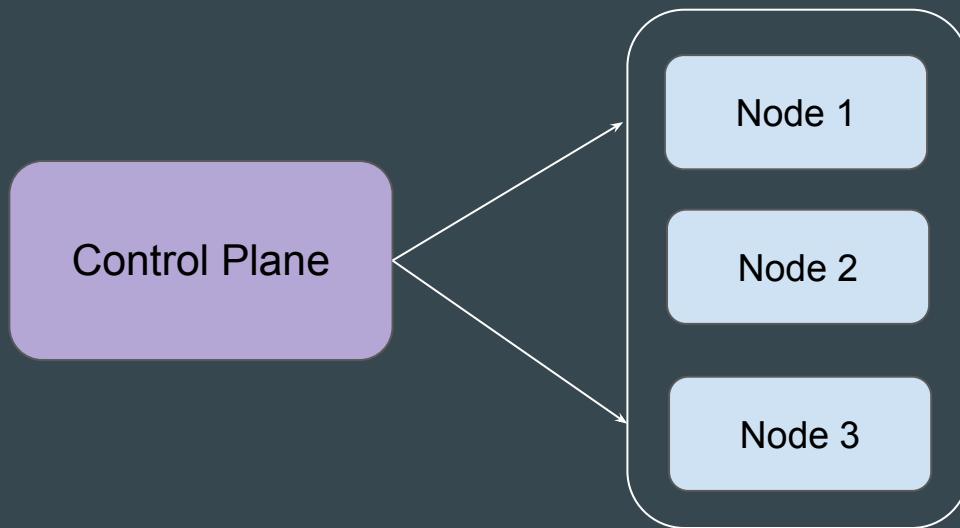
# Understanding the Challenge

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.

# **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

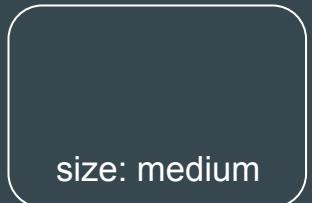
Deployment allows us to inspect the history of your deployments.

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

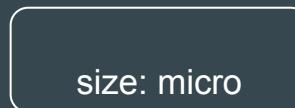
Add appropriate labels to your nodes depending on their hardware type.



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`

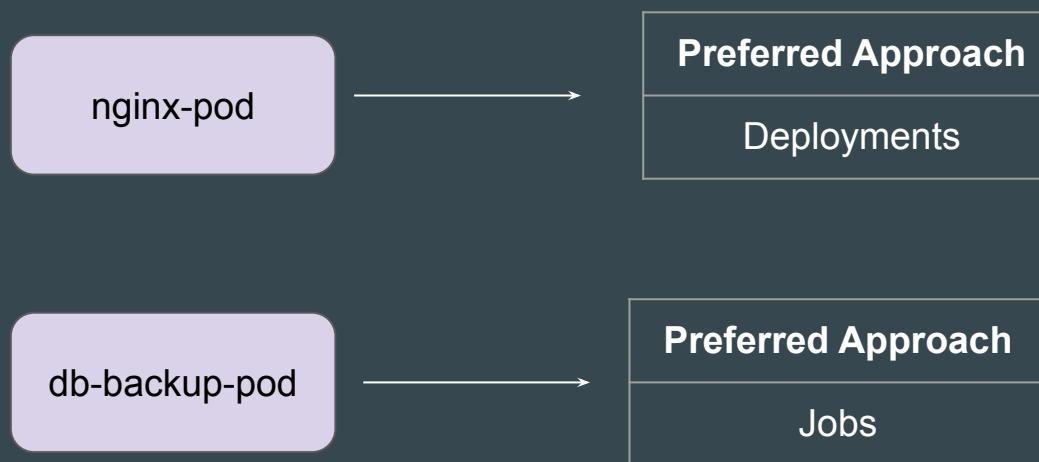
```
! nodeSelector.yaml
apiVersion: v1
kind: Pod
metadata:
  name: memory-intensive-app
spec:
  containers:
  - name: app
    image: nginx
  nodeSelector:
    size: Large
```

# **Overview of Jobs and CronJob**

# Setting the Base

Pods like web-servers are intended to run all the time.

Some Pods are intended to run for limited amount of time to complete some specific task like sending batch emails, database backups and then complete.



# Understand Manual Approach

Use-Case: Processing a large dataset, and generating reports.

Manual Approach:

1. Create a Pod that performs the task.
2. Monitor the Pod, and handle retries to task if it fails.
3. Once completed, verify the logs.
4. Delete the Pod

# Introduction to Jobs

Jobs automates the manual process for one time tasks, ensuring your task runs to completion and providing features like automatic retries and parallel execution.

Once the task is done, the Job is considered completed.

```
C:\kplabs-k8s>kubectl get jobs
NAME      STATUS   COMPLETIONS   DURATION   AGE
hello-job  Complete  1/1          10s        13s
```

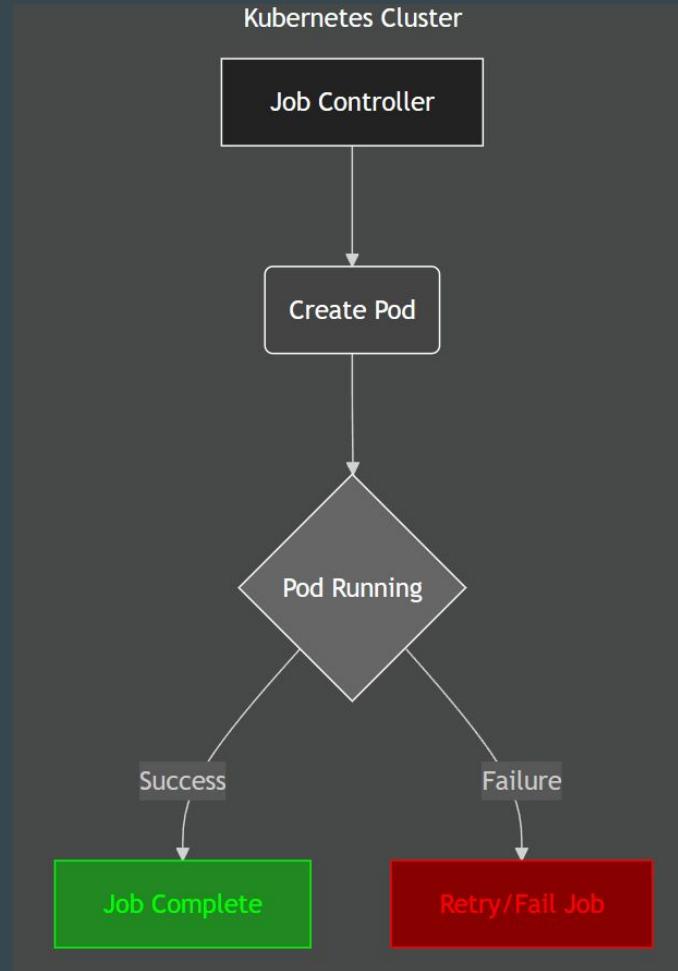
## Point to Note

A simple case is to create one Job object in order to reliably run one Pod to completion.

The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot).

```
C:\kplabs-k8s>kubectl delete pod ping-job-5hkrq
pod "ping-job-5hkrq" deleted

C:\kplabs-k8s>kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
ping-job-648lf 1/1     Running   0          8s
```



# CronJob

A **CronJob** creates Jobs on a **repeating schedule**.

You define a schedule using the familiar Cron format, and Kubernetes will automatically create and run Jobs at those specified times.



CronJob

Run DB Backup Job  
every day at 11PM



db-backup-job

```
C:\>kubectl get cronjob
NAME          SCHEDULE      TIMEZONE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
backup-cronjob 0 23 * * *  <none>     False      0        <none>   83s
```

# **Practical - Jobs**

# Setting the Base

You can use the `kubectl create job` command to create Jobs.

```
C:\>kubectl create job --help
Create a job with the specified name.

Examples:
# Create a job
kubectl create job my-job --image=busybox

# Create a job with a command
kubectl create job my-job --image=busybox -- date

# Create a job from a cron job named "a-cronjob"
kubectl create job test-job --from=cronjob/a-cronjob
```

# Reference Manifest File

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ping-job
spec:
  template:
    spec:
      containers:
        - name: hello-container
          image: busybox:latest
          command: ["ping", "-c", "30", "google.com"]
      restartPolicy: Never
```

# Point to Note - Restart Policy

In a Job manifest, only a `RestartPolicy` equal to `Never` or `OnFailure` is allowed.

RestartPolicy	Description
OnFailure	Pods will restart only if they fail (exit with a non-zero status). This is useful for retrying failed tasks.
Never	Pods will not restart regardless of whether they fail or succeed. This is useful for tasks that should not be retried automatically.

# Why is Always Not Allowed

Unlike Deployments or ReplicaSets, a Job **does not support** restartPolicy: Always.

This is because Jobs are designed for finite, short-lived workloads, not for long-running services that need constant availability.

# **activeDeadlineSeconds in Jobs**

# Understanding the Challenge

A Job can have a bug that causes it to loop infinitely or get stuck.

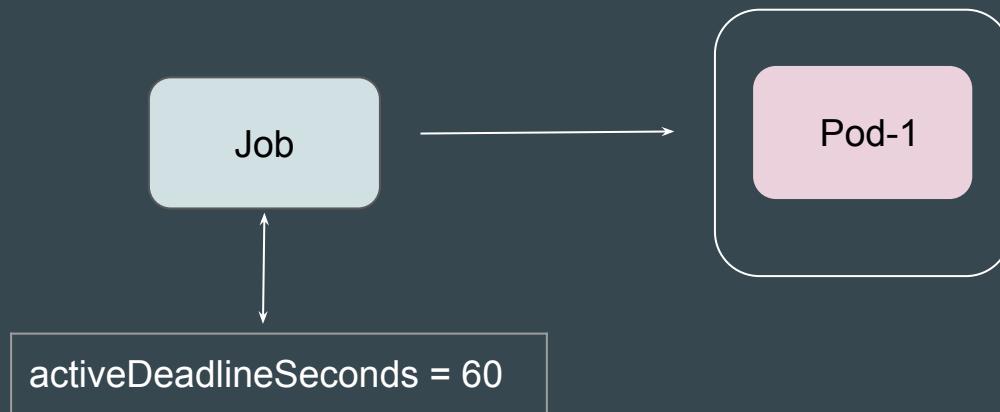
This will lead to unnecessary resource usage for longer duration of time.



# Introduction to the Topic

The `activeDeadlineSeconds` field in a Kubernetes Job specifies the maximum duration a Job can actively run.

If the Job doesn't complete within this time, Kubernetes terminates all its Pods and marks the Job as failed with a deadline exceeded condition.



## Point to Note

It's a crucial mechanism for preventing Jobs from running indefinitely due to unforeseen issues or bugs.

# Reference Manifest File

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ping-job-manifest
spec:
  activeDeadlineSeconds: 30
  template:
    metadata:
      spec:
        containers:
          - command:
              - ping
              - -c
              - "100"
              - google.com
            image: busybox:latest
            name: ping-job
  restartPolicy: Never
```

# **Practical - CronJob**

# Setting the Base

You can use the `kubectl create cronjob` command to create CronJob.

Specifying `--schedule` is important.

```
C:\>kubectl create cronjob --help
Create a cron job with the specified name.

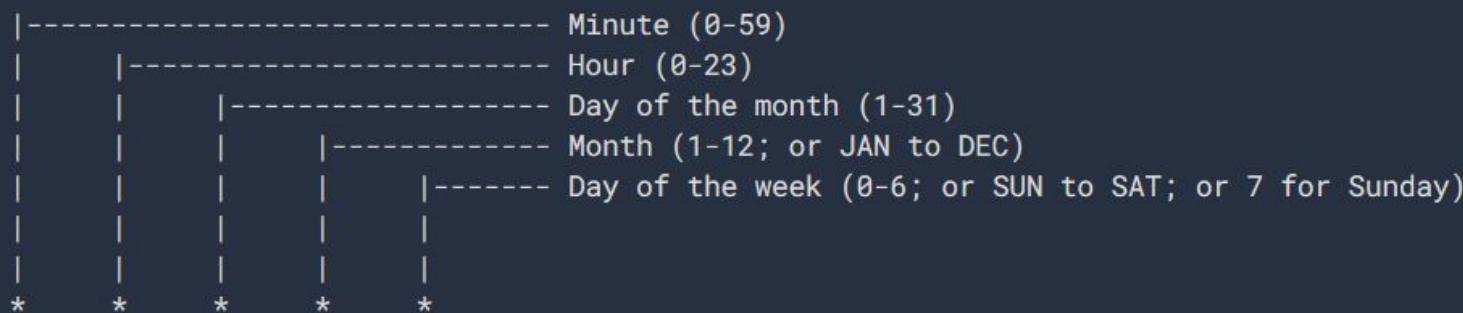
Aliases:
cronjob, cj

Examples:
# Create a cron job
kubectl create cronjob my-job --image=busybox --schedule="*/1 * * * *"

# Create a cron job with a command
kubectl create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

# Format of Schedule

The schedule is written in Cron format.



Crontab Schedule	Explanation
* * * * *	<p>The asterisk (*) in each position means "every."</p> <p>This schedule translates to "every minute of every hour of every day of every month of every day of the week." It runs a job every single minute.</p>
0 * * * *	<p>The first field (minutes) is set to 0. This means "at minute 0 of every hour." So, this schedule runs a job at the beginning of every hour (e.g., 1:00, 2:00, 3:00, etc.).</p>
0 9 1 * *	<p>This schedule is more specific. It translates to "at minute 0 of hour 9 on the 1st of every month." It runs a job on the first day of each month at 9:00 AM.</p>

# Reference Manifest File

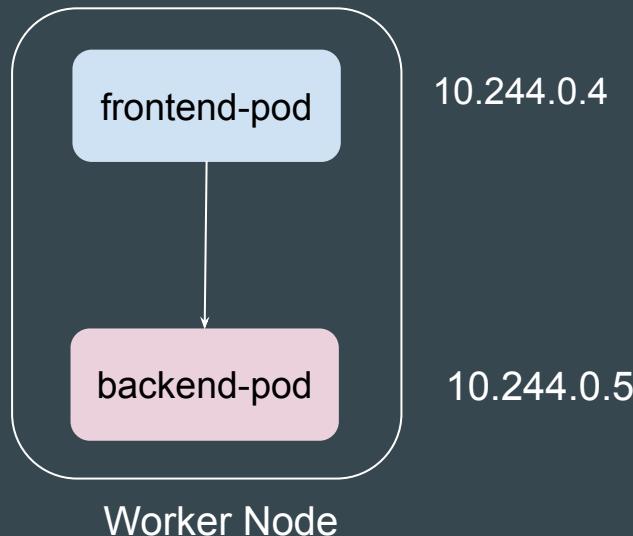
```
! cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: backup-cronjob
spec:
  schedule: "0 23 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: backup
              image: my-backup-tool:latest
          restartPolicy: OnFailure
```

# **Overview of Service**

# Setting the Base

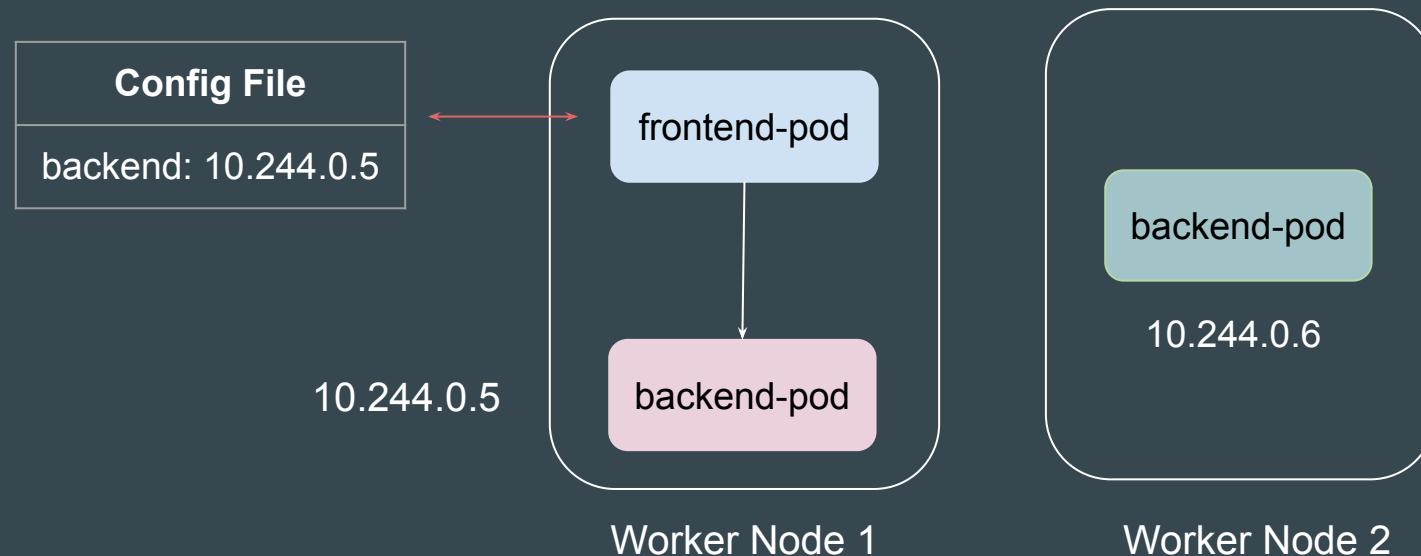
The Pods that get created in the Worker node have a private IP associated with them.

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



# Use-Case: Frontend to Backend

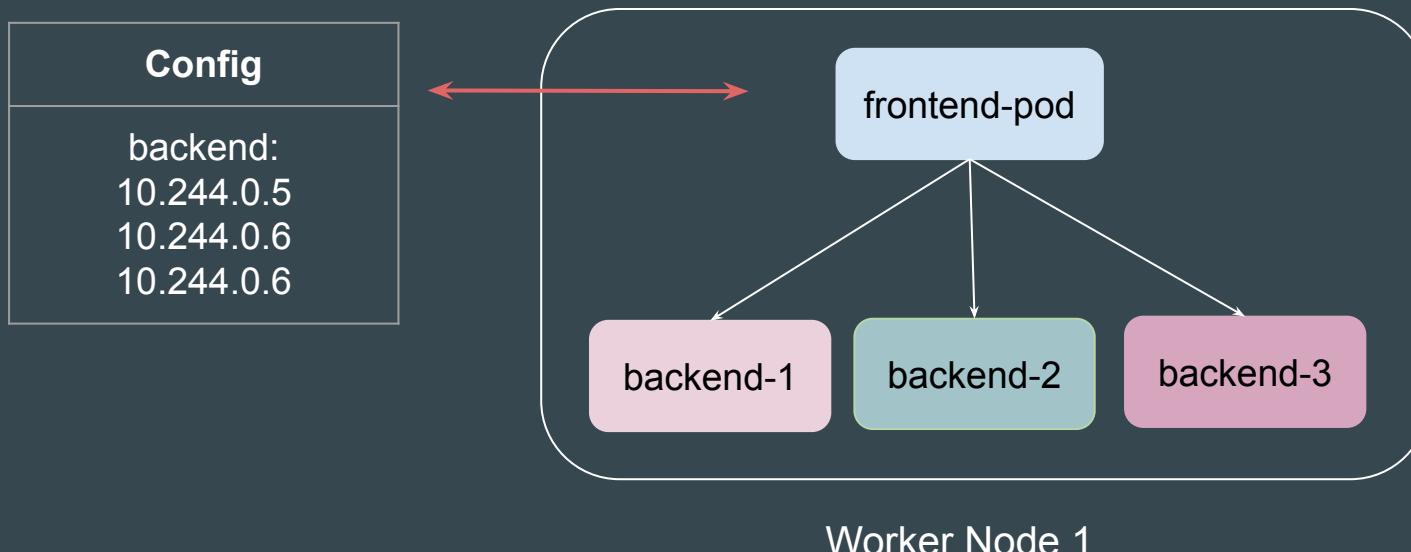
If the IP address of the backend pod is **hardcoded** in the front-end pod, it will create issues since Pods can be ephemeral.



# 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.

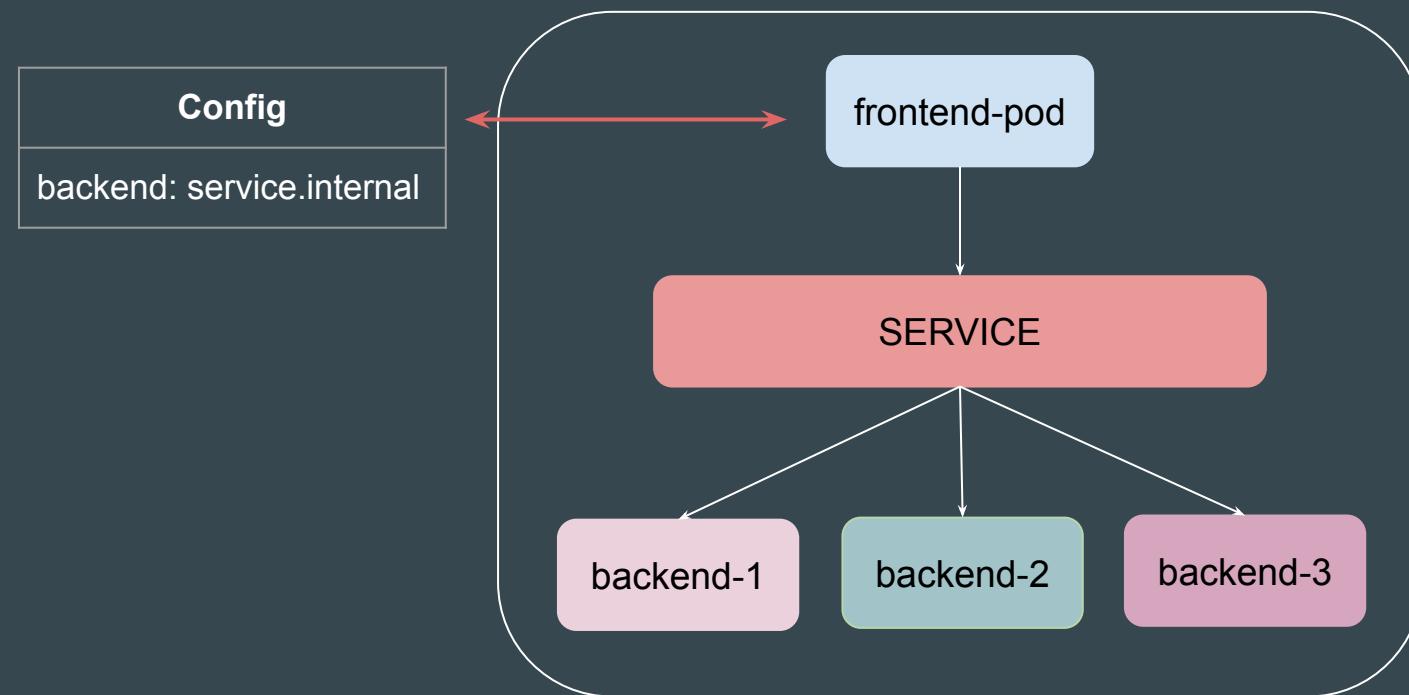


# Reference Screenshot - Distributing Traffic

```
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

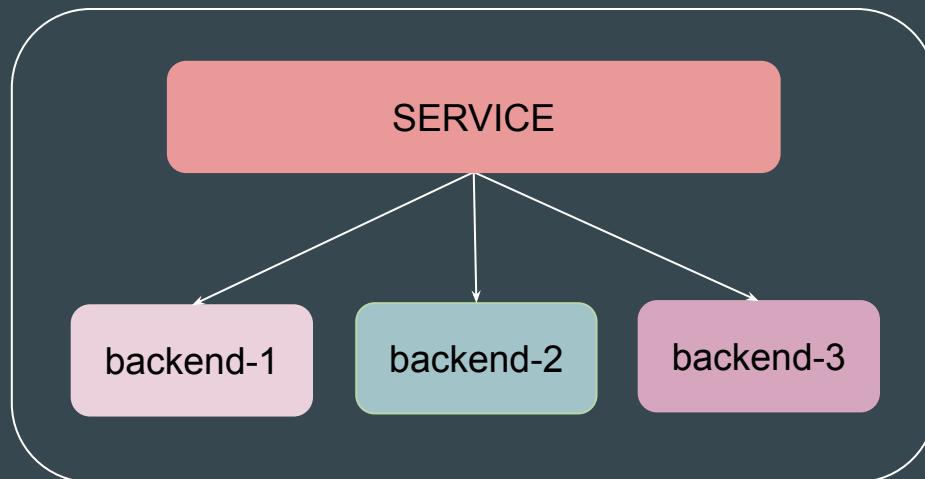
Service acts as a gateway that distributes incoming traffic between its endpoints.



# Service and Deployments

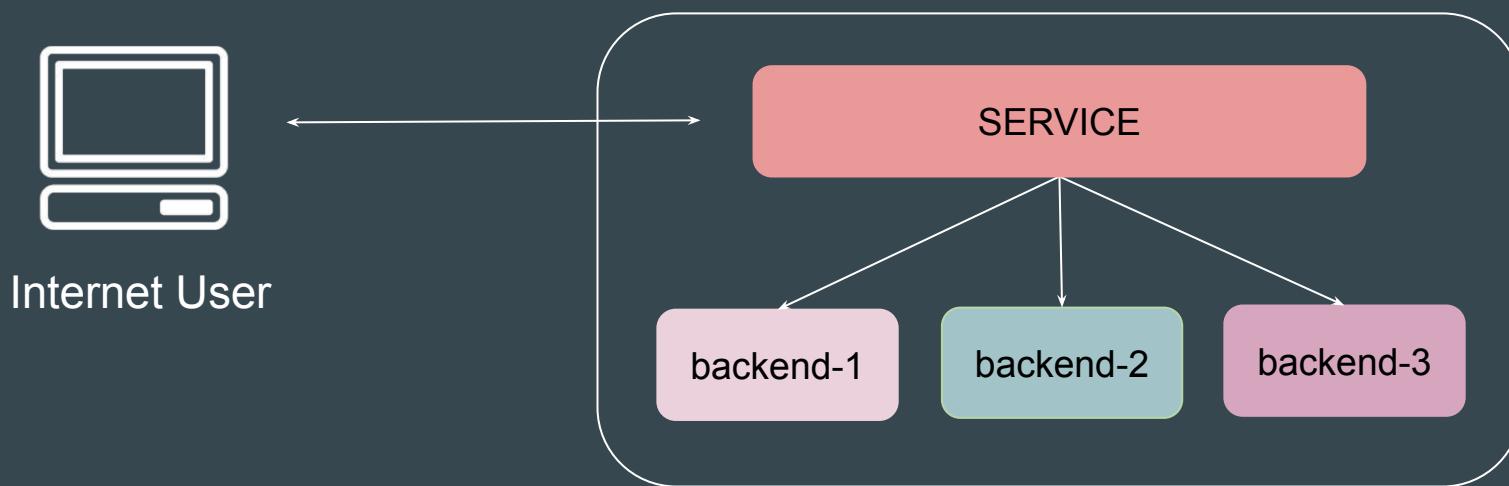
If you use a Deployment to run your app, that Deployment can create and destroy Pods dynamically.

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

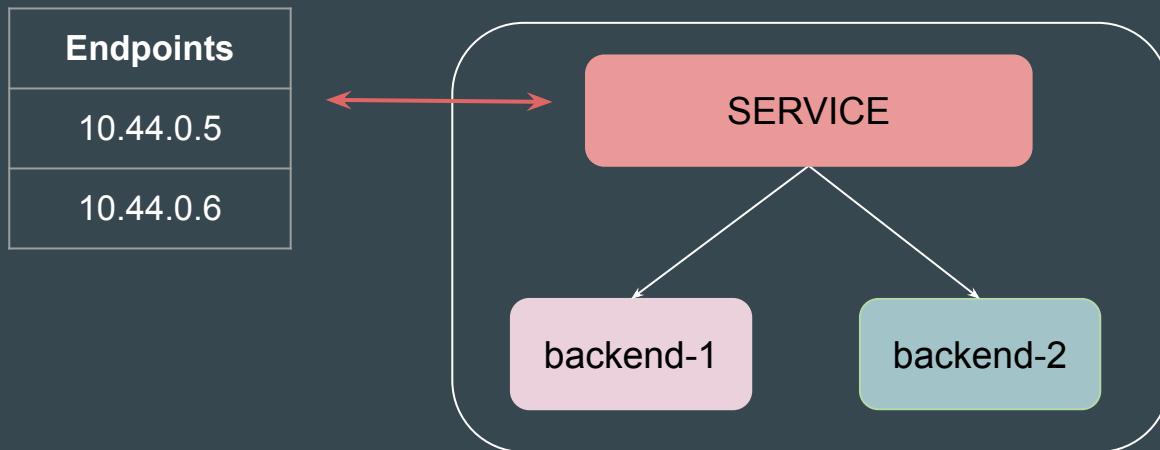
Service Type	Key Features	Use-Cases
ClusterIP	Default Service type. Accessible only within the cluster.	Internal microservices communication
NodePort	Exposes service on a static port (30000-32767) on each Node.	Development testing, demo applications
LoadBalancer	Exposes service externally using cloud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

# **Practical - Service and Endpoints**

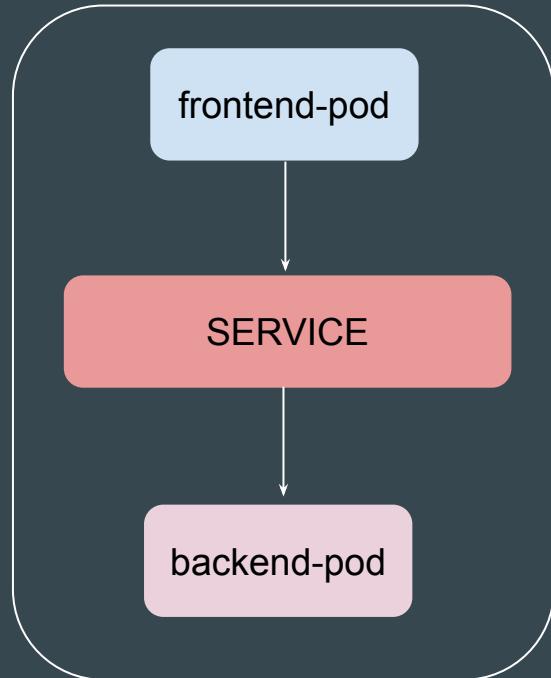
# Setting the Base

**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

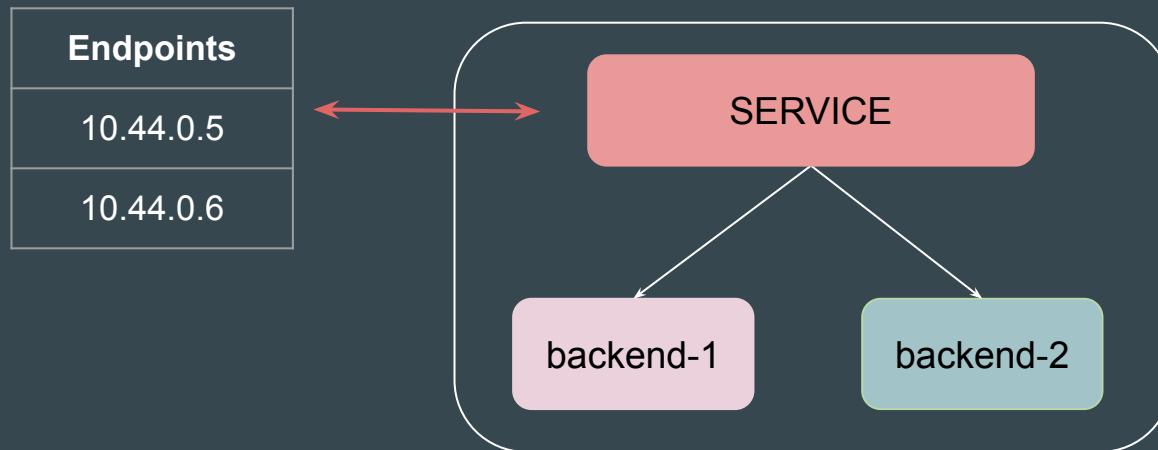
Add appropriate endpoint to the service manually for service to route traffic to.

```
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
```



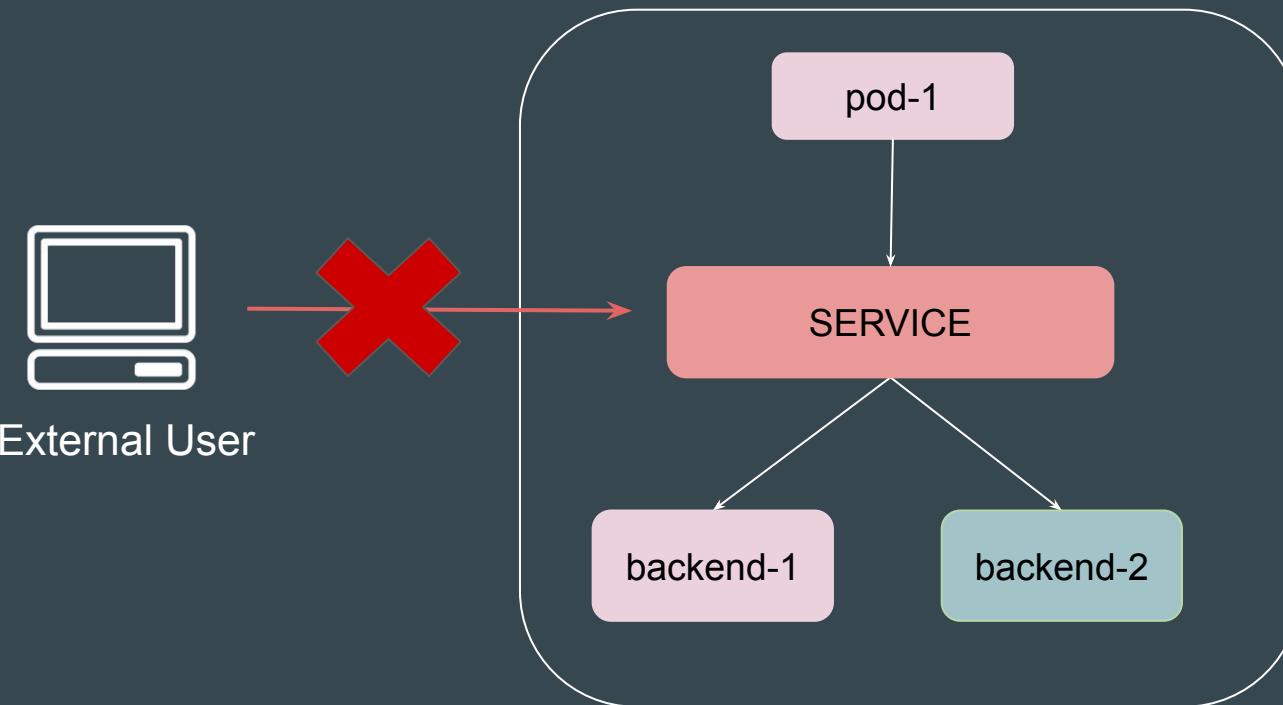
# **Service Type - ClusterIP**

# 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.

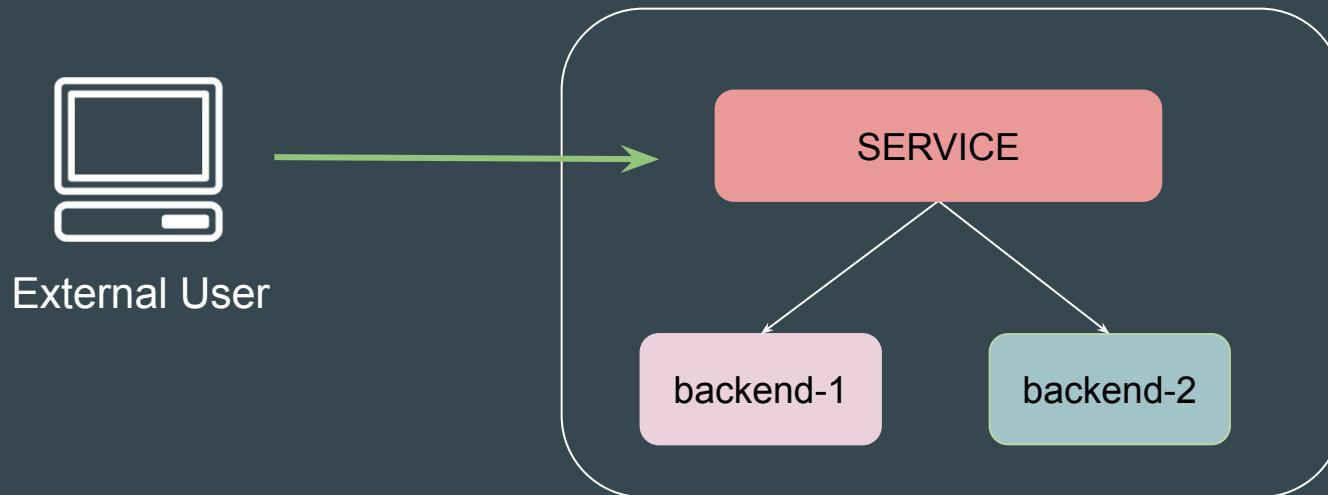
This IP can be used by other Pods inside the cluster to access the Service.

```
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

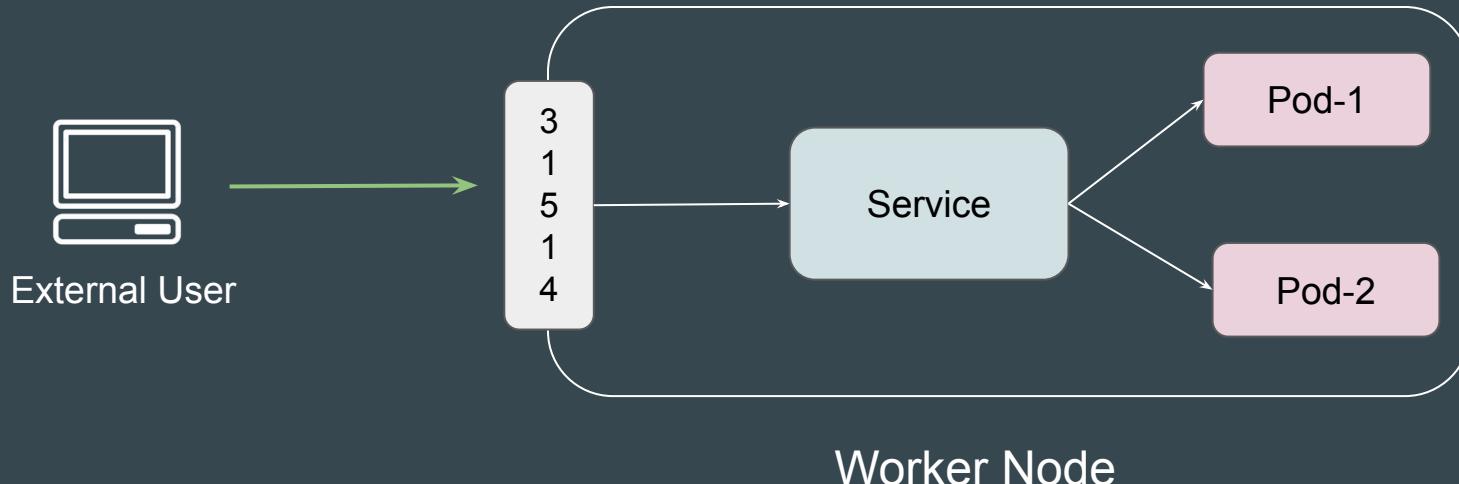
A **NodePort** is one of the service types used to **expose your application to the external world**.



# 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

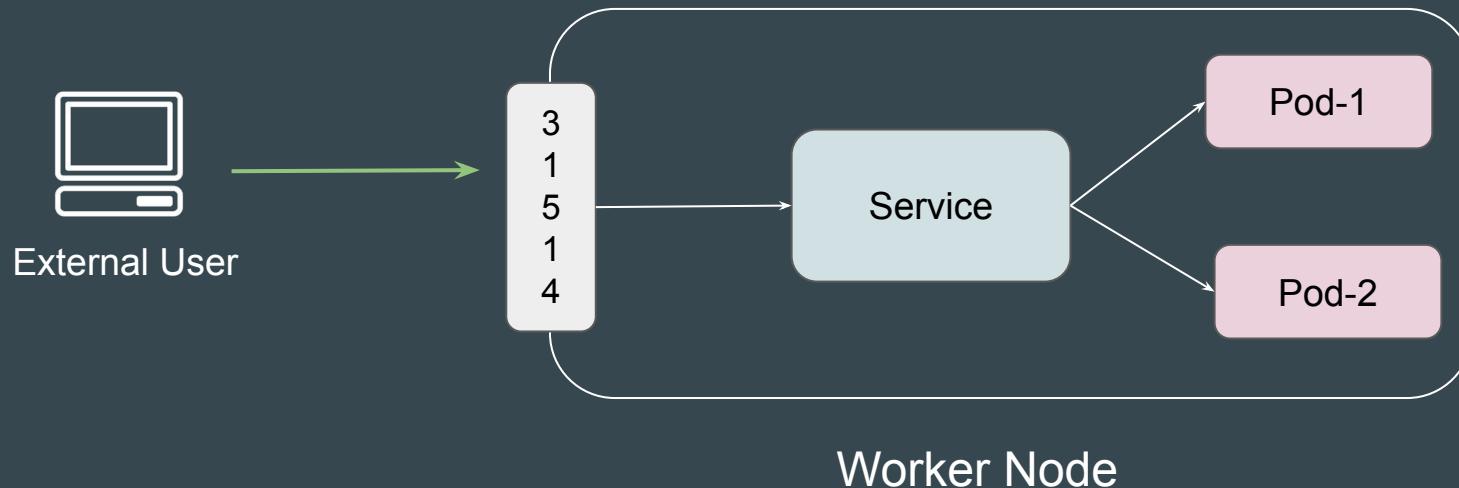
```
C:\>kubectl get service
```

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**

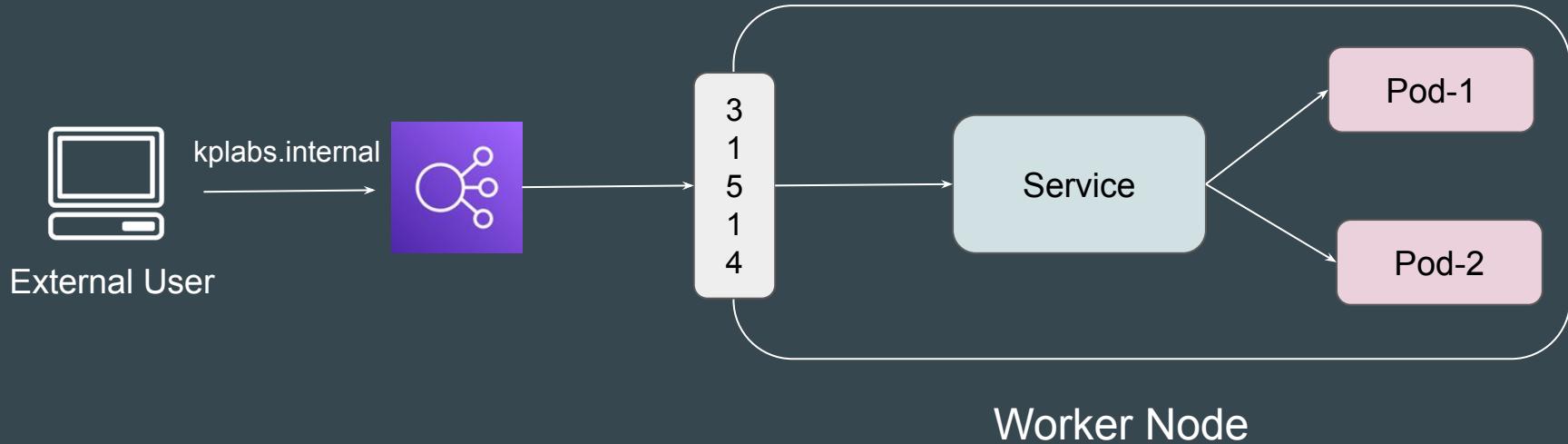
# Challenge with NodePort

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 / BLR11 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.

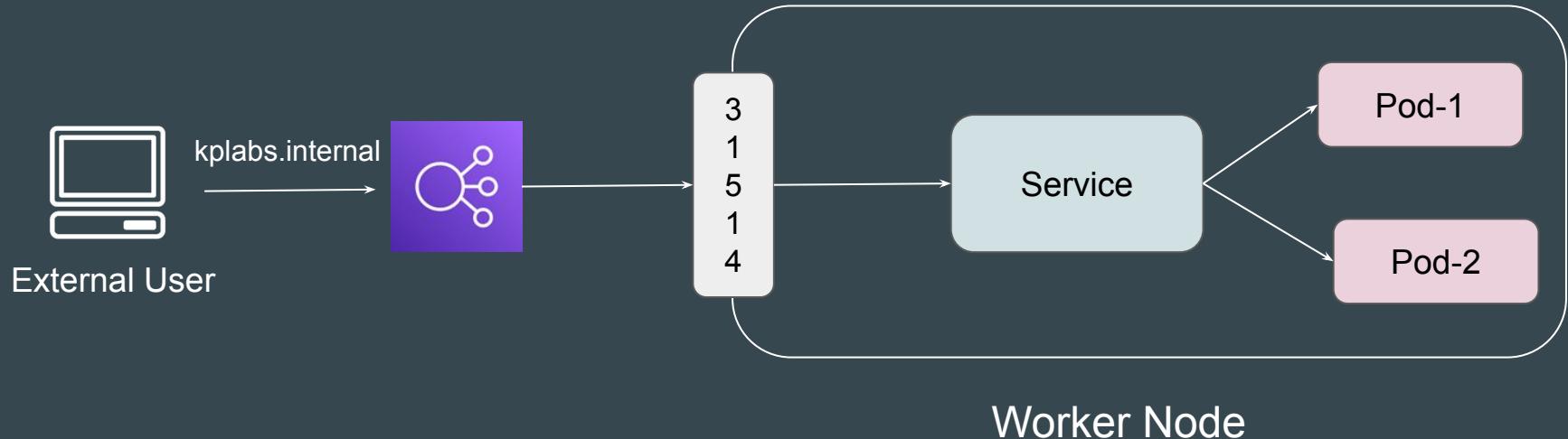
The main content area is titled **Settings**. It contains a note: **Load balancers provisioned by Kubernetes can only be modified through kubectl.** Below this note, it says: **Follow the accompanying how-to guides below to edit settings through kubectl. We also have more guidance around [configuring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

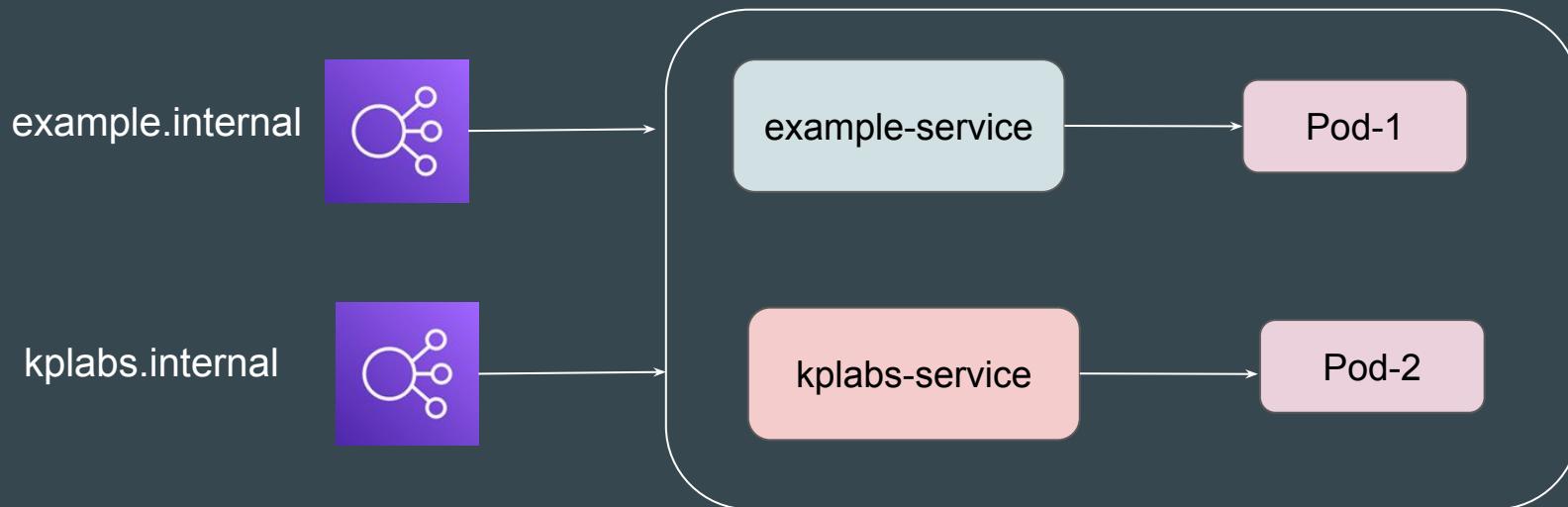
# Challenge with Basic Configuration

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



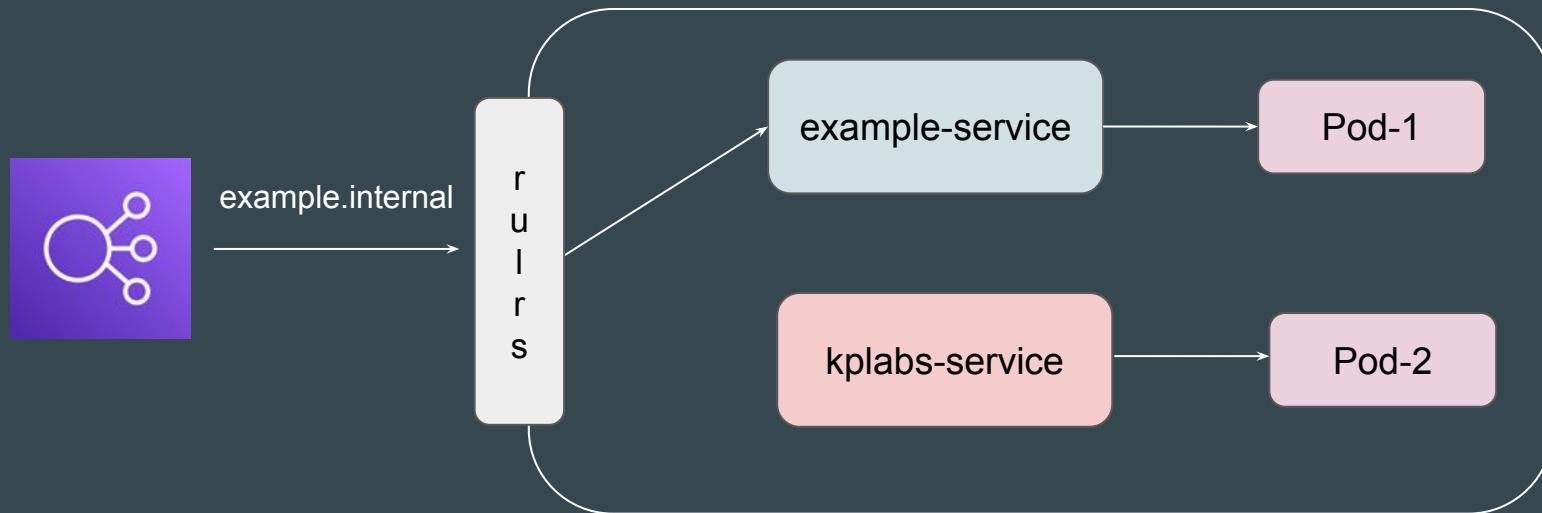
# Multiple Service Scenario

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.



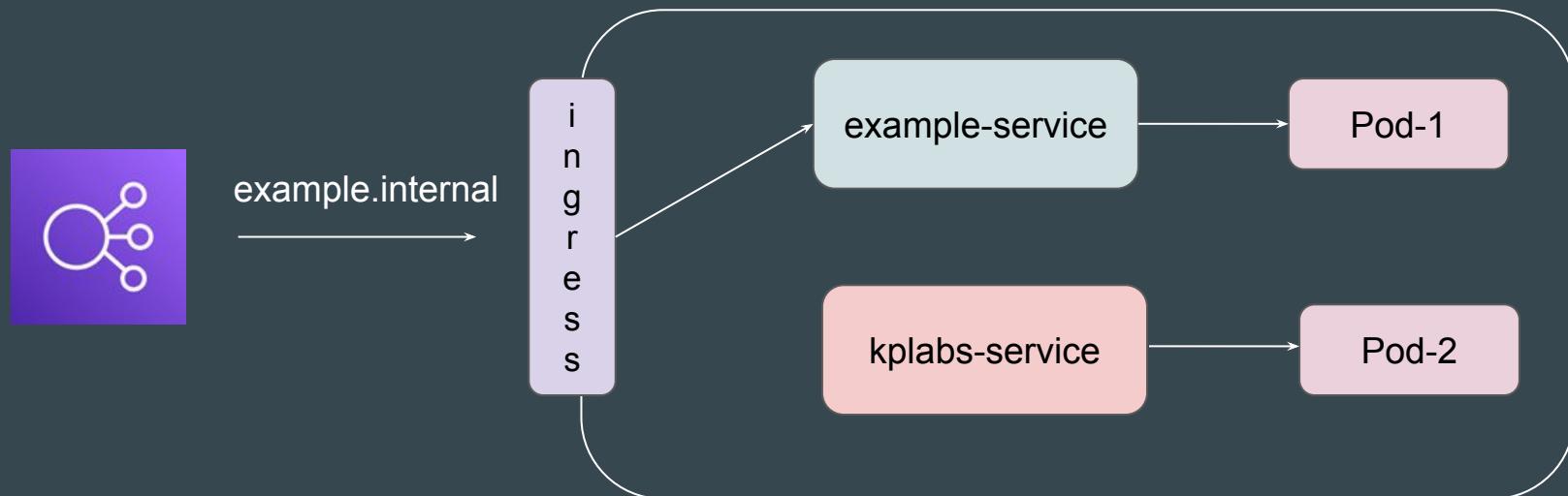
# Ideal Approach

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

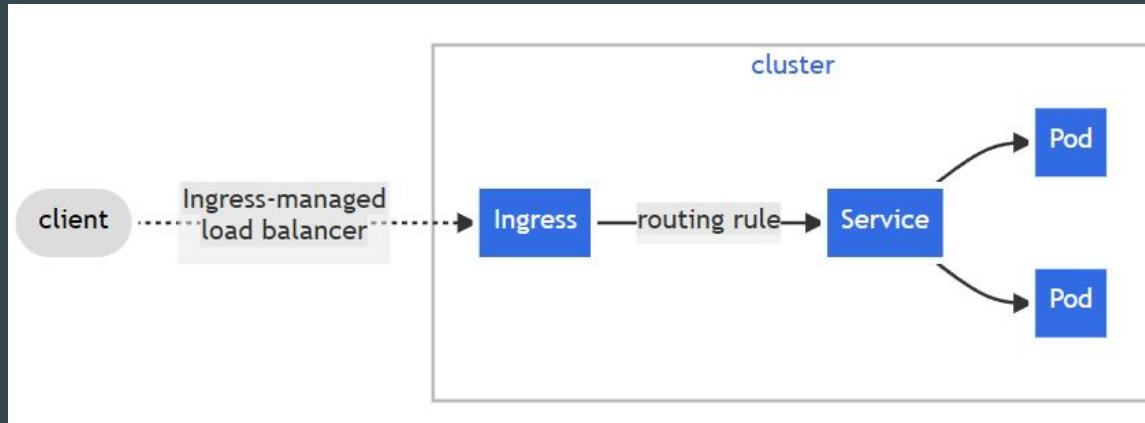


# Introducing Ingress

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



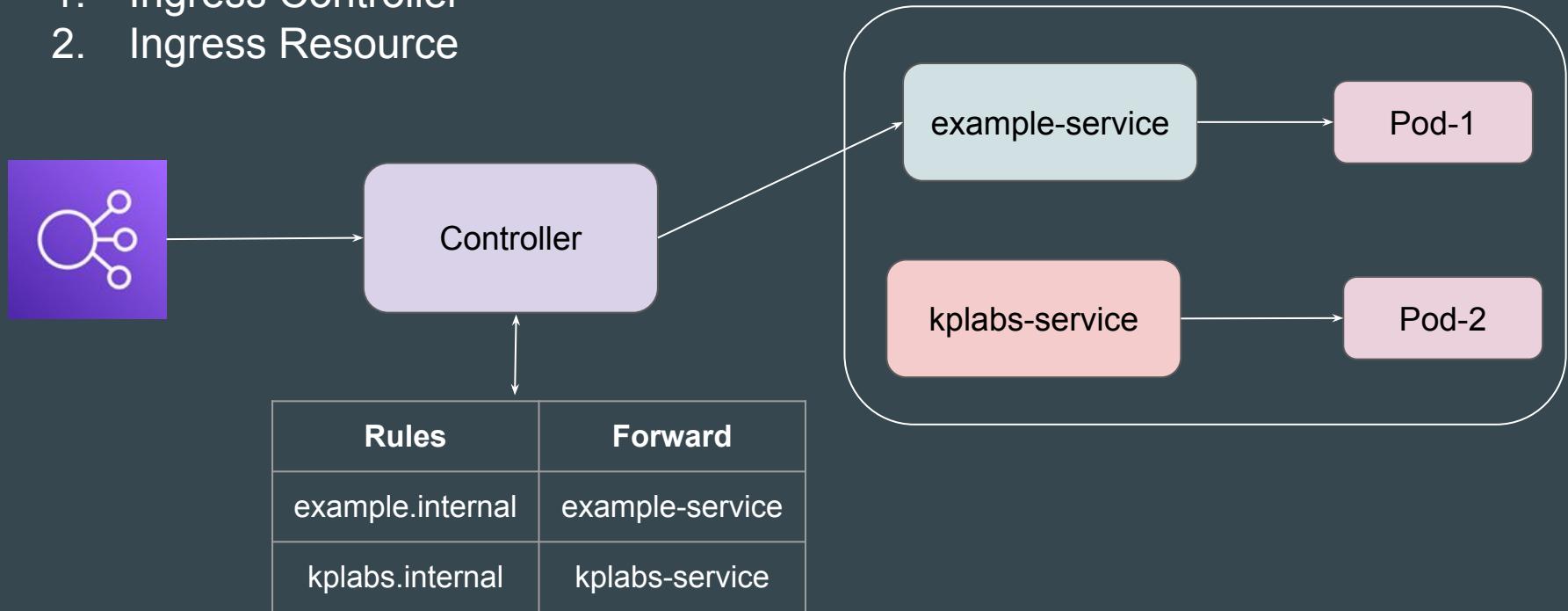
# Reference Diagram



# Components of Ingress

There are two sub-components of Ingress:

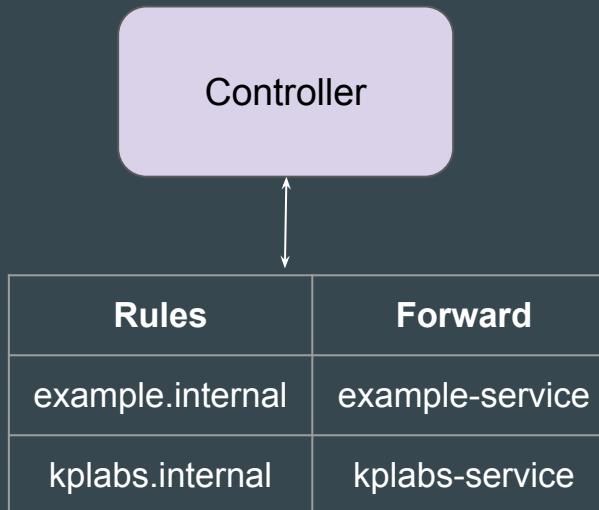
1. Ingress Controller
2. Ingress Resource



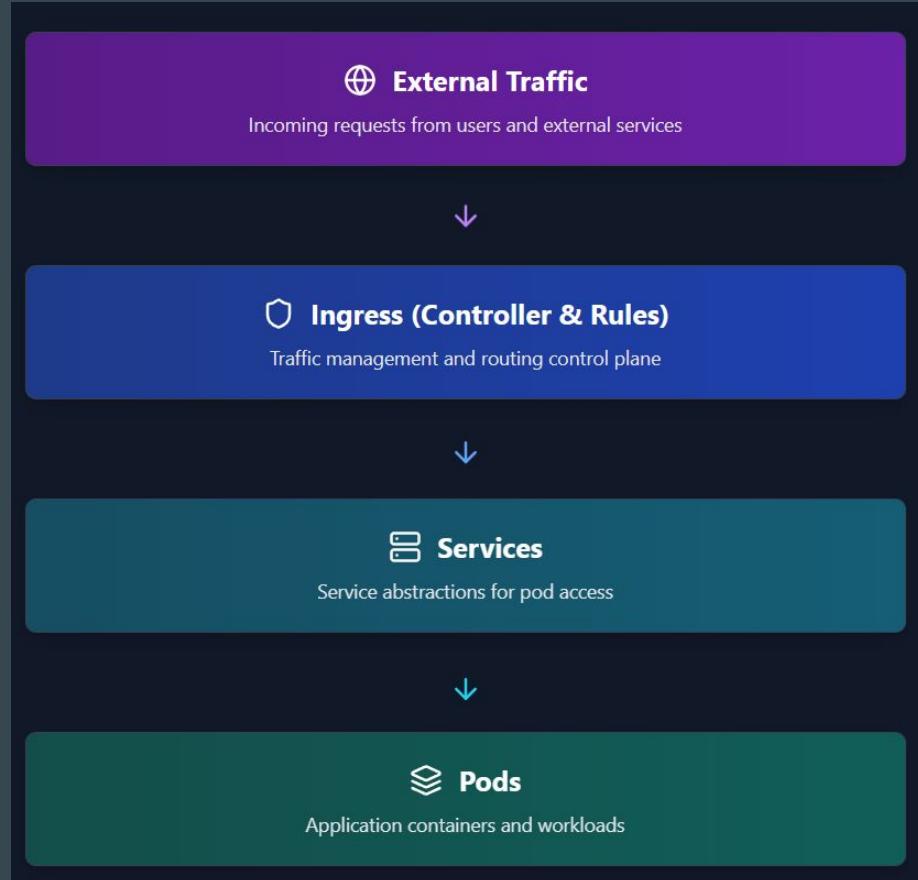
# Components of Ingress

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



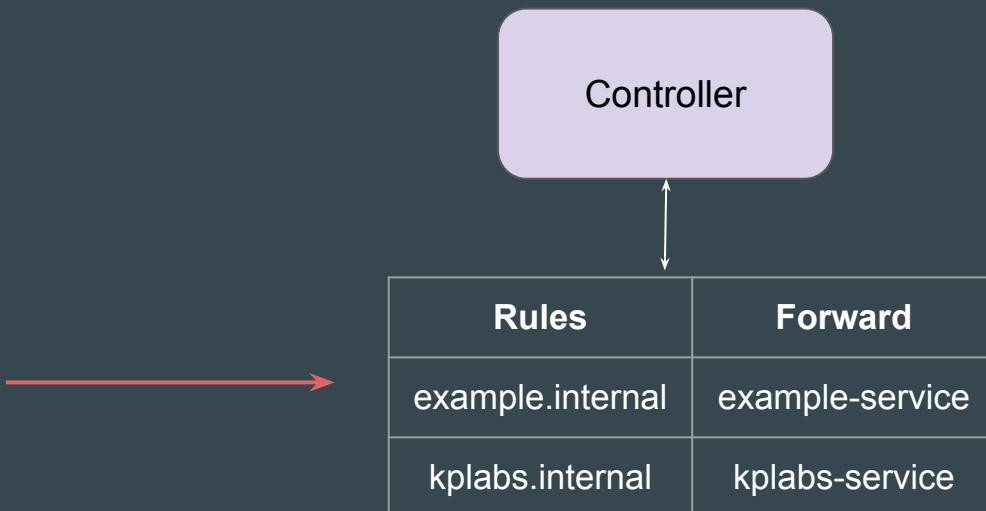
# Key Difference Summarized

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
```

# 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 banner below the header says 'An Ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.' followed by 'Get Started' and 'Documentation'. The main content area contains sections for 'Overview', 'Installations', 'Usage', 'Configuration', 'Performance Tuning', 'FAQ', and 'Contributing'. Each section has a brief description and a 'View file' link. The footer contains links to 'GitHub', 'Issues', 'Pull requests', 'Commits', 'Releases', and 'Wiki'.

Ingress NGINX Controller

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

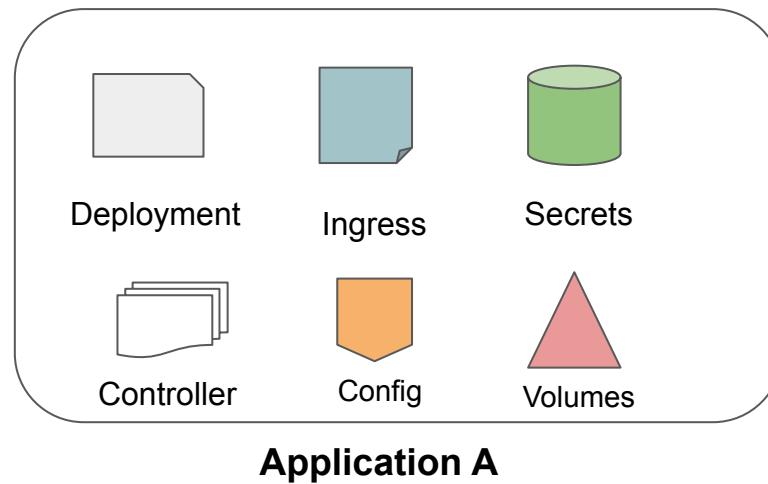
---

# Understanding Helm

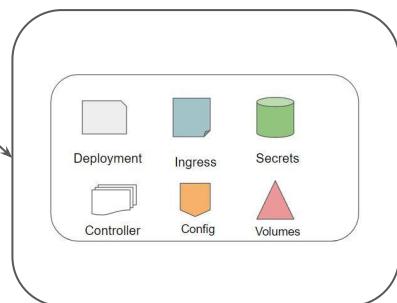
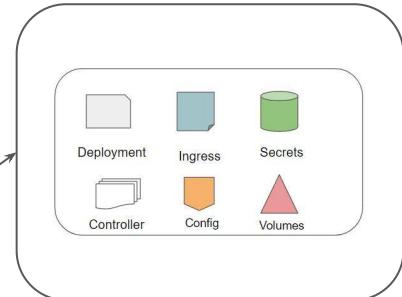
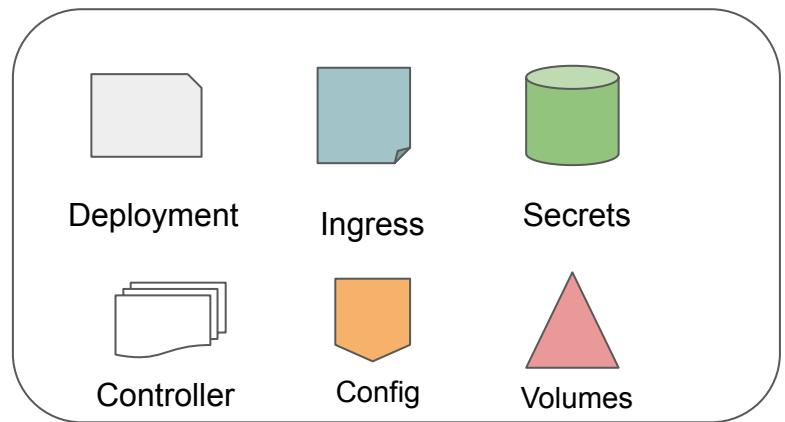
Helm is one of the package manager for Kubernetes.

Kubernetes application can contain lot of lot of objects like:

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

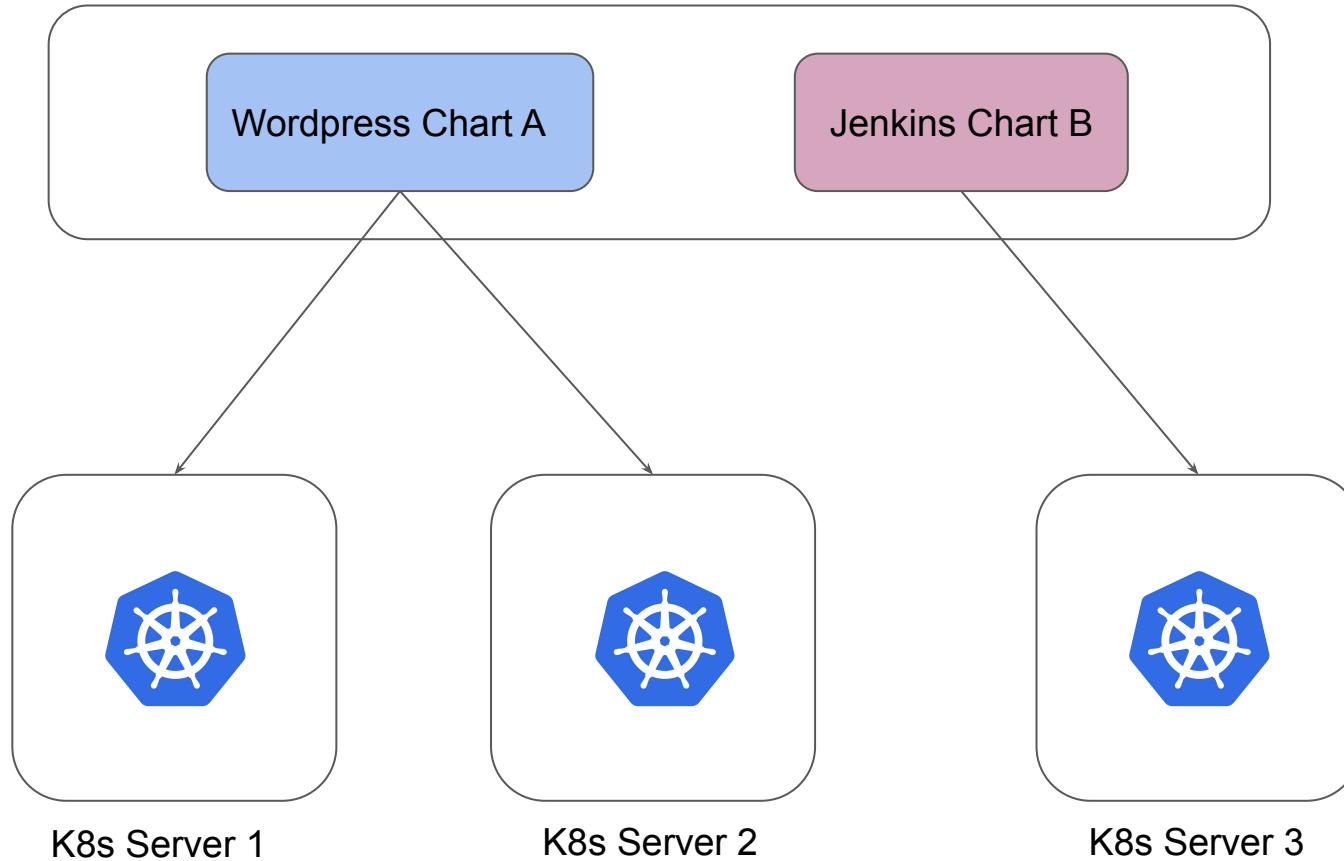


# Deploying Helm Charts



# 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

---

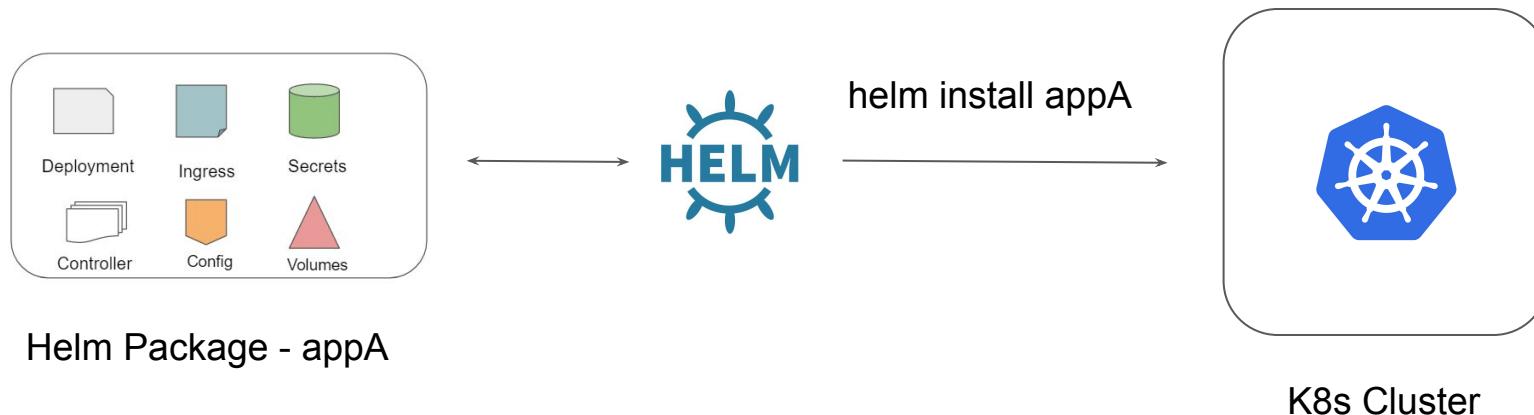
# Deploying Helm Charts

Let's Deploy Applications

---

# Installing Helm Chart

To install a new package, use the `helm install` command



# Helm Package Location

Helm packages can be placed in local disk or can also be stored centrally in public or private repository.

Depending on the location of the package, there is a slight change in the installation step.

**Public Repo**

Wordpress Chart A

Jenkins Chart B

`helm repo add public-repo URL`



`helm install wordpress`



K8s Cluster

# Important Notes

Every application packaged in Helm has its own set of requirements. Make sure to read the instructions carefully.

Install Helm Package only from trusted repositories.

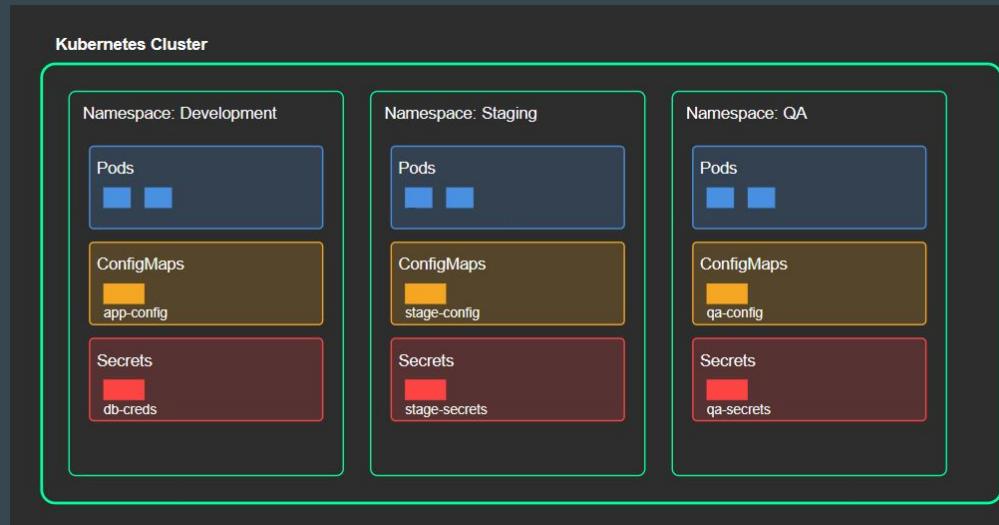
# Basic Helm Commands

<b>Commands</b>	<b>Description</b>
helm repo add	Adds a chart repository.
helm install <chart>	Installs Helm Package
helm uninstall <release>	Uninstalls the release.
helm repo list	List repositories
helm list	List the releases

# **Namespaces**

# Setting the Base

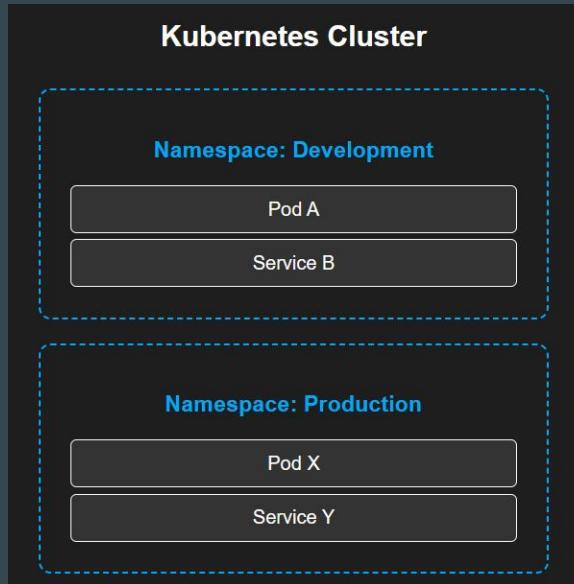
In Kubernetes, namespaces provide a mechanism for **isolating groups of resources** within a single cluster



# 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.

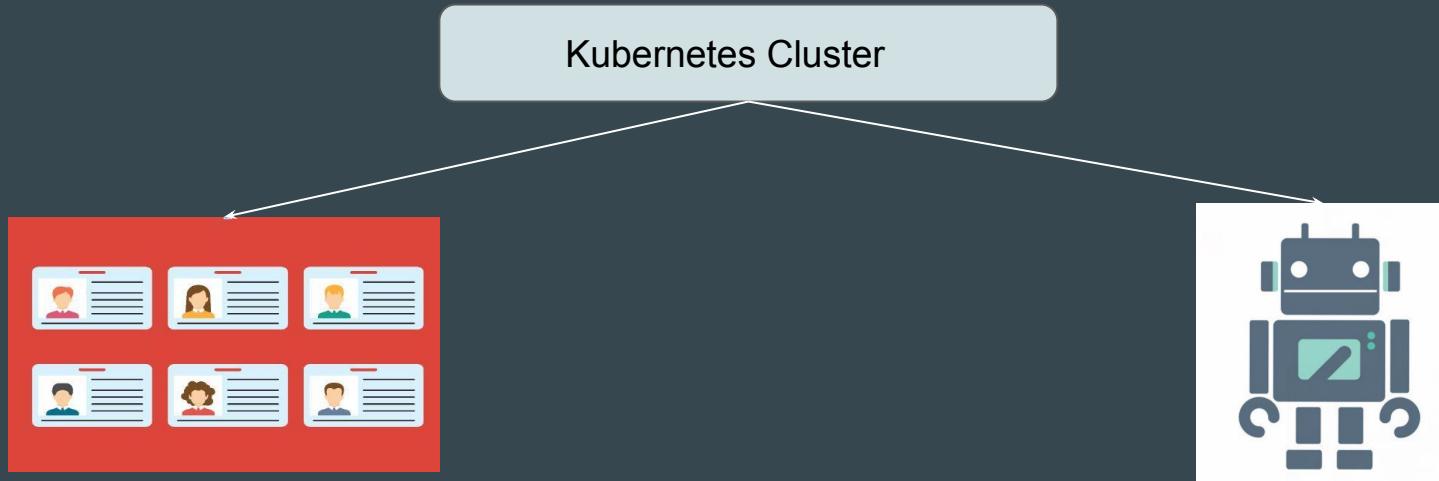
```
C:\>kubectl run test-pod --image=nginx -n my-namespace  
pod/test-pod created
```

# **Service Accounts**

# Understanding the basics

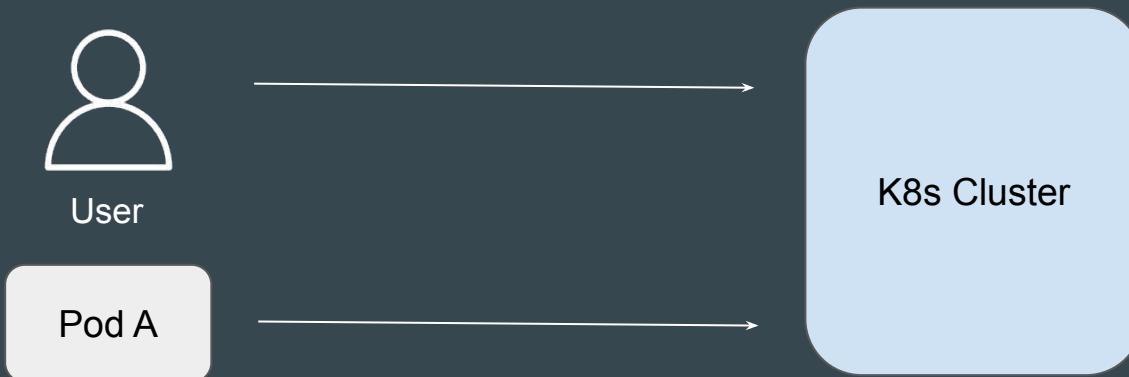
Kubernetes Clusters have two categories of accounts:

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



# Importance of Credentials

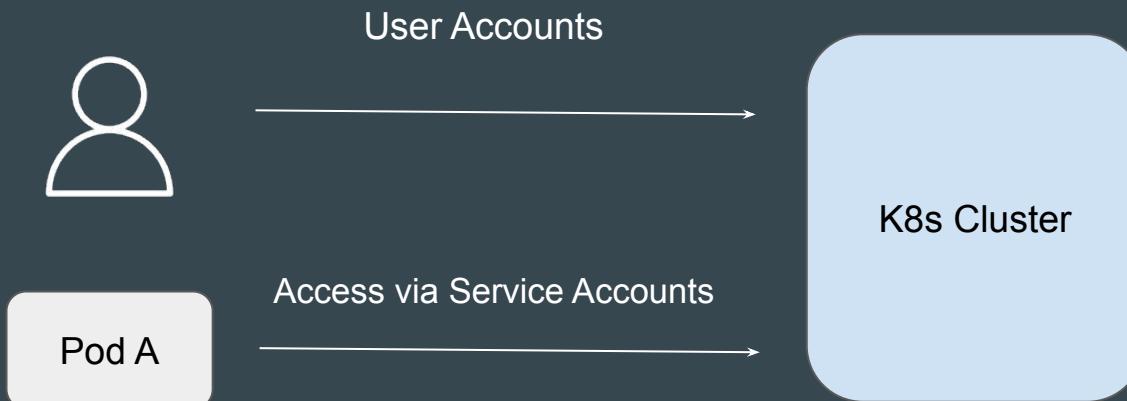
To connect to Kubernetes cluster, an entity needs to have some kind of authentication credentials.



# 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
eyJhbGciOiJSUzIiNlIsImtpZCI6IiJLZWlNejJxdTh3NlY2SjhRZHvkSEZfMWxKazFoZmttb3JLNnNTNzZTxMifQ.eyJhdWQiolsic3lzdGVtOmrvbm5lY3Rpdm
l0eS1zZXJ2ZXIiXSwiZXhwIjoxNzI3NzU50TY2LCJpYXQiOjE2OTYyMjM5NjYsImlzcyI6Imh0dHBz0i8va3ViZXJuZXRlcyc5kZWZhdWx0LnN2Yy5jbHVzdGVyLmx
vY2FsIiwiia3ViZXJuZXRlcyc5pbI6eyJuYW1lc3BhY2Ui0iJkZWZhdWx0IiwickG9kIjp7Im5hbWUiOjJuZ2lueCIsInVpZCI6ImU3ZjUyYWY4LWM5MGUtNDY5Zi1i
MTg3LTc3MjliMmNmE5MyJ9LCJzZXJ2aWN1YWNjb3VudCI6eyJuYW1lIjoIZGVmYXVsdcIsInVpZCI6ImZlOTUzMTQ0LThjNDYtNDF1My1iODFjLTmxMThkNTNhM
zAyOCJ9LCJ3YXJuYWZ0ZXIIoje2OTYyMjc1NzN9LCJuYmYiOjE2OTYyMjM5NjYsInN1Yi6In5c3R1bTpzZXJ2aWN1YWNjb3VudDpkZWZhdWx00mR1ZmF1bHQifQ
.i5jS2uEbNgj_1GA6LQh2YxQKEJsNjZbvoJh-Qjf-vH_f1Ow0KTvhmLPCL1UxCLfoI1NejT07Agckj3SFXYjEqmlvcbEYpLXt8eUjrHC8s6Ra105XGnpbt5uUy3GU
BYP4HnRE97OzzU3HyU2gM14Kd8d8a9iiHb7yov82ph6moOPuuxCxBowNo5edWMWnNDWLKLNOFX168oxAmQo8ZQI5k37INIE1oN5N2bzp7Y40Gv3CURK1X904B0YuT
UN8gSYZsHaQaVq8FT-Q_xGfGQbvjqUzi0rRaKNc9QJ0BiIE3CKQN7PL6C0Lxyt_9LieJV438xPgwer2V_aNeHBWgU99oAroot@nginx:/#
```

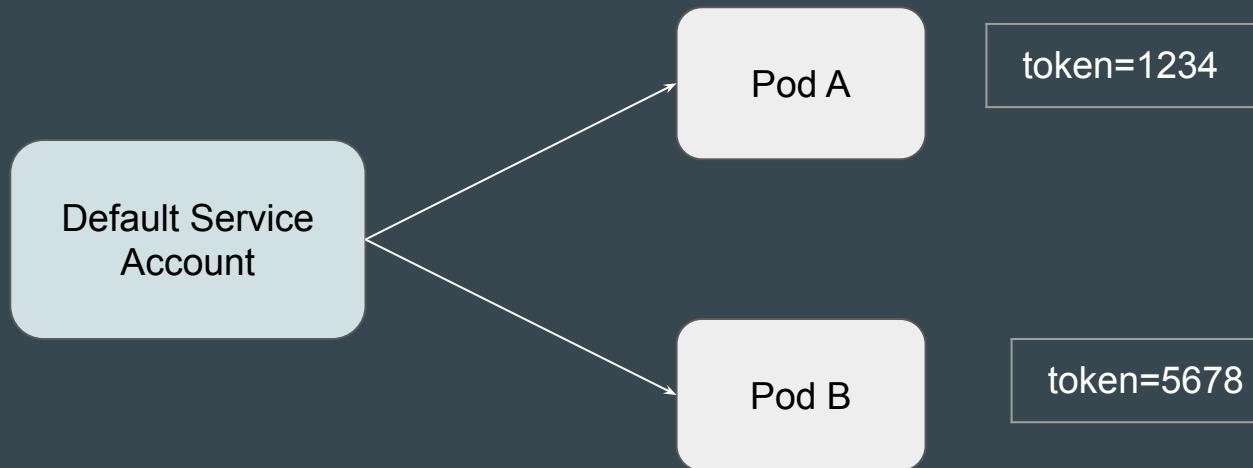
# Connecting to K8s using Token

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"  
      ]  
    },  
  ],  
  "nonResourceURLs": []  
}
```

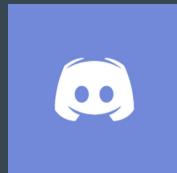
## Points to Note

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

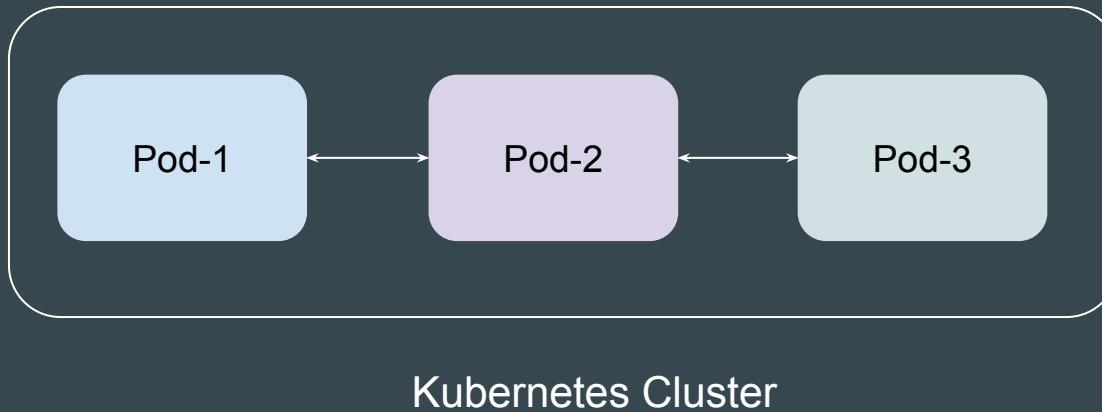
You can use `serviceAccountName` to specify custom service account token as part of the Pod.

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

# **Overview of Network Policies**

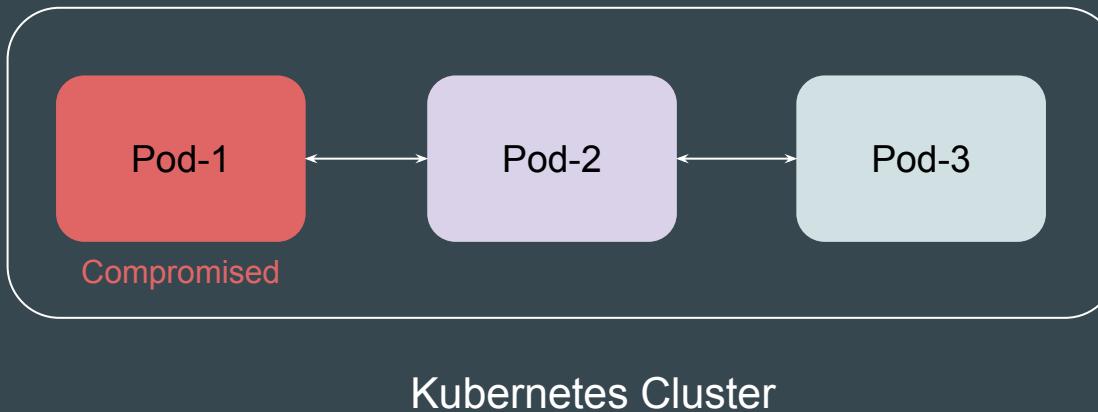
# Understanding the Basics

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.



# Ideal Scenario

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



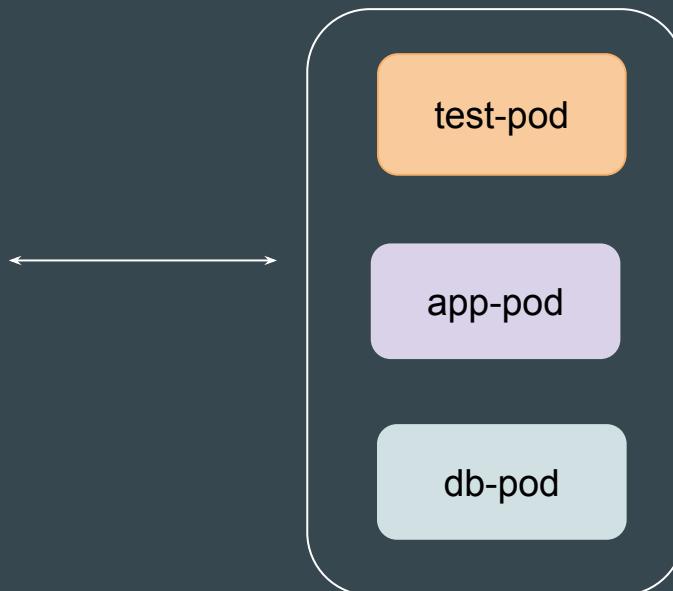
Kubernetes Cluster

# 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

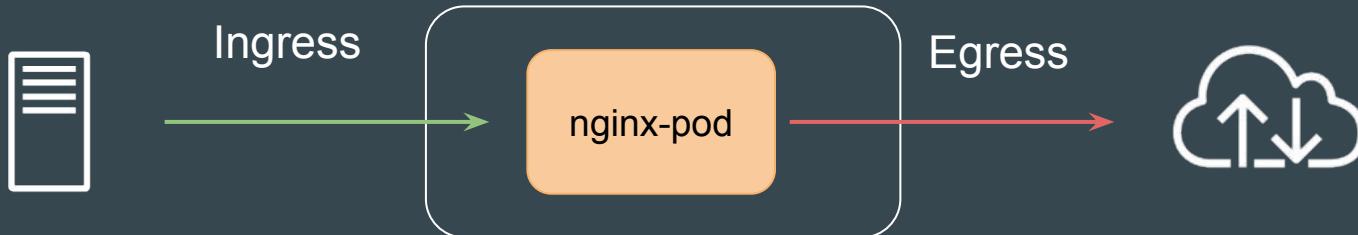
Network Policies



# Types of Rules

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

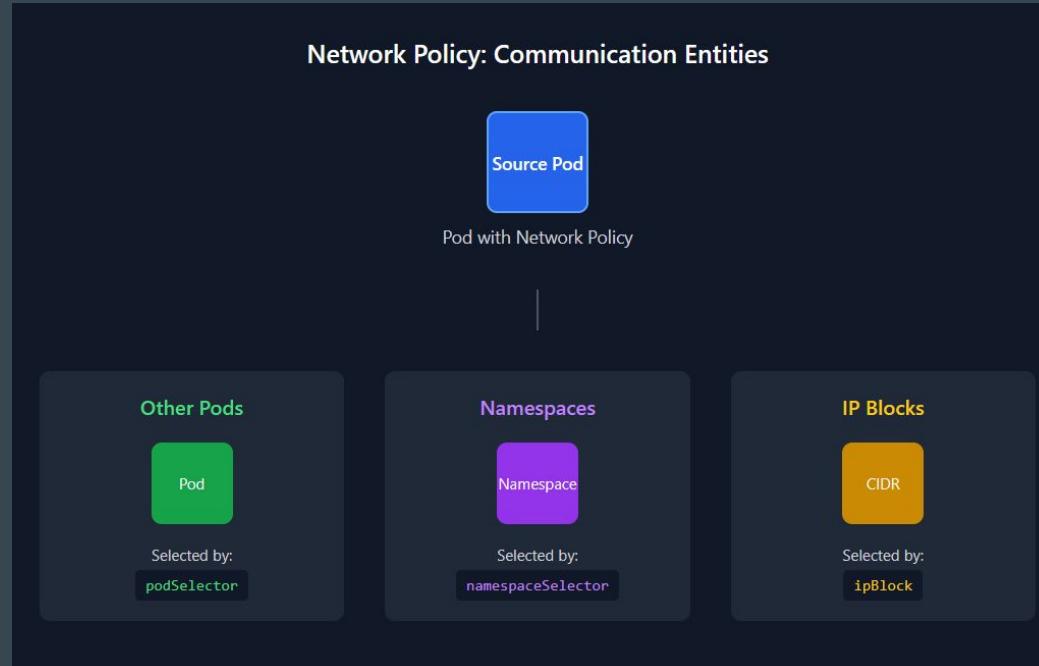
1. Ingress Rules (Inbound Rule)
2. Egress Rules (Outbound Rule)



# Supported Filtering Entities

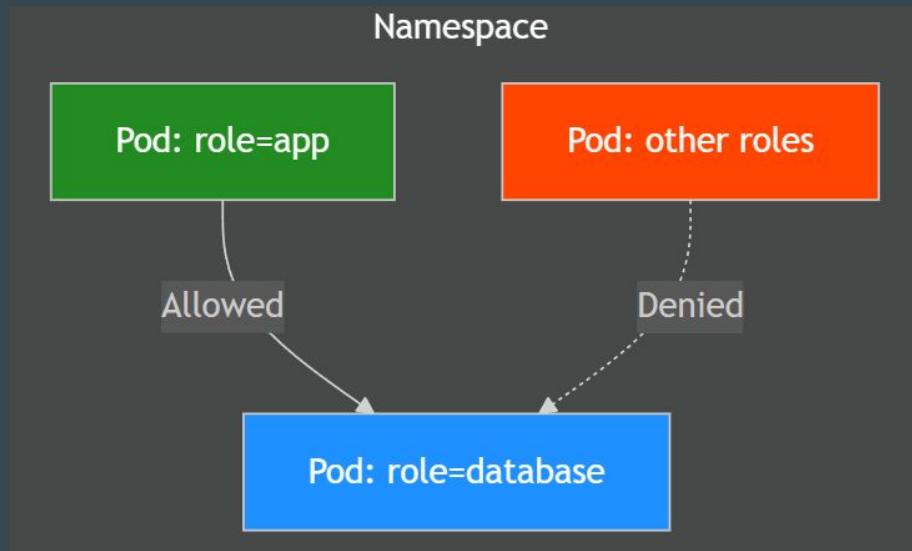
The entities that a Pod can communicate with are identified through a combination of the following three identifiers:

1. Other pods
2. Namespaces
3. IP Blocks



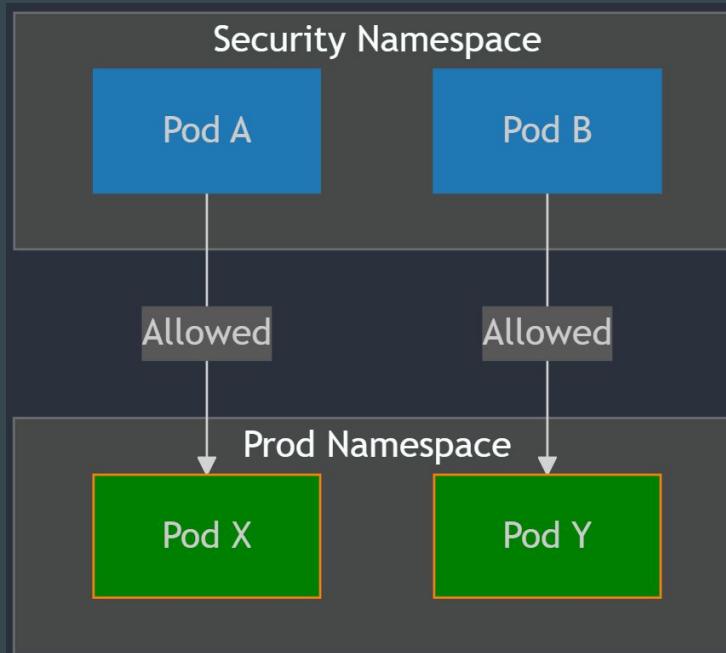
# Example 1 - Pod Selector

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



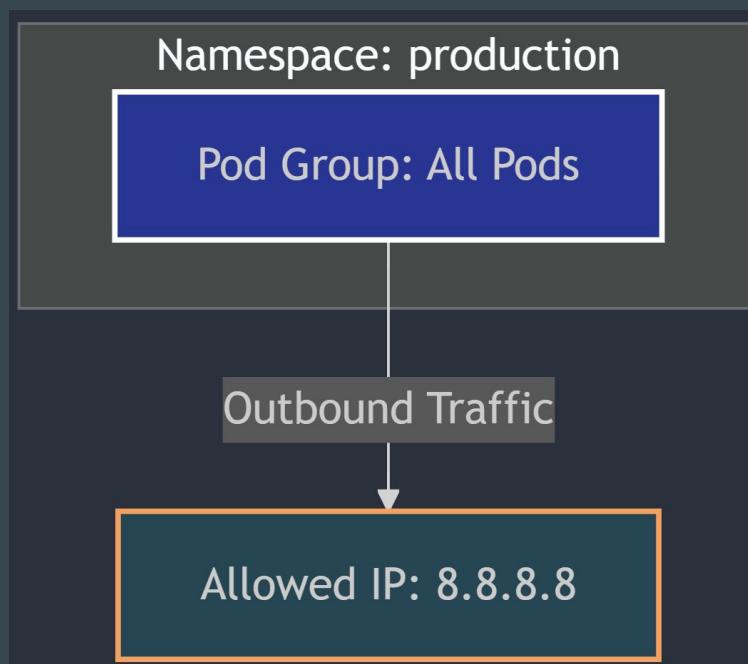
## Example 2 - NameSpace Selector

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

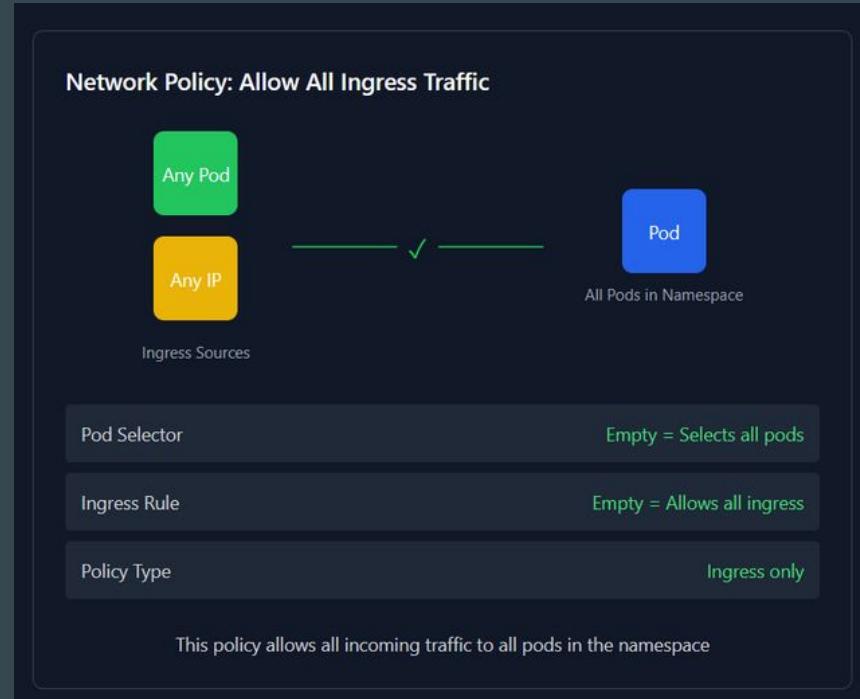
podSelector { }

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:

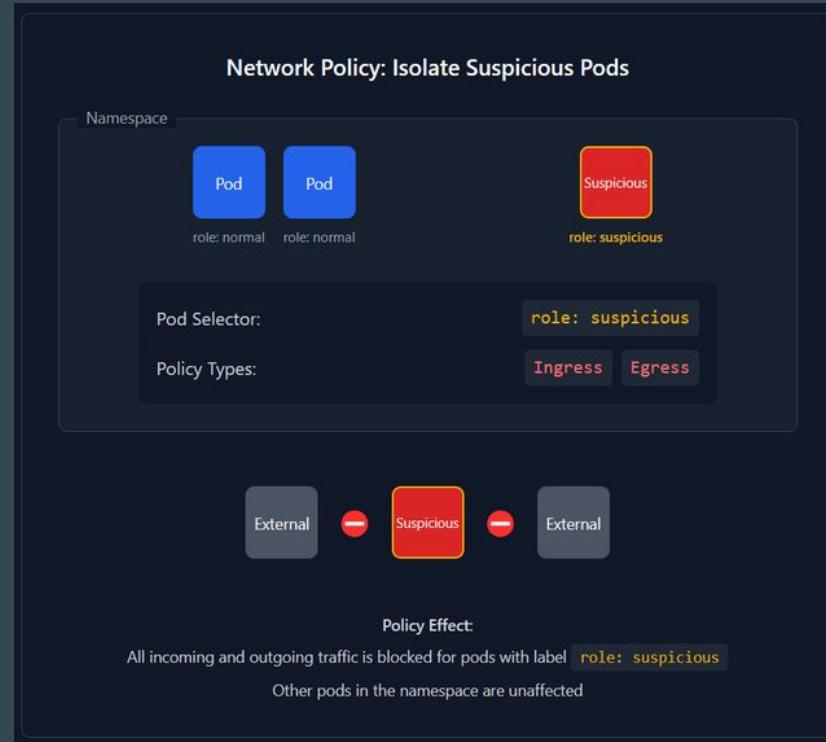
- { }

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

# Example 3 - Isolate Suspicious Pod

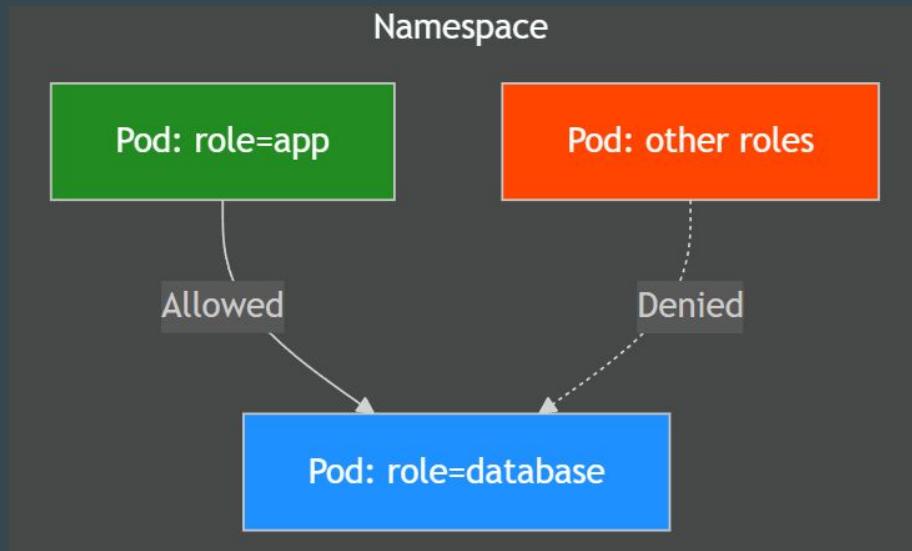
This policy blocks all ingress and egress access to pod with role=suspicious.

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



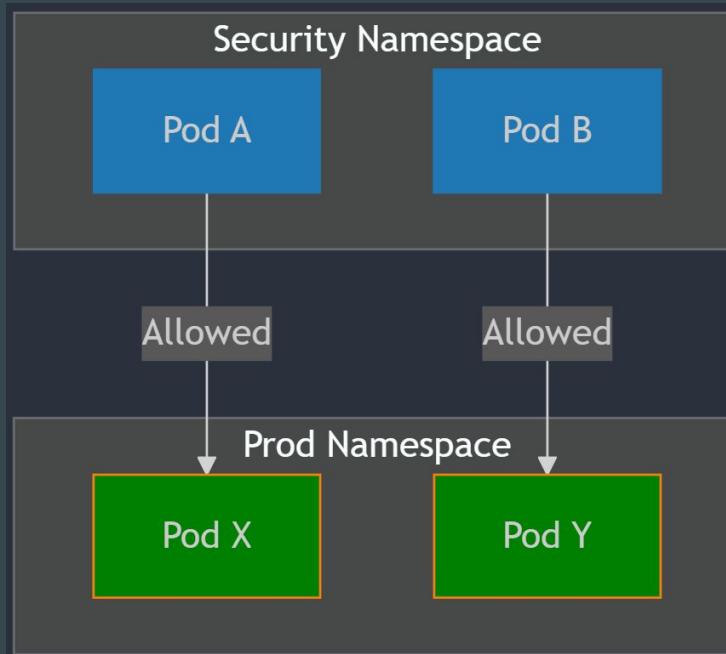
## Example 4 - PodSelector

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.

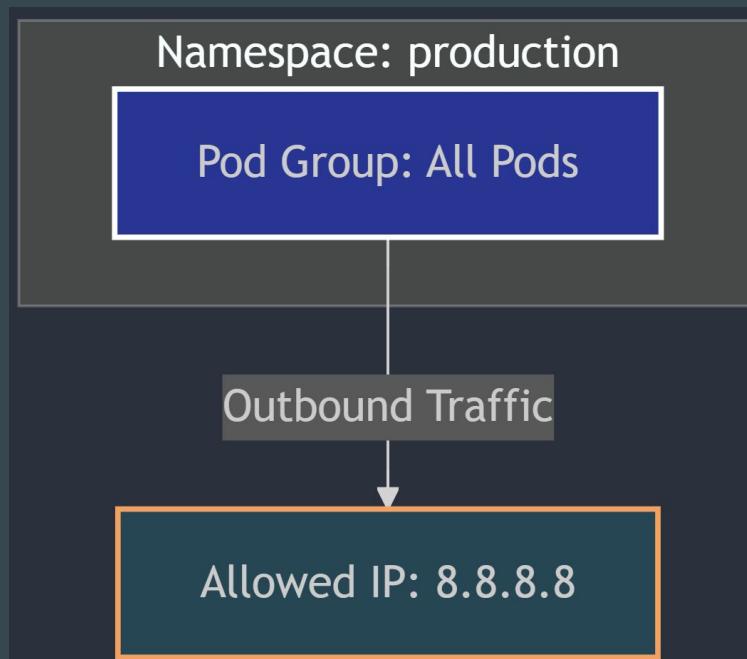


## Example 5 - Reference Code

```
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.



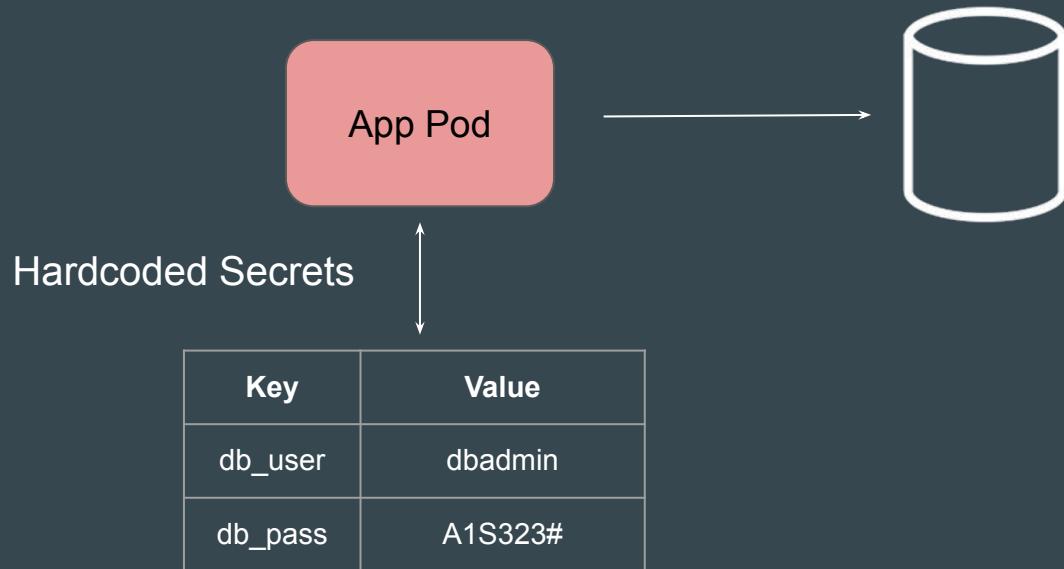
## Example 6 - Reference Code

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

# **Kubernetes Secrets**

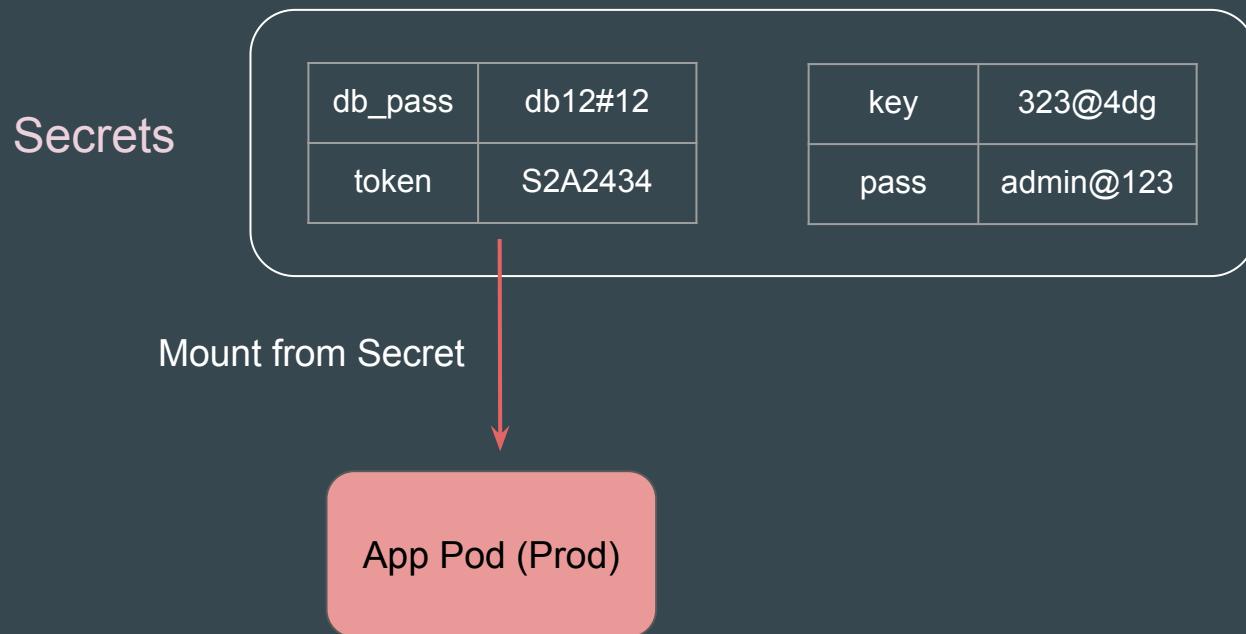
# HardCoding Secrets Should be Avoided

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

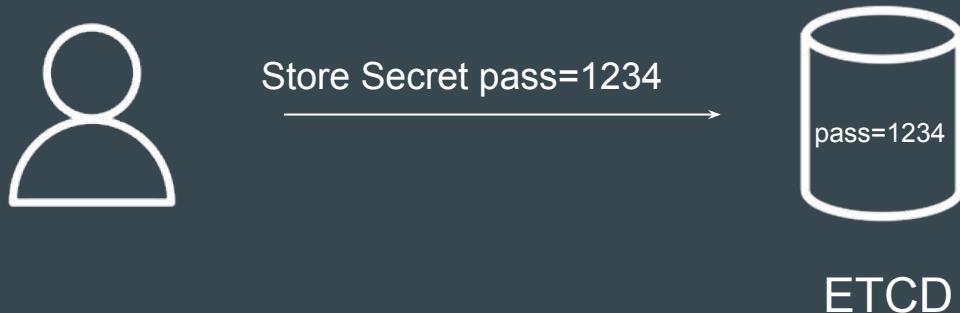
```
C:\>kubectl get secret
NAME          TYPE        DATA   AGE
my-secret     Opaque      1      22m
```

```
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
```

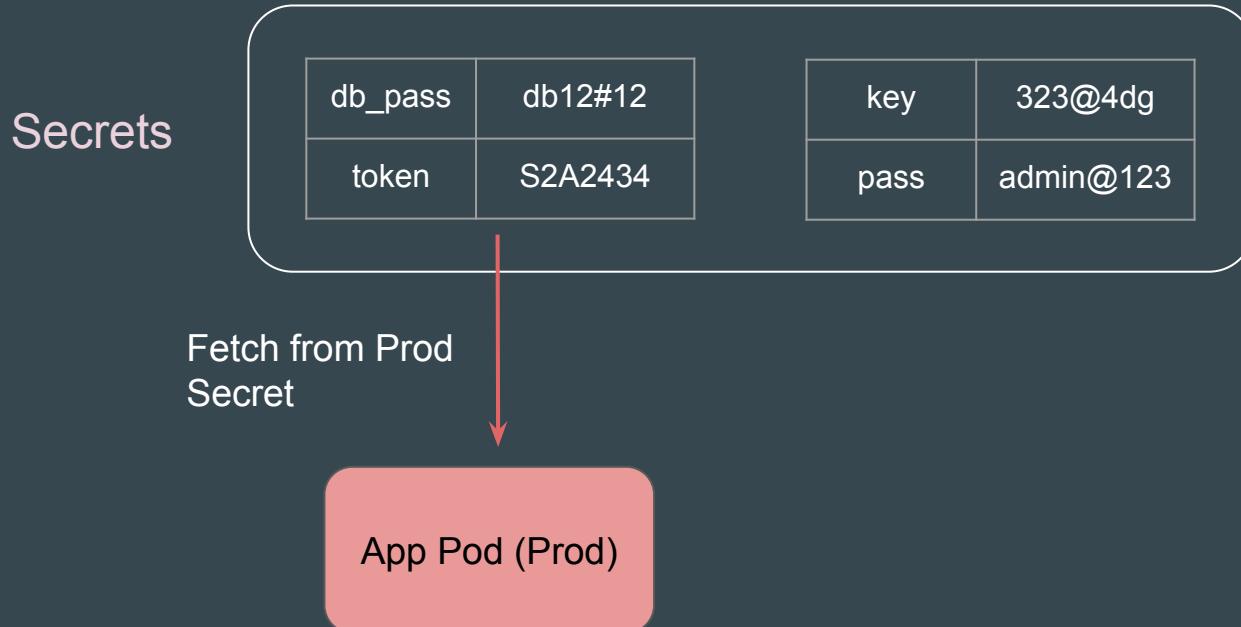
base64 encoded

# **Kubernetes Secrets - Practical**

# Two Parts of this Practical

First Part: Create Kubernetes Secret

Second Part: Mount the Secret inside the Pod.



# Part 1 - Create Secret

Use the `kubectl create secret` command to create secret in Kubernetes.

Examples:

```
# Create a new secret named my-secret with keys for each file in folder bar
kubectl create secret generic my-secret --from-file=path/to/bar

# Create a new secret named my-secret with specified keys instead of names on disk
kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa
--from-file=ssh-publickey=path/to/id_rsa.pub

# Create a new secret named my-secret with key1=supersecret and key2=topsecret
kubectl create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

# Create a new secret named my-secret using a combination of a file and a literal
kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-literal=passphrase=topsecret

# Create a new secret named my-secret from env files
kubectl create secret generic my-secret --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

# Different Approaches for Reference

A Pod can reference the Secret in a variety of ways, such as in a volume mount or as an environment variable.



# Part 1 - Mount Secret Inside Pod (Volume)

Using Volume Mounts, you can mount a specific secret inside a Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
    - name: secretmount
      image: nginx
      volumeMounts:
        - name: secret-volume
          mountPath: "/etc/secrets"
          readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: firstsecret
```

## Part 2 - Mount Secret Inside Pod (Env)

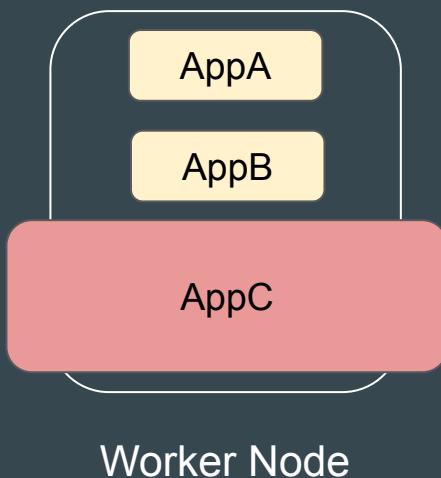
In this method, the values in the Secrets are exposed as **environment variables** to the container.

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env
spec:
  containers:
  - name: secret-env
    image: nginx
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: firstsecret
          key: dbpass
```

# **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

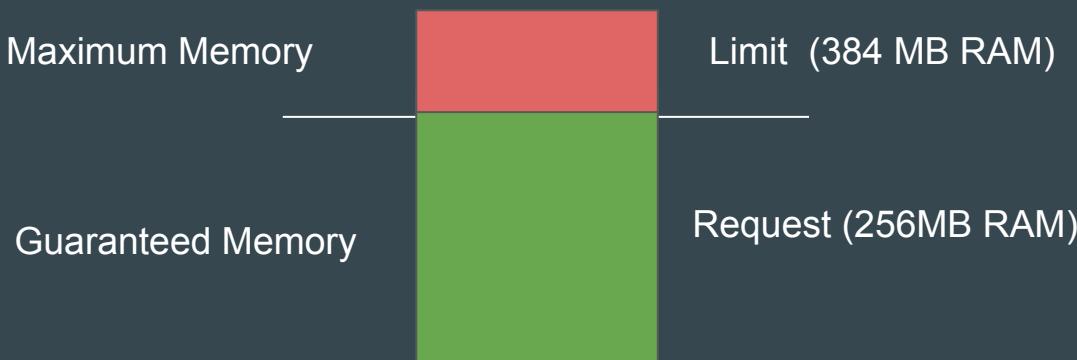
A Pod requires a certain amount of memory and computing resources to run properly.

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).

---

# Liveness Probe

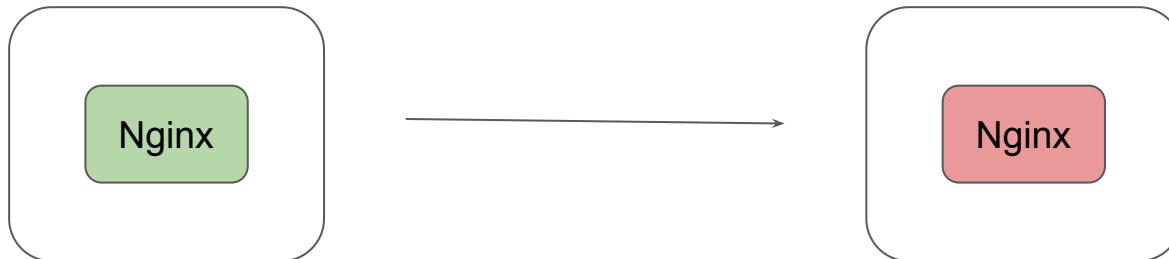
## Health Checks

---

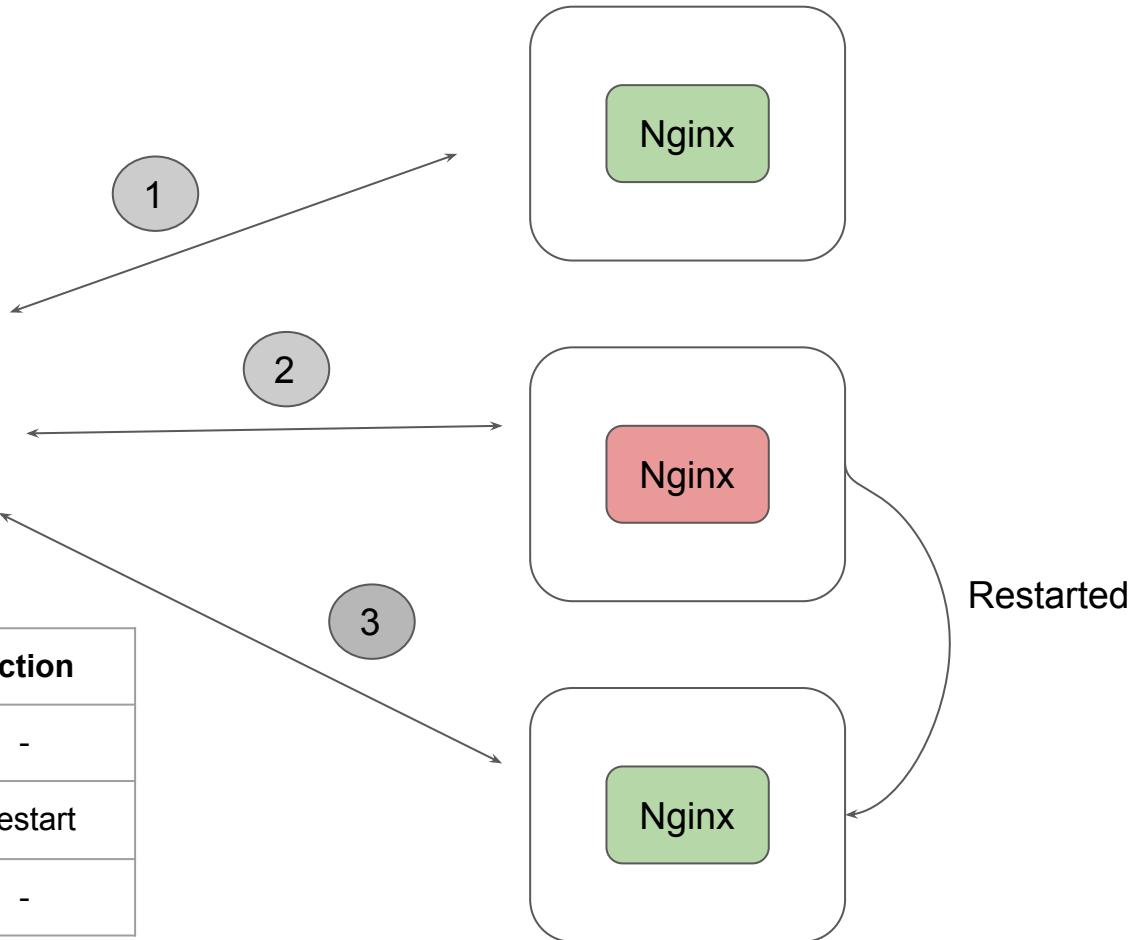
# Overview of Liveness Probe

Many applications running for long periods of time eventually transition to broken states, and cannot recover except by being restarted.

Kubernetes provides liveness probes to detect and remedy such situations.



Probe	Status	Action
1	Heathy	-
2	Unhealthy	Restart
3	Heathy	-



# Types of Probes

There are 3 types of probes which can be used with Liveness

- HTTP
- Command
- TCP

In this demo we had taken an example based on command.

---

# Readiness Probe

Readiness Checks

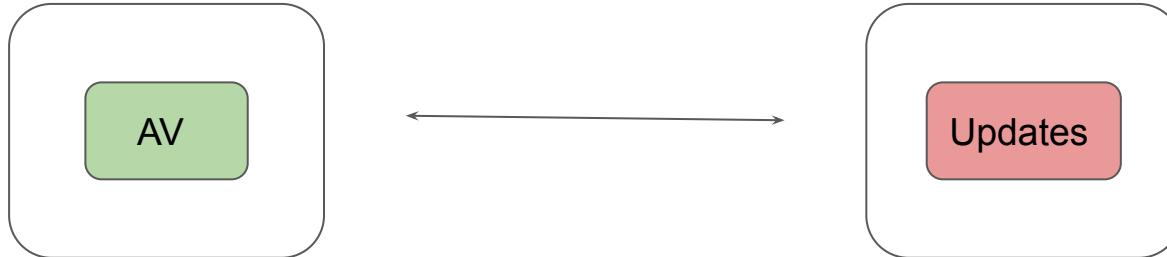
---

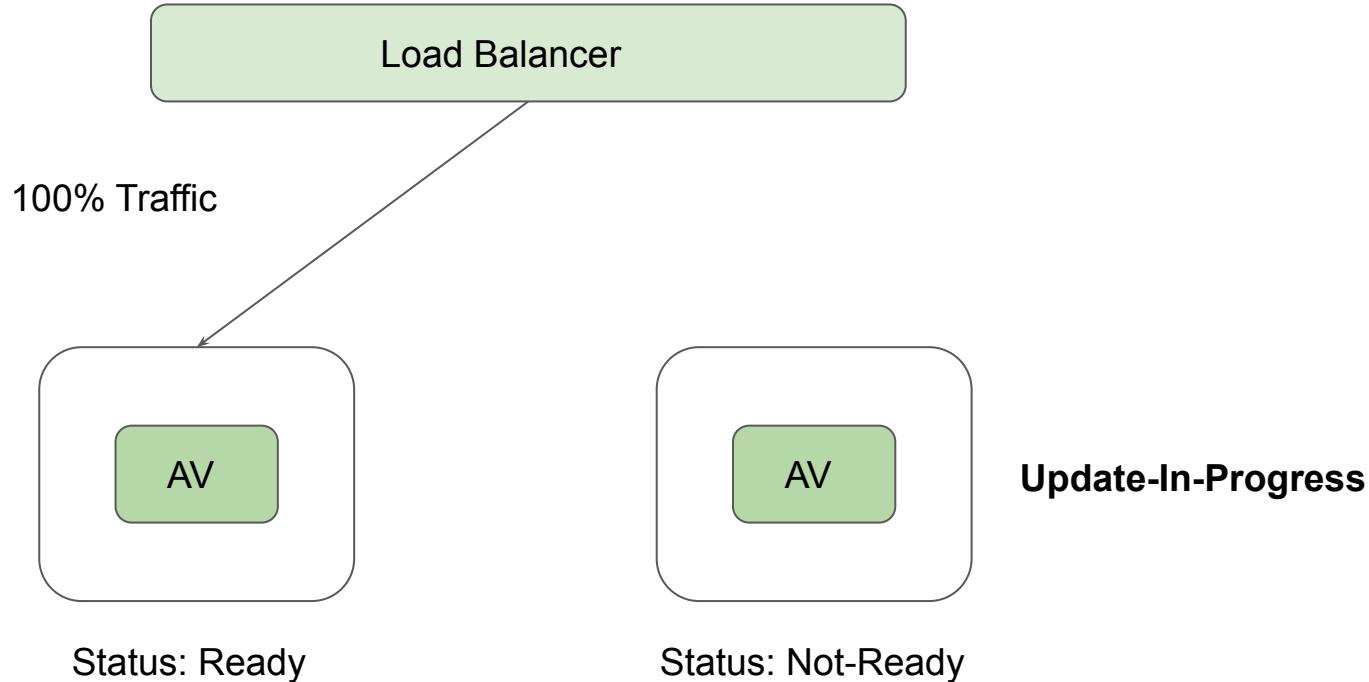
# Overview of Readiness Probe

It can happen that an application is running but temporarily unavailable to serve traffic.

For example, application is running but it is still loading its large configuration files from external vendor.

In such-case, we don't want to kill the container however we also do not want it to serve the traffic.





# Overview of Readiness Probe

Syntax of Readiness Probe:

readinessProbe:

exec:

command:

- cat

- /tmp/healthy

initialDelaySeconds: 5

periodSeconds: 5

---

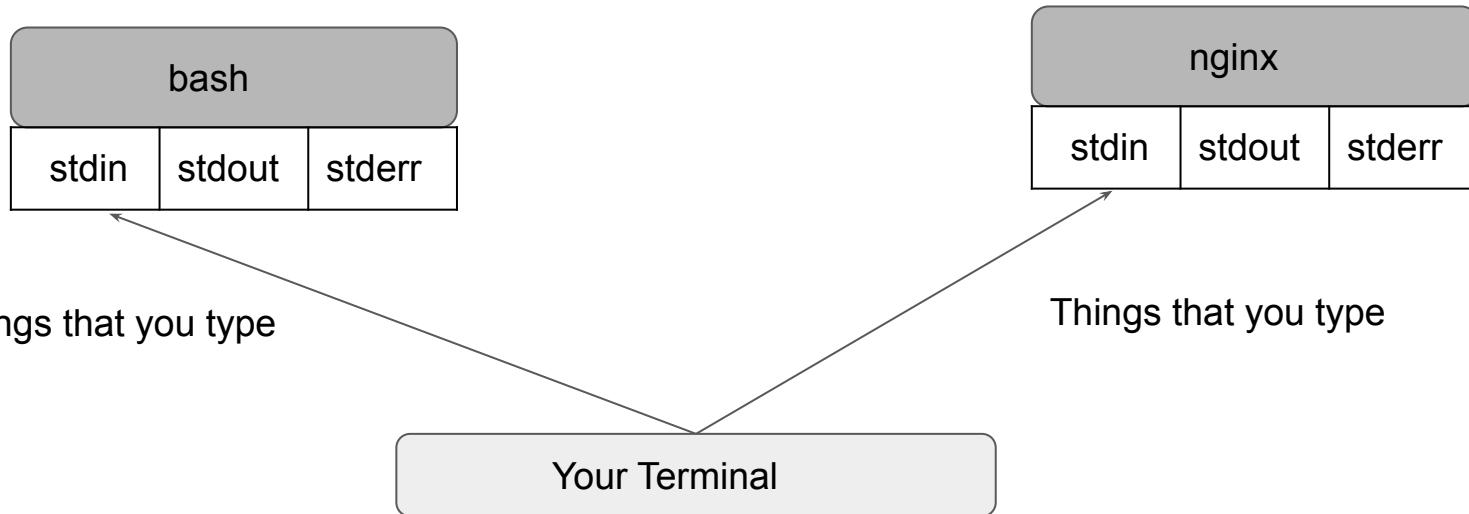
# Logging Drivers

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.

---

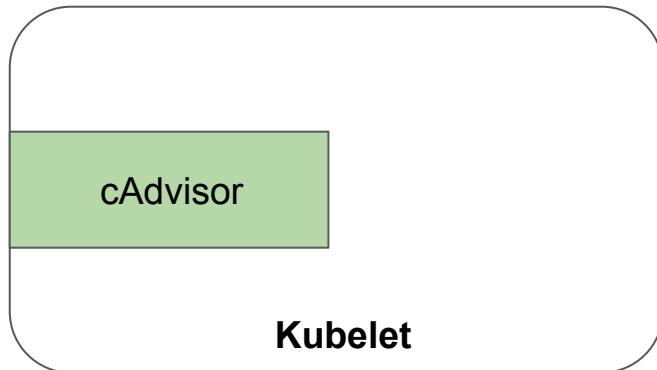
# Monitoring Cluster Components

Let's Monitor!

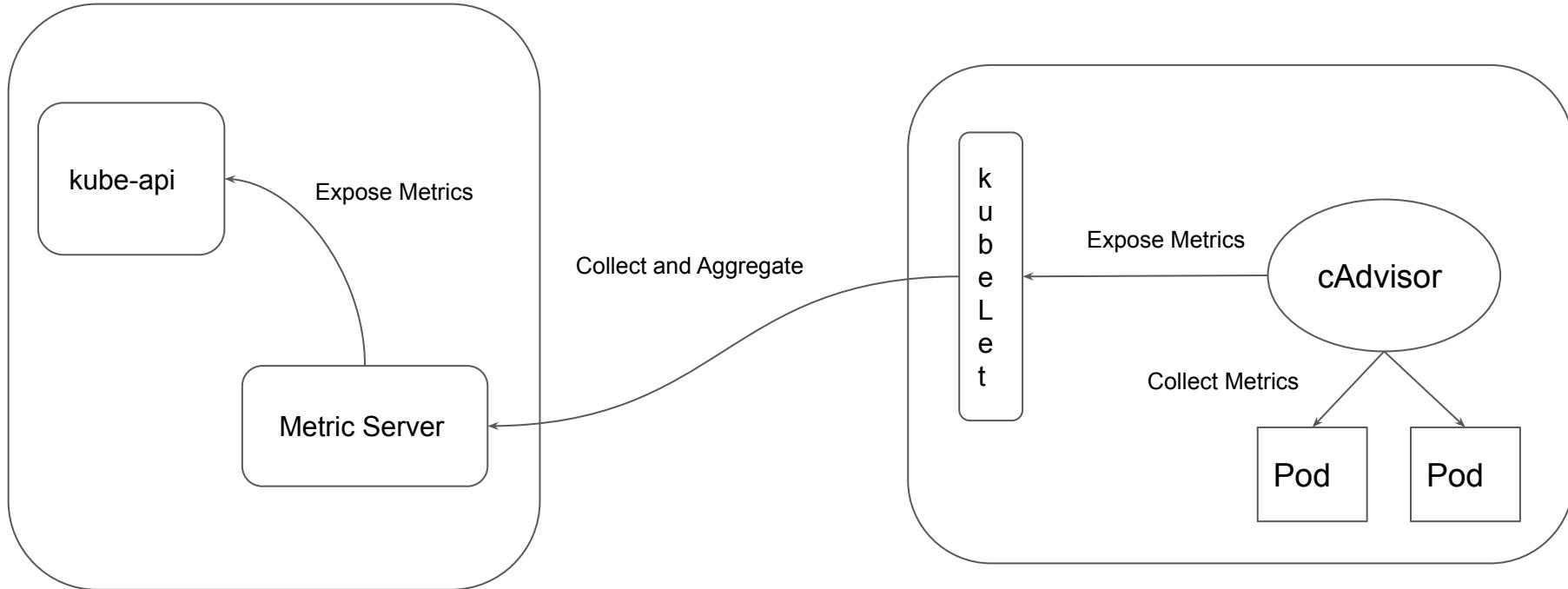
# Overview about cAdvisor

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.



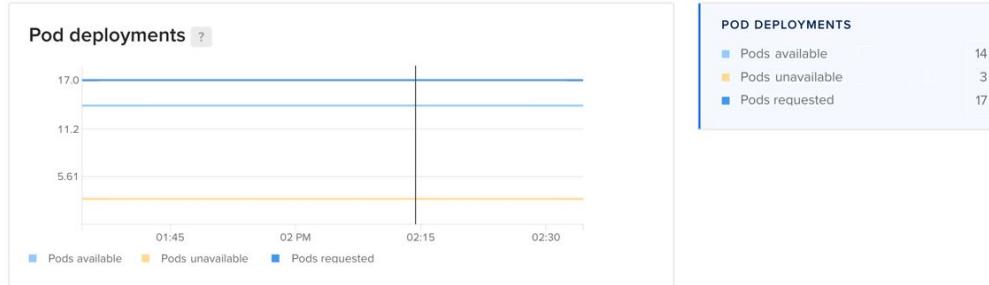
# Overall Workflow



# 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.



---

# 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.

---

# Docker Volumes

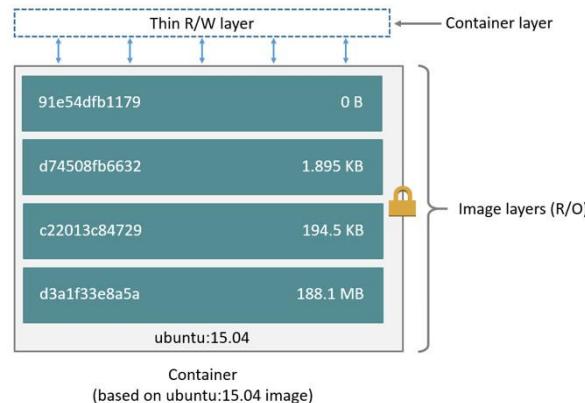
Build once, use anywhere

---

# Challenges with files in Container Writable Layer

By default all files created inside a container are stored on a writable container layer. This means that:

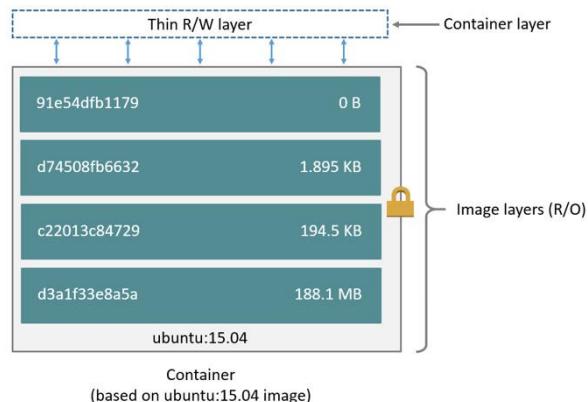
The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.



# Challenges with files in Container Writable Layer - Part 2

Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel.

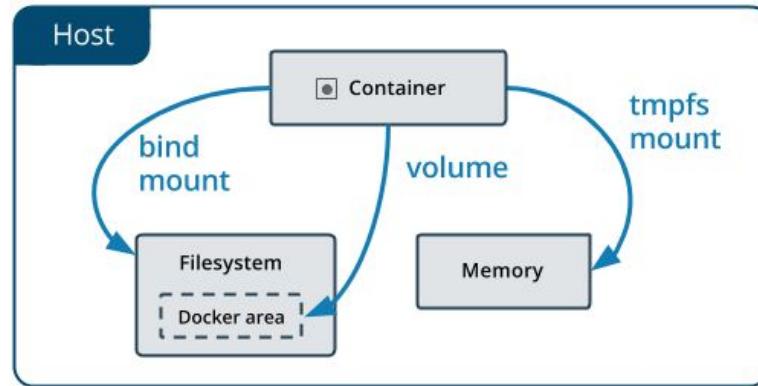
This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.



# Ideal Approach for Persistent Data

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.

If you're running Docker on Linux you can also use a tmpfs mount.



# Important Pointers to Remember

A given volume can be mounted into multiple containers simultaneously.

When no running container is using a volume, the volume is still available to Docker and is not removed automatically.

When you mount a volume, it may be named or anonymous. Anonymous volumes are not given an explicit name when they are first mounted into a container, so Docker gives them a random name that is guaranteed to be unique within a given Docker host.

---

# PersistentVolume and PersistentVolumeClaim

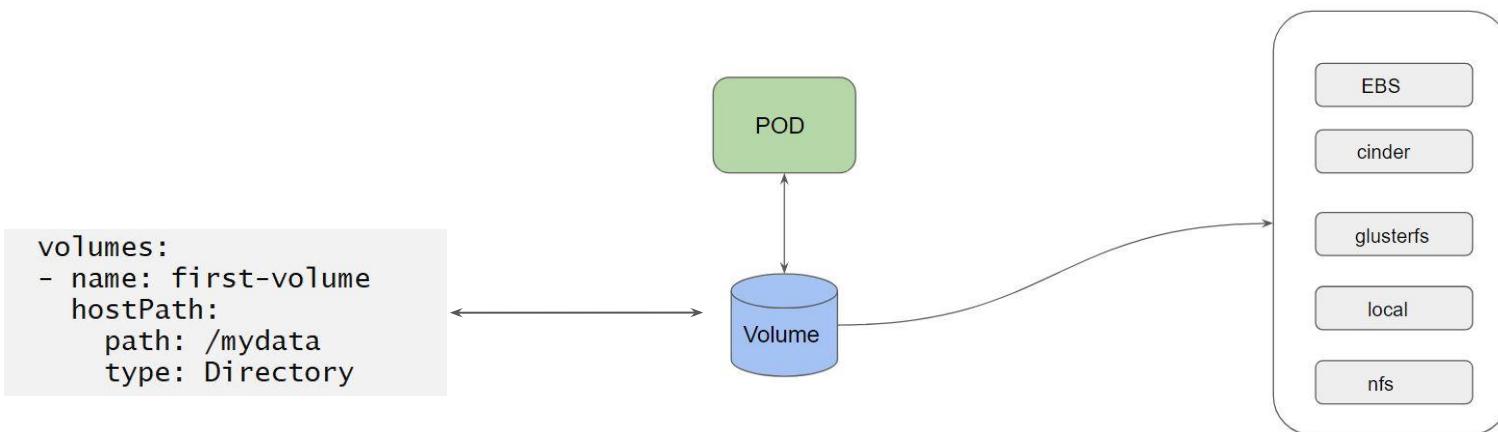
Storage Aspect

---

# Volumes in Kubernetes

Generally developers creates the YAML files associated with pods that they want to deploy.

In a scenario of directly referencing to volumes, developer need to be aware of configurations



# Understanding the Challenge



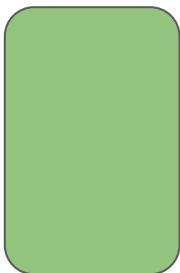
I am a Developer. I just want to write my code and pod.yaml file. I don't want to take care of storage provisioning, and its configurations.



I am a Storage Administrator. I can take care of provisioning storage and its associated configurations. Developers can reference the storage within their podspec.

# Persistent Volumes

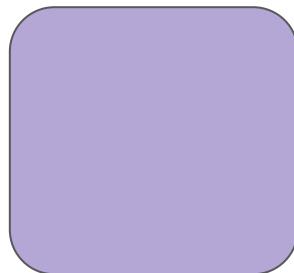
A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes



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



Volume 5  
Size: Very Big  
Speed: Very Slow

# Different Types of Persistent Volumes

- Every Volume which is created can be of different type.
- This can be taken care by the Storage Administrator / Ops Team



Volume 1  
Size: Small  
Speed: Fast



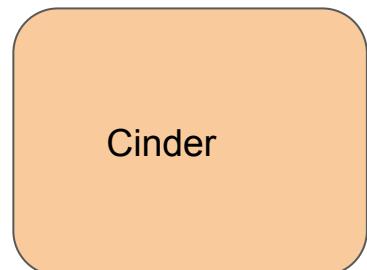
Volume 2  
Size: Medium  
Speed: Fast



Volume 3  
Size: Big  
Speed: Slow



Volume 4  
Size: VSmall  
Speed: Ultra Fast



Volume 5  
Size: Very Big  
Speed: Very Slow

# PersistentVolumeClaim

A PersistentVolumeClaim is a request for the storage by a user.

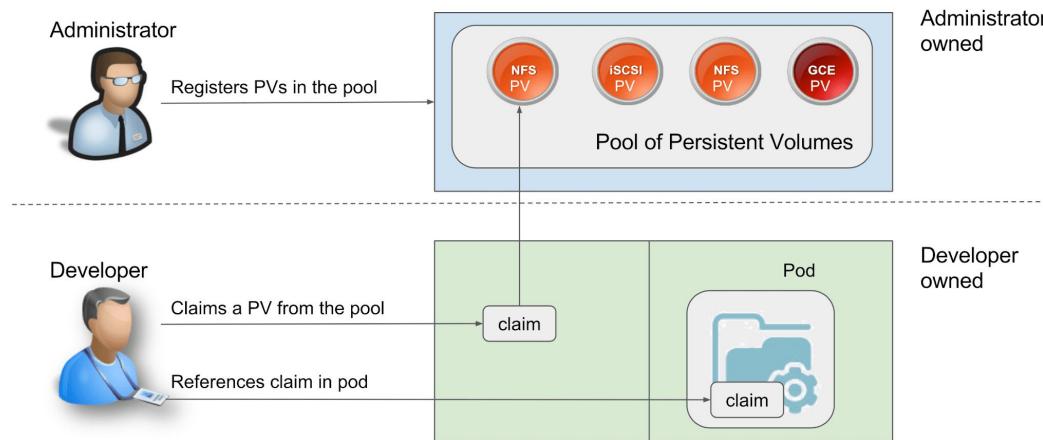
Within the claim, user need to specify the size of the volume along with access mode.

**Developer:**

I want a volume of size 10 GB which is has speed of Fast for my pod.

# Overall Working Steps

1. Storage Administrator takes care of creating PV.
2. Developer can raise a “Claim” (I want a specific type of PV).
3. Reference that claim within the PodSpec file.



# **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.

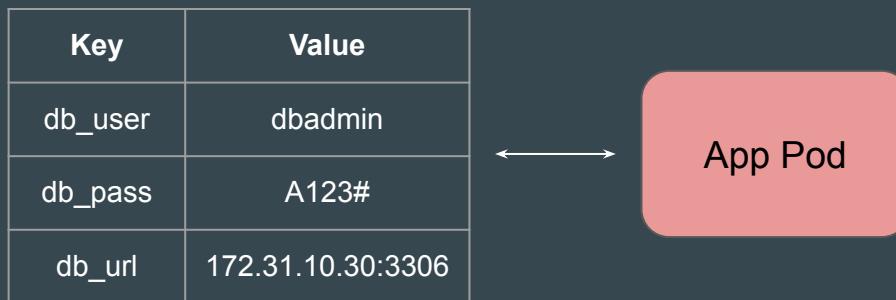
Key	Value
db_user	dbadmin
db_pass	A123#
db_url	172.31.10.30:3306



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



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.

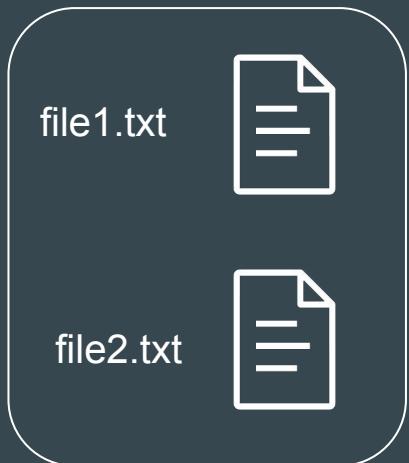


```
kubectl create configmap --from-file=large-file.txt
```

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>
```

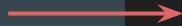
# Comparison of All the Methods

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.



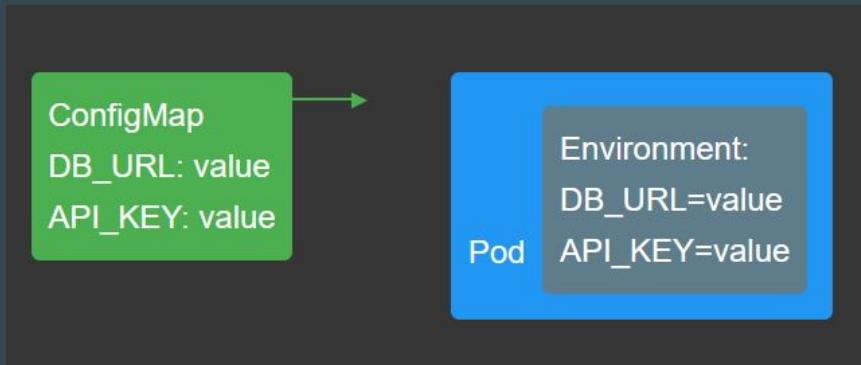
# Reference Manifest File

```
! 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

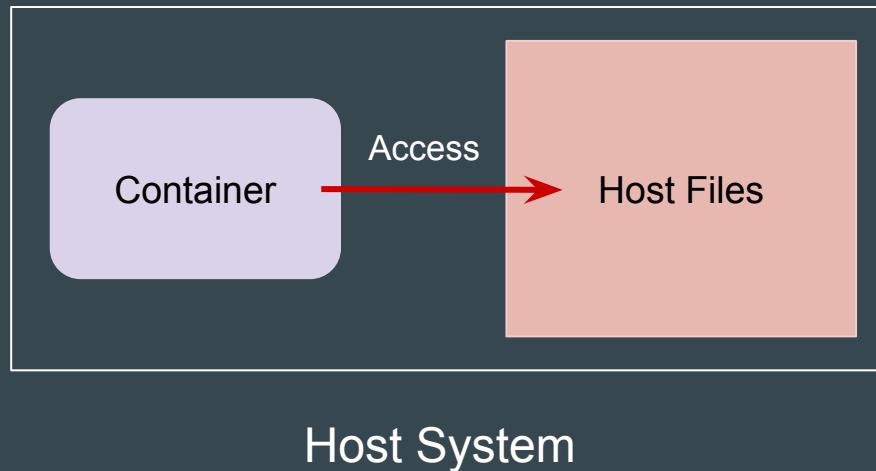
```
! pod-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
    - name: app-container
      image: nginx
      env:
        - name: APP_MODE
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: APP_MODE
        - name: APP_ENV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

# **Security Context**

# Understanding the Challenge

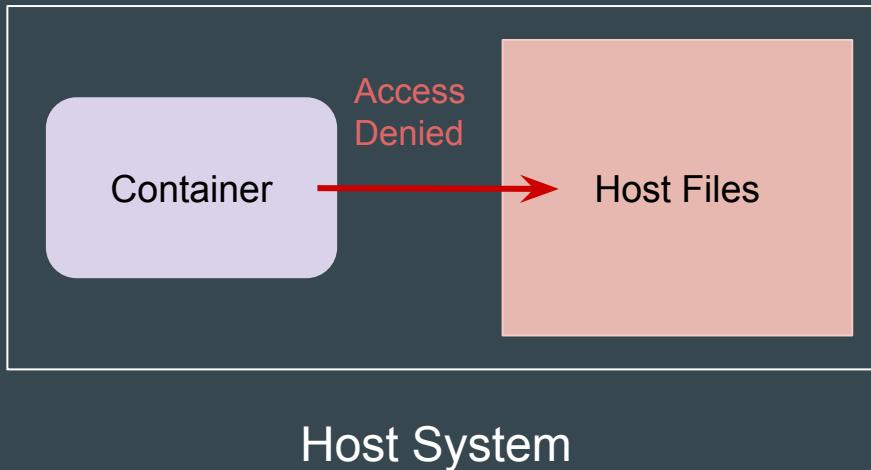
Many times, the containers run with **root user privileges**.

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



# Running Container with Non Root User

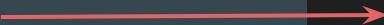
If the container runs with non-root privileges, it will be unable to modify the critical host files and will have limited access to the host system.



# 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).

---

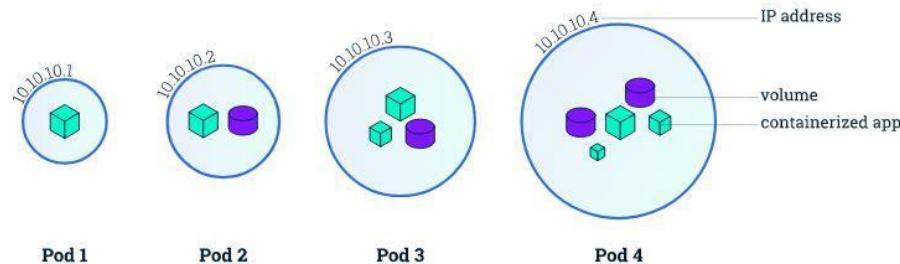
# Multi-Container Pods

Let's get started

# Kubernetes POD

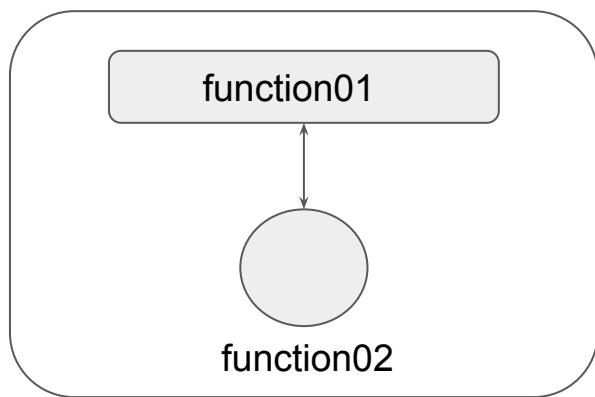
A Pod in Kubernetes represents a group of one or more application containers , and some shared resources for those containers.

A single pod can have multiple containers running.

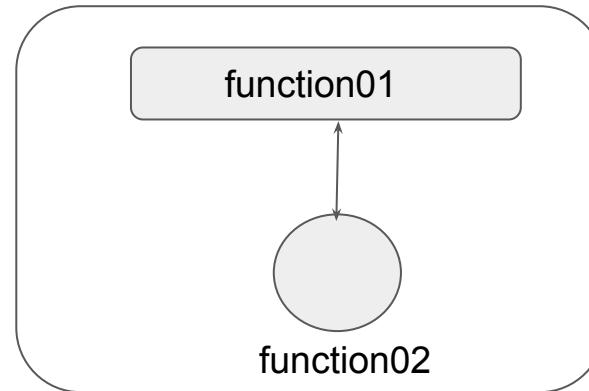


# Dealing with Tightly Coupled Application

Containers within a Pod share an IP address and port space, and can find each other via localhost.



**POD 1**



**POD 2**

---

# Multi-Container POD Patterns

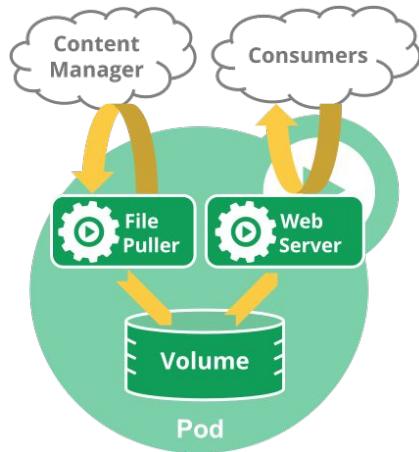
Multi-Container Patterns

---

# 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.

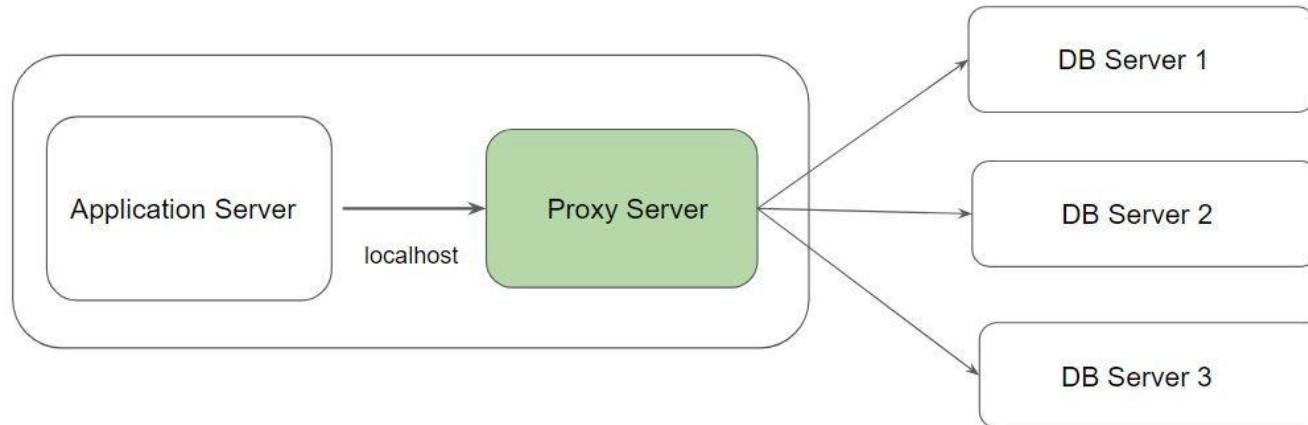


# Overview of Sidecar Pattern



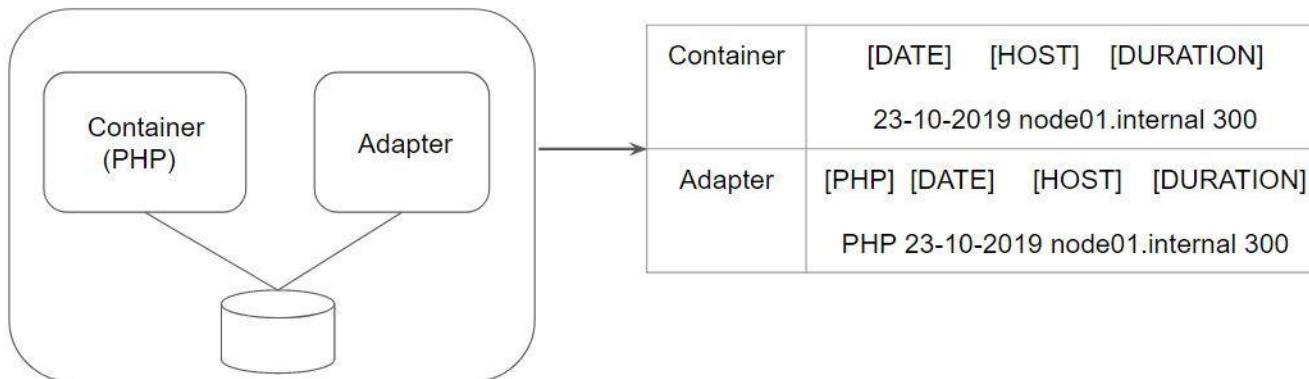
# Ambassador Pattern

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.



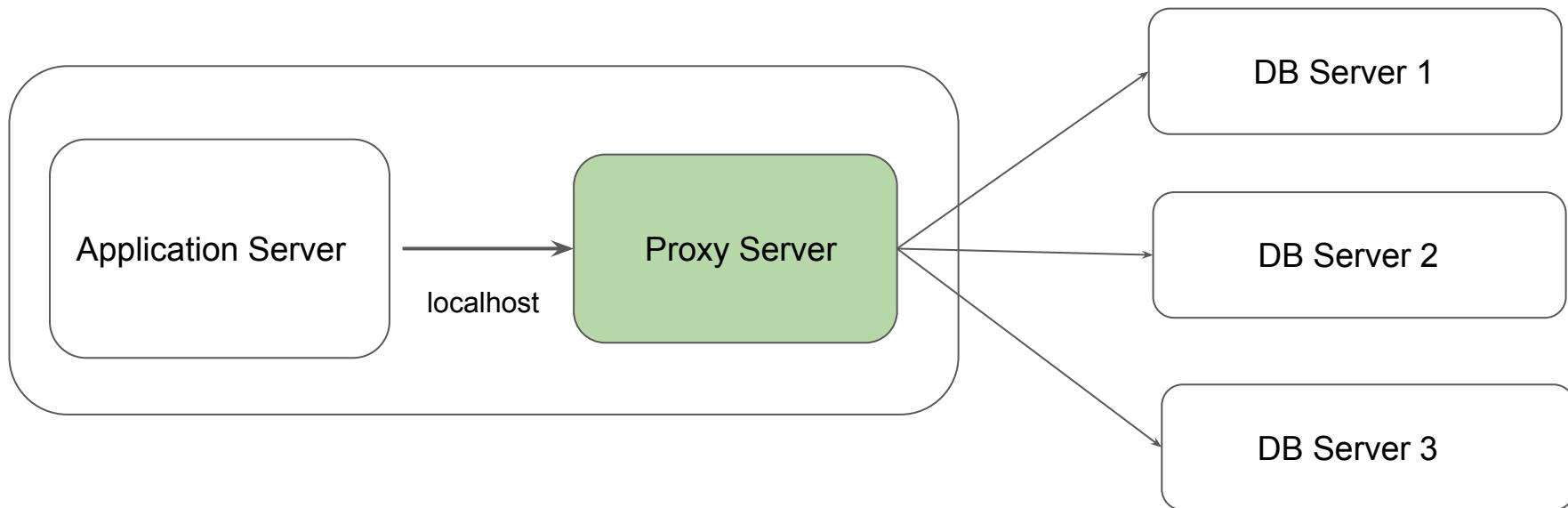
---

# Ambassador Pattern

## Multi-Container Patterns

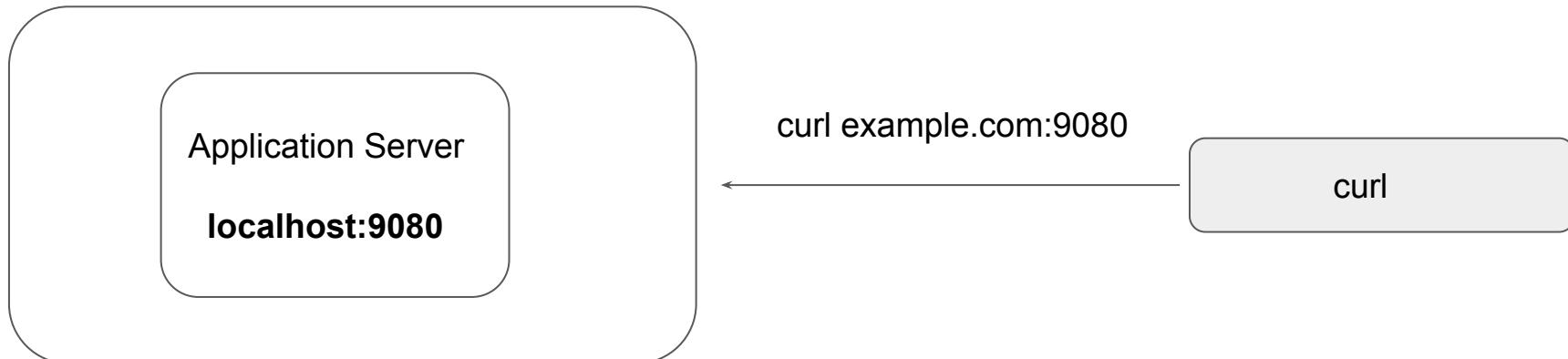
# Overview of Ambassador Pattern

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



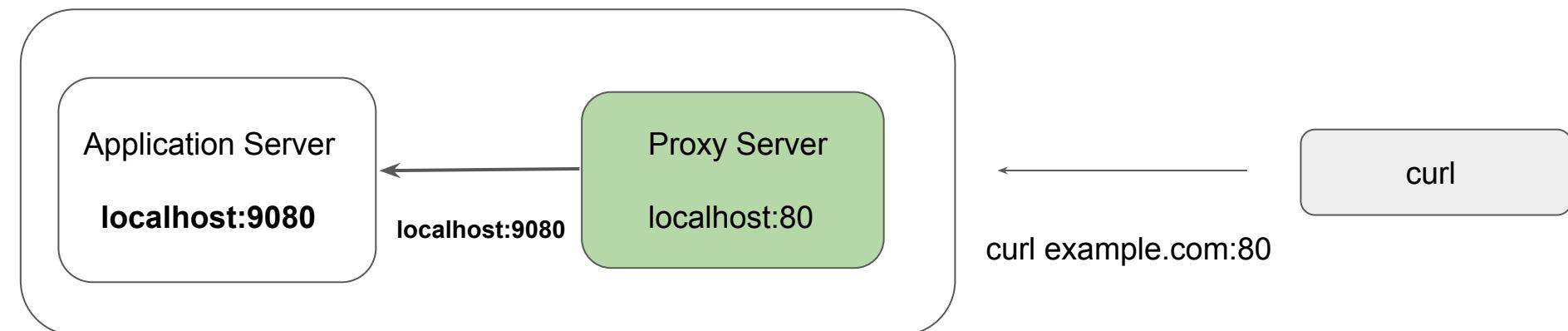
# Different Architecture Pattern

- You have a legacy application server which listens on port 9080.
- This port is hardcoded in the image and cannot be changed.
- You want external applications to call the application server on port 80



# Solution - Ambassador Container

- Create a proxy server which listens on Port 80.
- Forward all the requests to port 9080.



---

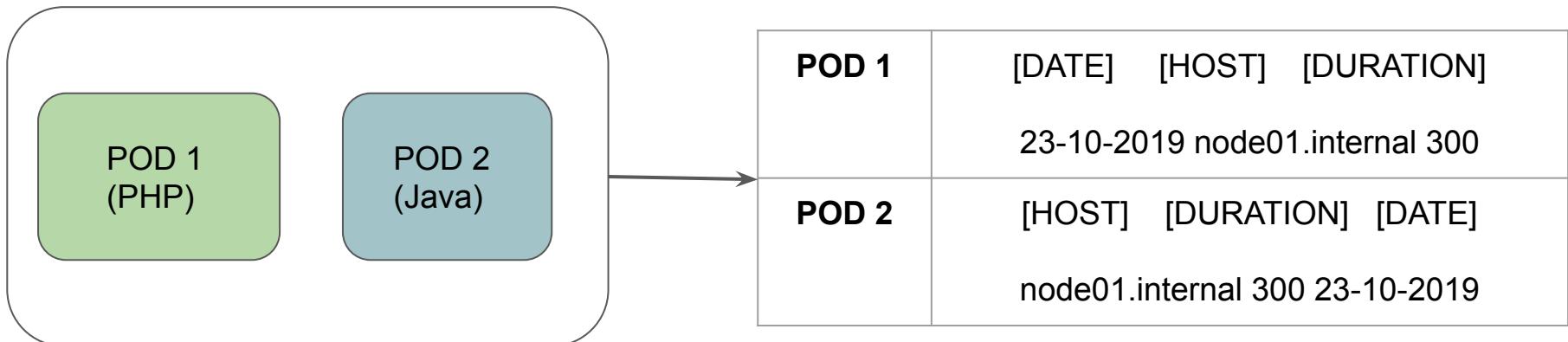
# Adapter Pattern

## Multi-Container Patterns



# Overview of Adapter Pattern

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



# Standardizing Log Format

Within your logging application, you want to have a common standard format.

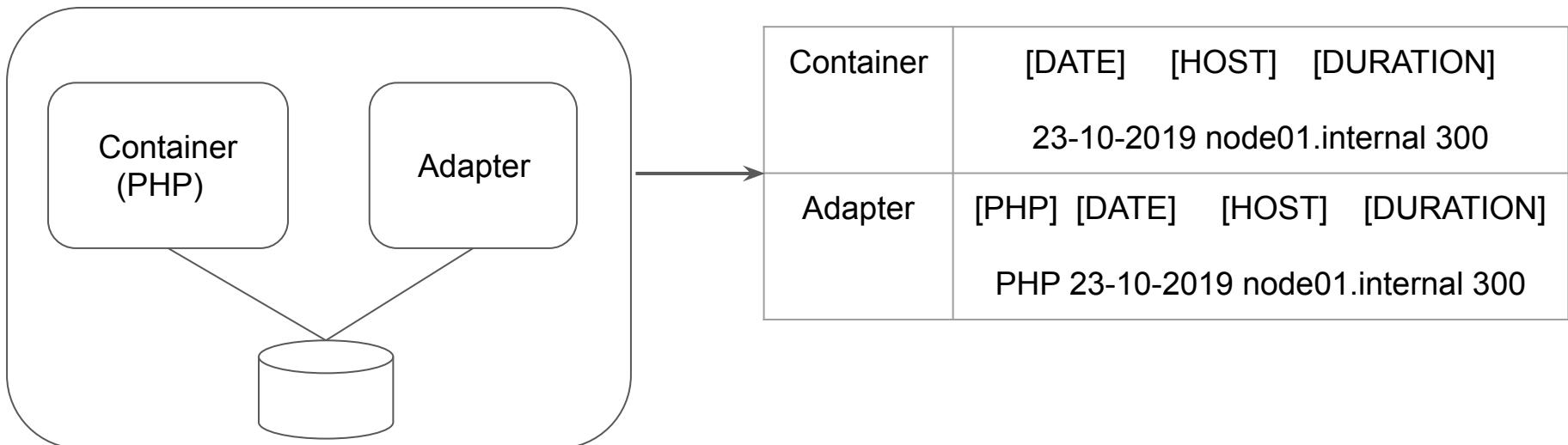
[PHP|JAVA] [DATE] [HOST] [DURATION]

PHP 23-10-2019 node01.internal 300

# Transforming Logs

With Adapter Container, you can transform your logs to standardize it.

Since containers in PODS can share volumes, adapter container can easily access App logs.



---

# K8s Deployment Strategy

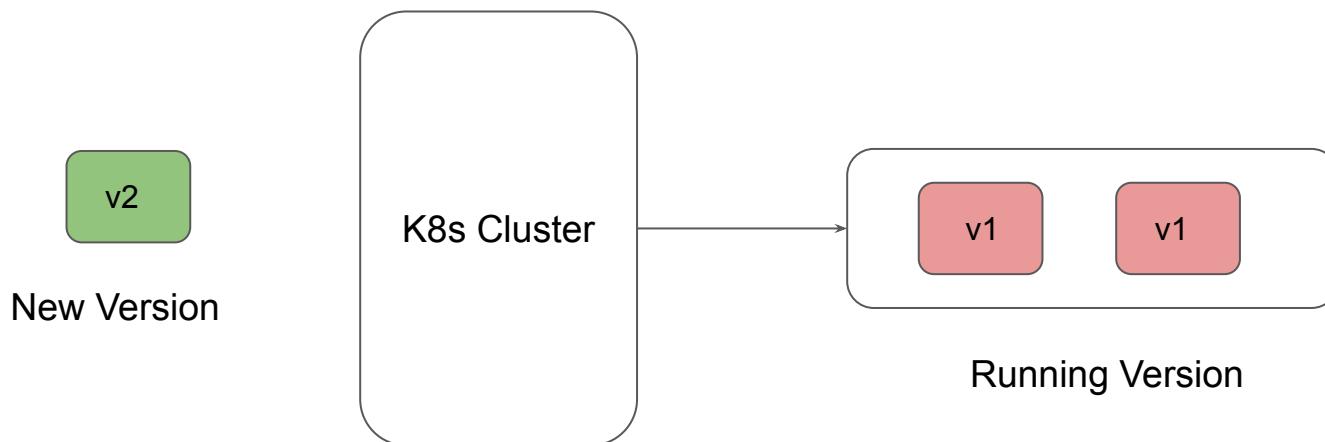
Deploying New Application Updates

---

# Overview of Deployment Strategy

A deployment strategy is a way to change or upgrade an application

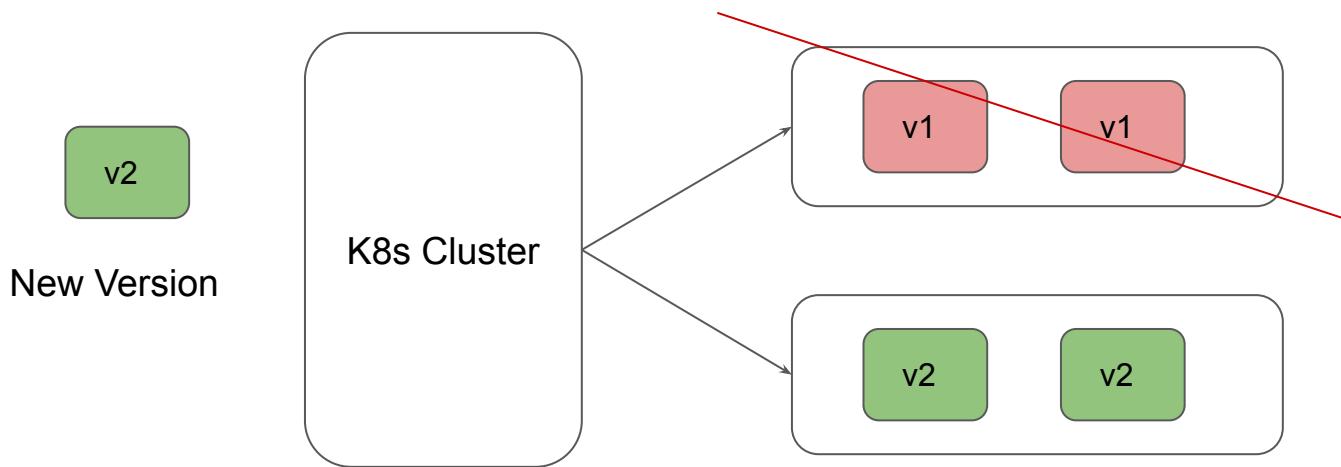
How will you deploy v2 of the application to the Kubernetes cluster?



# Deployment Strategy - Recreate

All of the PODS get killed all at once and get replaced all at once with the new ones.

Recommended for Dev/Test environment.



# Advantages and Disadvantages - Recreate

## Advantages:

Overall easy to setup and deploy.

## Disadvantages:

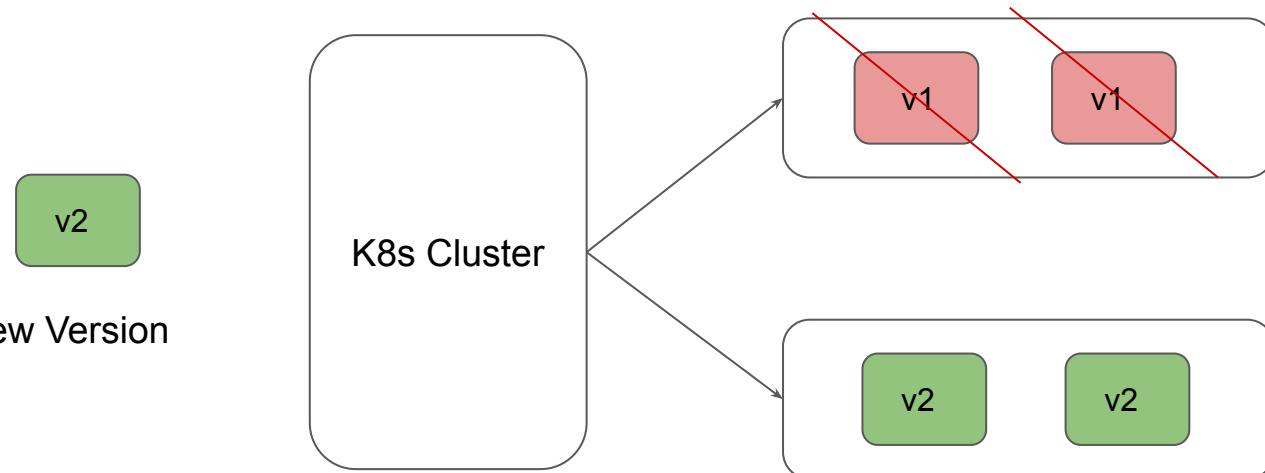
There will be certain amount of downtime for the users.

# Traffic Pattern - Recreate Stratery



# Deployment Strategy - RollingUpdate

Your application is deployed to your environment one batch of instances at a time.



# Traffic Pattern - Rolling Update Strategy



# Advantages and Disadvantages - Recreate

## Advantages:

New version is slowly released and can reduce the downtime.

## Disadvantages:

Takes more amount of time for the overall rollout process.

No control over the traffic.

# Blue Green Deployment Strategy

**Blue environment** is an existing environment in production receiving live traffic.

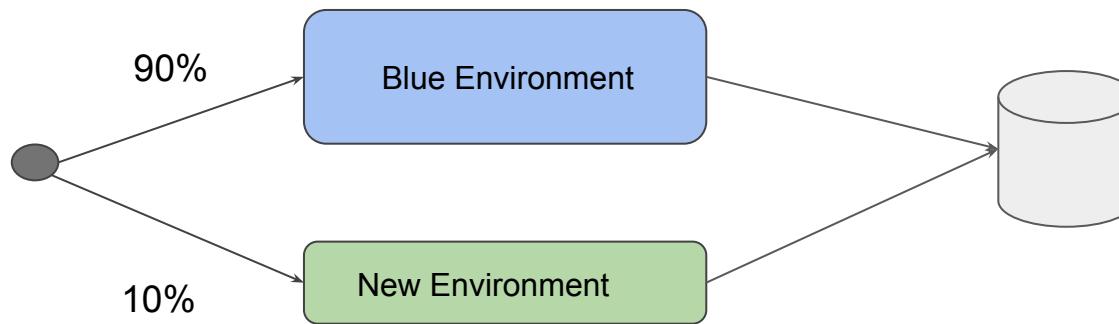
**Green environment** is parallel environment running different version of the application.

Deployment = Routing production traffic from blue to green environment.



# Understanding Canary Deployment Model

**Canary Deployment** is a process where we deploy a new feature and shift some % of traffic to the new feature to perform some analysis to see if feature is successful.



---

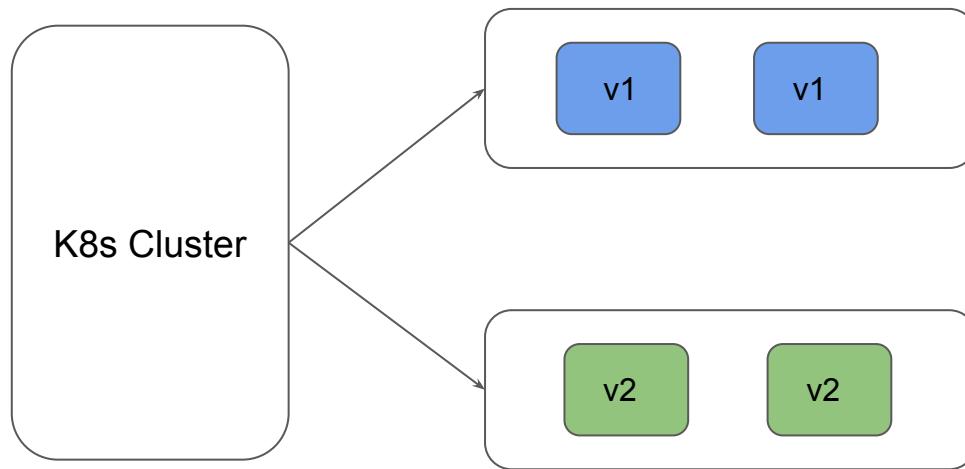
# Blue/Green Deployments

Deploying New Application Updates

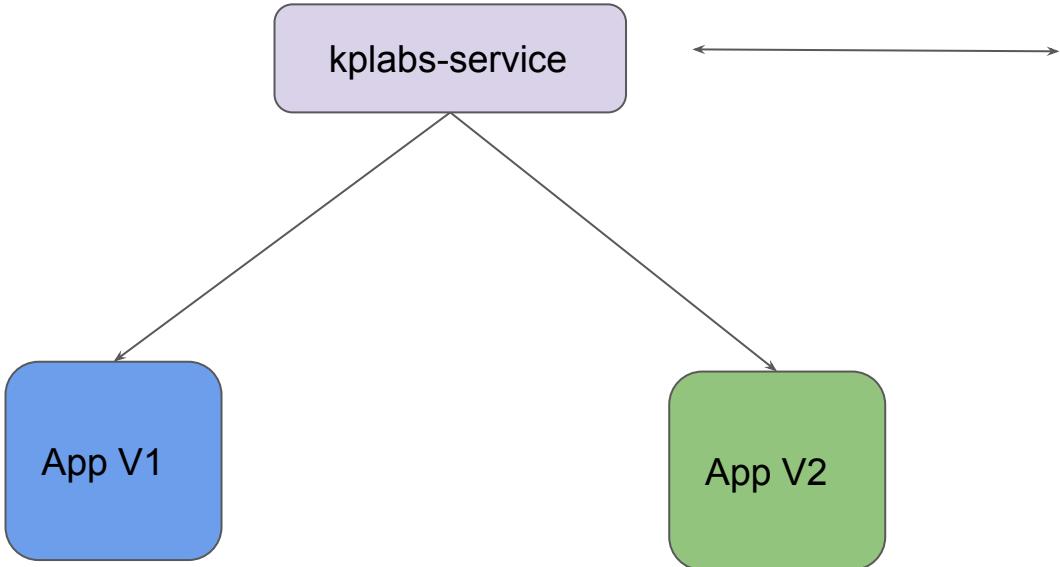
---

# Deployment Strategy - Blue Green

Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.



# Possible Implementation Method - 1



```
apiVersion: v1
kind: Service
metadata:
  name: application-service
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
  selector:
    app: demo-app-v2
```

# Advantages and Disadvantages - Blue/Green

## Advantages:

Instantaneous rollouts and rollbacks if required.

## Disadvantages:

Requires double the amount of resources.

---

# Canary Deployments

Deploying New Application Updates

---

# Story behind Canary

The story goes back to the old british mining practise.

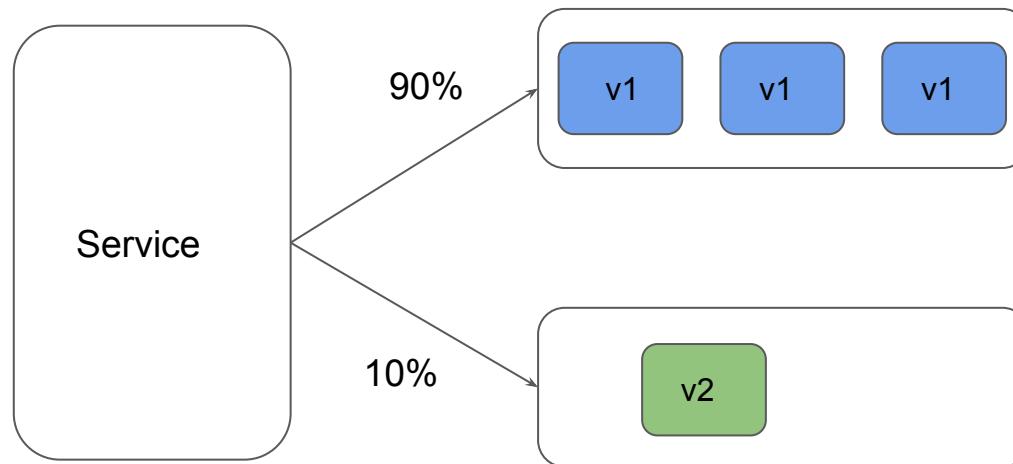
Practice included using "canaries" in coal mines to detect carbon monoxide and other toxic gases before they hurt humans."

To make sure mines were safe for them to enter, miners would take canaries; **if something bad happened** to the canary, it was a warning for the miners to abandon the mine.



# Deployment Strategy - Canary

Canary Deployment is a process where we deploy a new feature and shift some % of traffic to the new feature to perform some analysis to see if feature is successful.



---

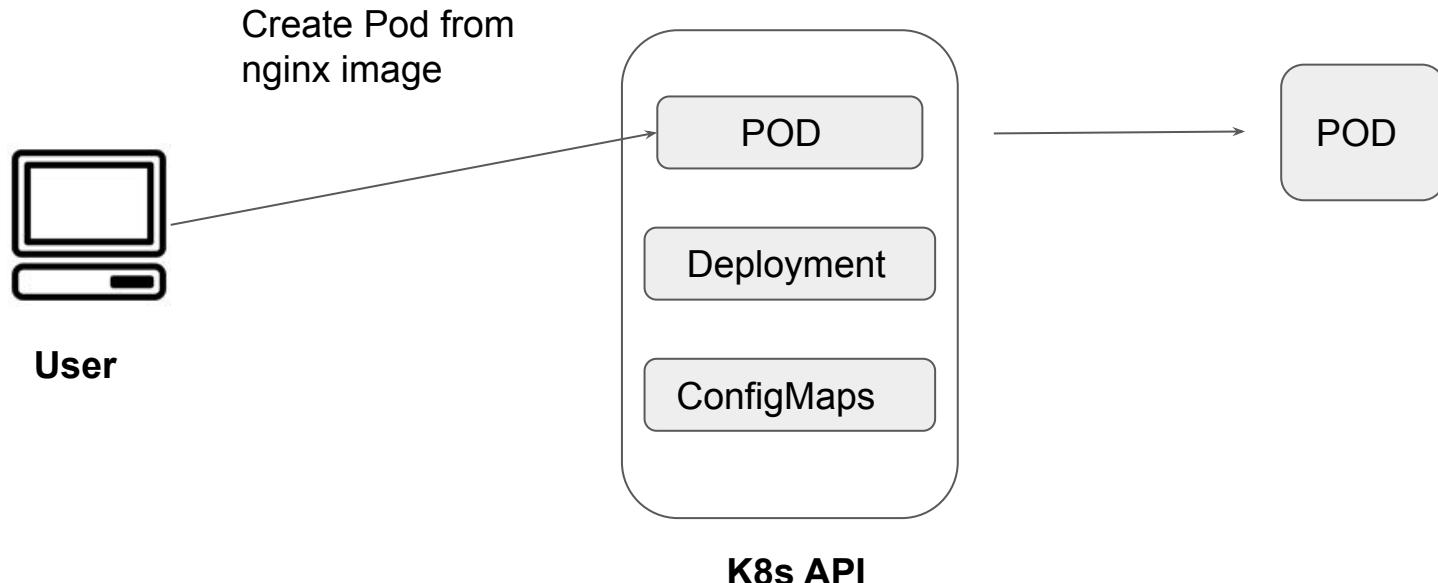
# Custom Resources

Create Custom Logic

---

# Understanding the Basics

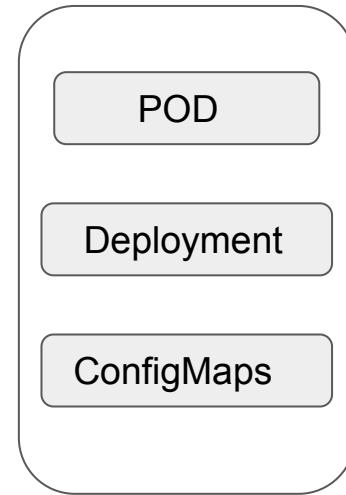
A resource is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind.





kubectl run <pod>

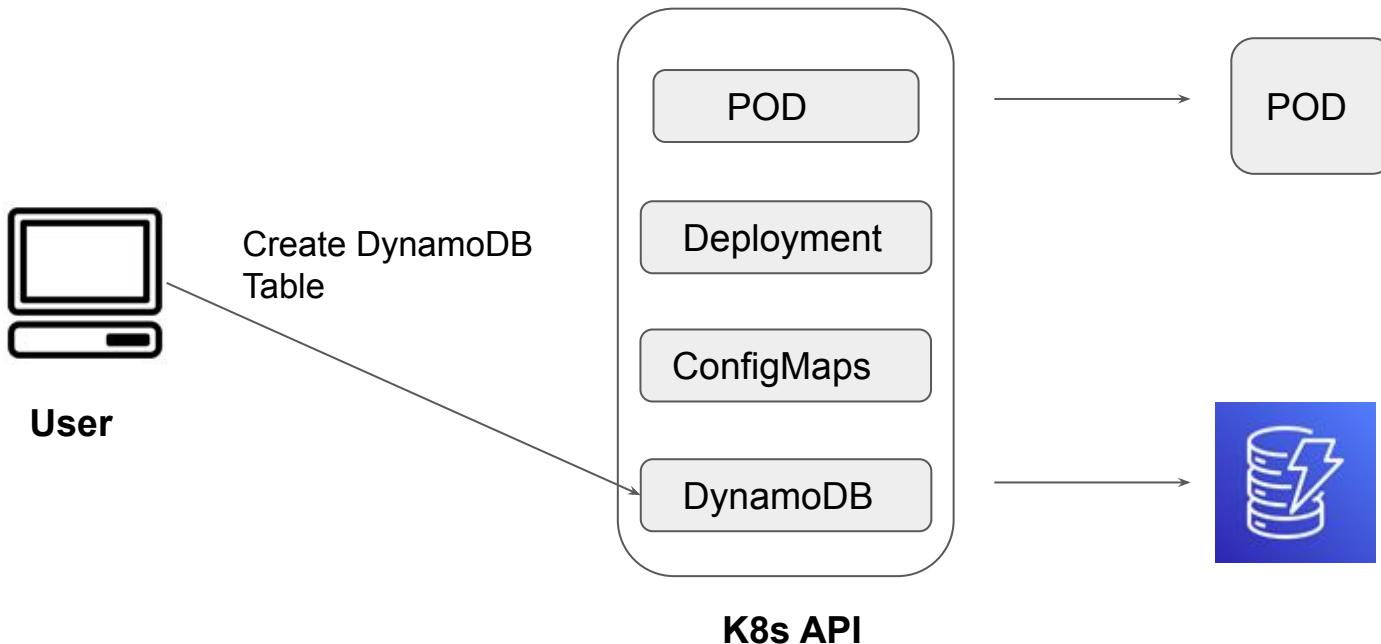
```
{  
    "apiVersion": "v1",  
    "kind": "Pod",  
    "metadata": {  
        "name": "nginx1"  
    },  
    "spec": {  
        "containers": [  
            {  
                "name": "nginx",  
                "image": "nginx:1.7.9",  
                "ports": [  
                    {  
                        "containerPort": 80  
                    }  
                ]  
            }  
        ]  
    }  
}
```



**K8s API**

# Understanding the CRDs

Custom Resource Definition (CRDs) are extensions of Kubernetes API that stores collection of API objects of certain kind.



# 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: crontabs.kplabs.internal
spec:
  group: kplabs.internal
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: number
```

## Step 2 - Create Custom Objects

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

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

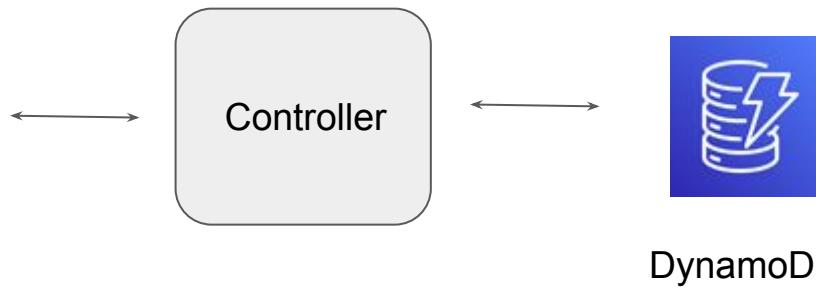
```
apiVersion: "kplabs.internal/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: demo-replica
```

# Importance of Kubernetes 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: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: demo-replica
```



DynamoDB

---

# Authentication

Kubernetes Authentication

---

# Basics of Authentication

Authentication is the process or action of verifying the identity of a user or process.



kubectl delete pod database



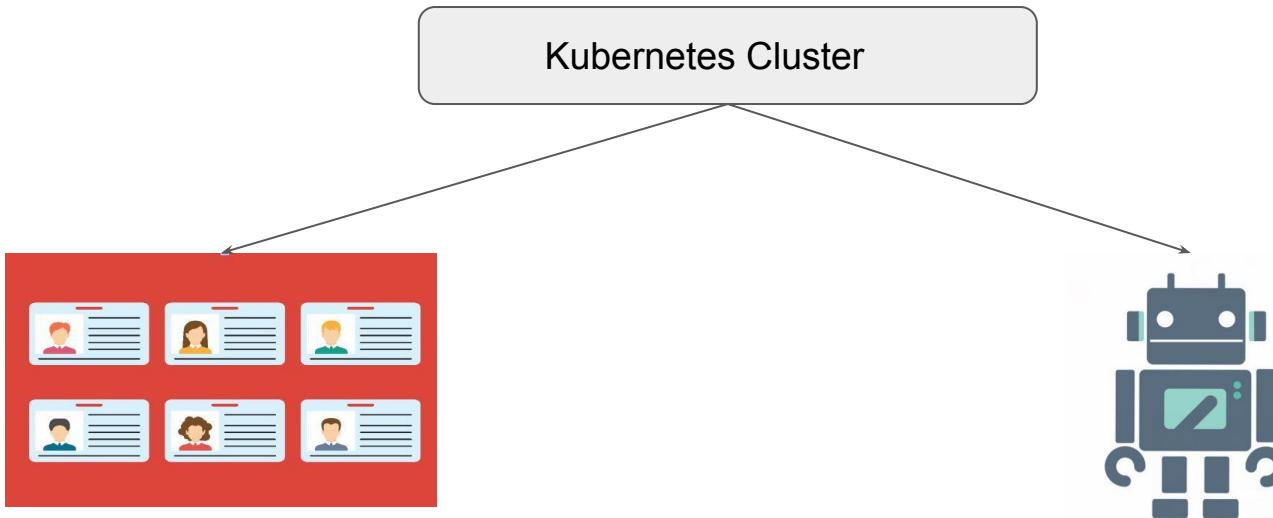
K8s Cluster

Who are you? Authenticate first.

# Categories of Users

Kubernetes Clusters have two categories of users:

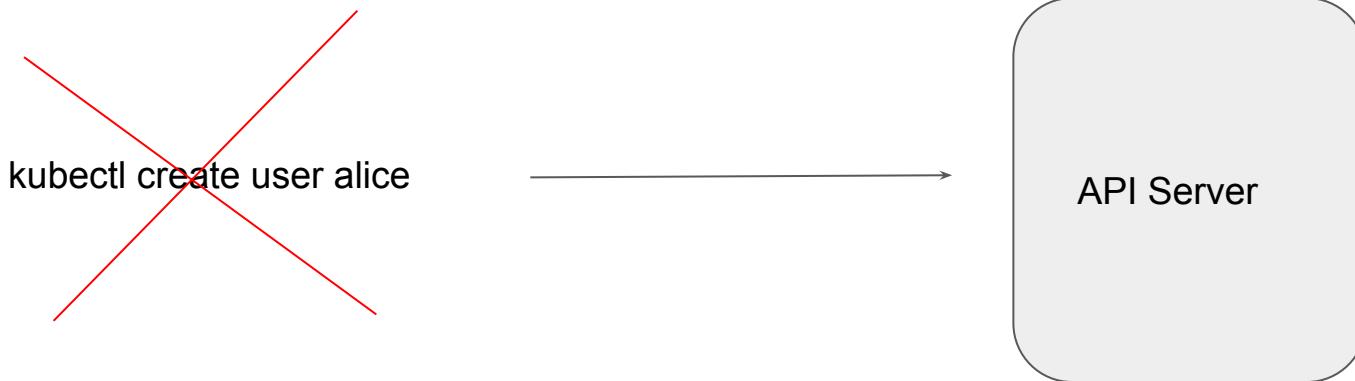
- Normal Users.
- Service Accounts



# Important Aspect

Kubernetes does not have objects which represent normal user accounts.

Kubernetes does not manage the user accounts natively.



# Authentication Strategies

There are multiple ways in which we can authenticate. Some of these include:

- Usernames / Passwords.
- Client Certificates
- Bearer Tokens
- etc

Let's take a demo for the same.

---

# Authorization

Authorization Step

---

# Basics of Authorization

Authorization is the process of giving someone permission to perform some operation.



New AWS User

Delete All the Servers

---



AWS Account

# Authorization Modules

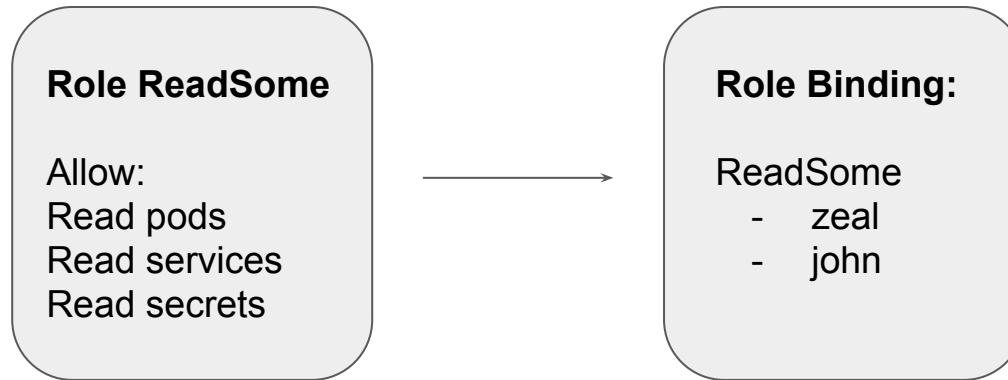
Kubernetes provides multiple authorization modules like:

- AlwaysAllow
- AlwaysDeny
- Attribute-Based Access Control (ABAC)
- **Role-based access control (RBAC)**
- Node
- WebHook

# Two Concepts

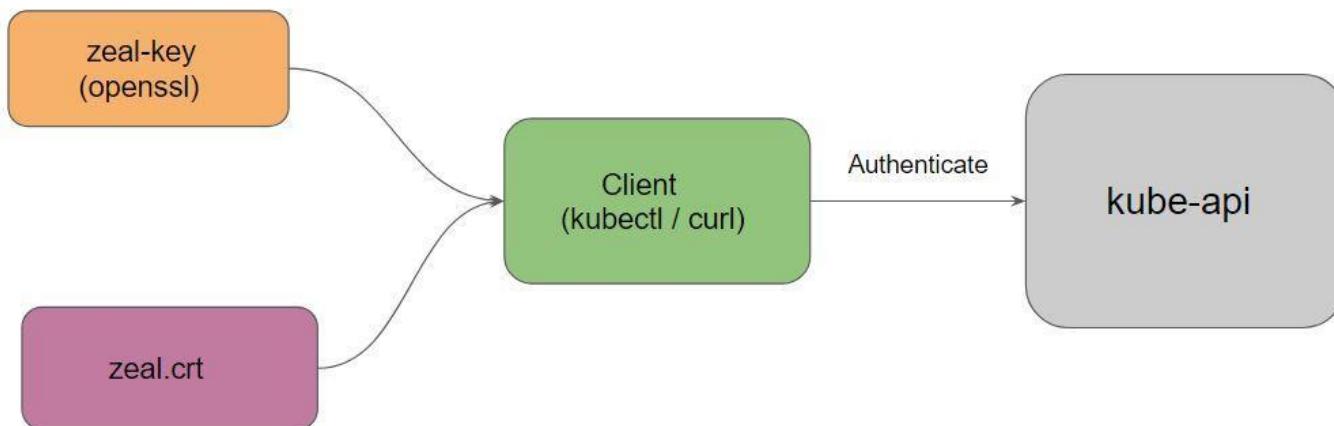
A **role** contains rules that represent a set of permissions

A **role binding** grants the permissions defined in a role to a user or set of users.



# Understanding Authorization

By default, the user do not have any permission granted to them.



---

# ClusterRole and ClusterRoleBinding

Security Aspect

# Overview of ClusterRole

A Role can only be used to grant access to resources within a single namespace.

A ClusterRole can be used to grant the same permissions as a Role, but because they are cluster-scoped, they can also be used to grant access to:

- cluster-scoped resources (like nodes)
- namespaced resources (like pods) across all namespaces

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["get", "watch", "list", "create"]
```

# Important Pointer

A RoleBinding may also reference a ClusterRole to grant the permissions to resources defined in the ClusterRole within the RoleBinding's namespace

This allows the administrator to have set of central policy which can be attached via RoleBinding so it is applicable at a per-namespace level.



---

# Images and Containers

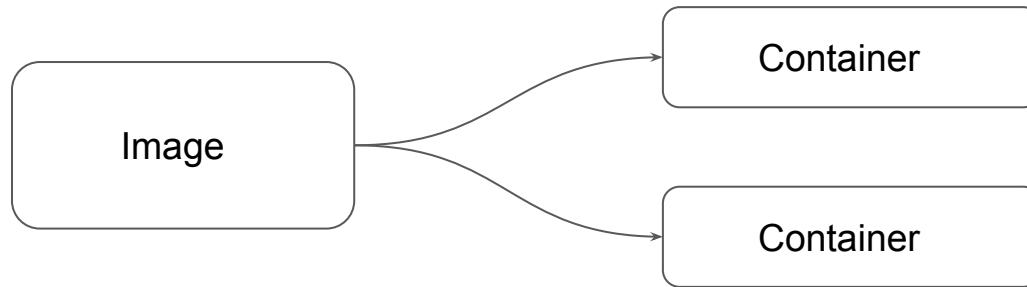
Build once, use anywhere

---

# Images and Containers

Docker Image is a file which contains all the necessary dependency and configurations which are required to run an application.

Docker Containers is basically a running instance an image.



---

# Container Identification

Build once, use anywhere

---

# Getting Started

When you create a Docker container, it is assigned a universally unique identifier (UUID).

These can help identify the docker container among others.

```
[root@docker-demo ~]# docker run -dt -p 80:80 nginx  
d5187cb1c7f4380b3e37e0c0c811a437d7b8a49d5beb705711a4e54e99d72d77
```

# Random Container Names

To help humans, Docker also allow us to supply container names.

By default, if we do not specify the name, docker supplies randomly-generated name from two words, joined by an underscore

[root@docker-demo ~]# docker ps				
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
d5187cb1c7f4	nginx	"nginx -g 'daemon ..."	47 minutes ago	Up 31 minutes
0.0.0.0:80->80/tcp	inspiring_poitras			

# Naming Docker Container

By adding `--name=meaningful_name` argument during docker run command, we can specify our own name to the containers.

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx  
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969  
[root@docker-demo ~]# docker ps  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS  
          NAMES  
9fba1f62038d        nginx              "nginx -g 'daemon ..."   5 seconds ago      Up 5 seconds  
          0.0.0.0:800->80/tcp    mynginx
```

---

# Working with Docker Images

Build once, use anywhere

---

# Getting Started with Images

Every Docker container is based on an image.

Till now we have been using images which were created by others and available in Docker Hub.

Docker can build images automatically by reading the instructions from a **Dockerfile**

# Understanding Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.



---

# Dockerfile

Building Docker Images

---

# Understanding Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.



# Format of Dockerfile

The format of Dockerfile is similar to the below syntax:

```
# Comment  
INSTRUCTION arguments
```

A Dockerfile must start with a **FROM`** instruction.

The FROM instruction specifies the Base Image from which you are building.

# Dockerfile Commands

The format of Dockerfile is similar to the below syntax:

- FROM
- RUN
- CMD
- LABEL
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- Many More ...

# Use-Case - Create Custom Nginx Image

## Simple Use-Case

We want to create a custom Nginx Image from which all containers within organization will be launched.

The base container image should have custom index.html file which states:

“Welcome to Base Nginx Container from KPLABS”

..

---

# Docker Commit

Build once, use anywhere

---

# Getting Started

Whenever you make changes inside the container, it can be useful to commit a container's file changes or settings into a new image.

By default, the container being committed and its processes will be paused while the image is committed.

**Syntax:**

```
docker container commit CONTAINER-ID myimage01
```

# Change Option while Commiting

The --change option will apply Dockerfile instructions to the image that is created.

Supported Dockerfile instructions:

- CMD | ENTRYPOINT | ENV | EXPOSE
- LABEL | ONBUILD | USER | VOLUME | WORKDIR

---

# Layers of Docker Image

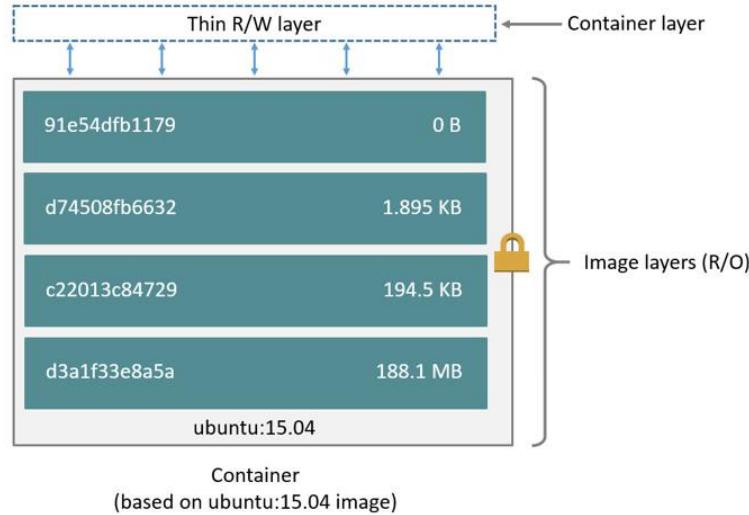
Building Docker Images

---

# Understanding Layers

- A Docker image is built up from a series of layers.
- Each layer represents an instruction in the image's Dockerfile.

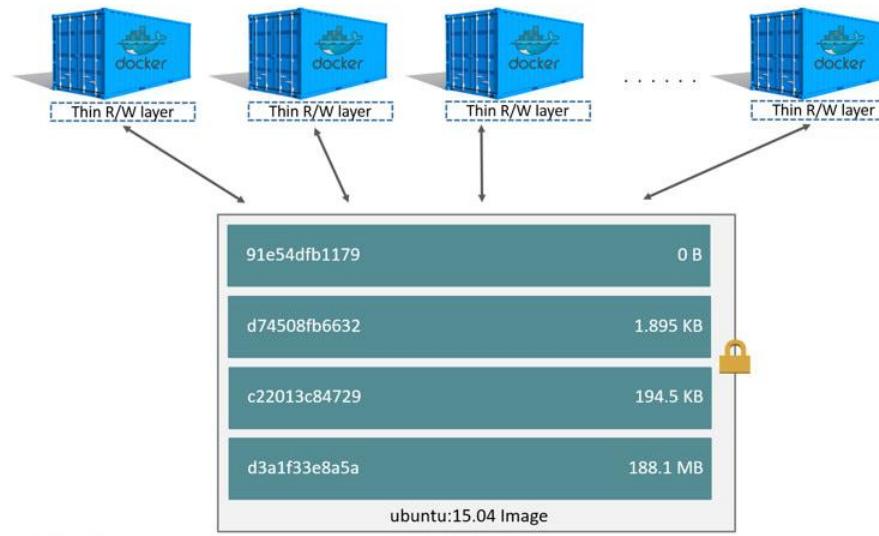
```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



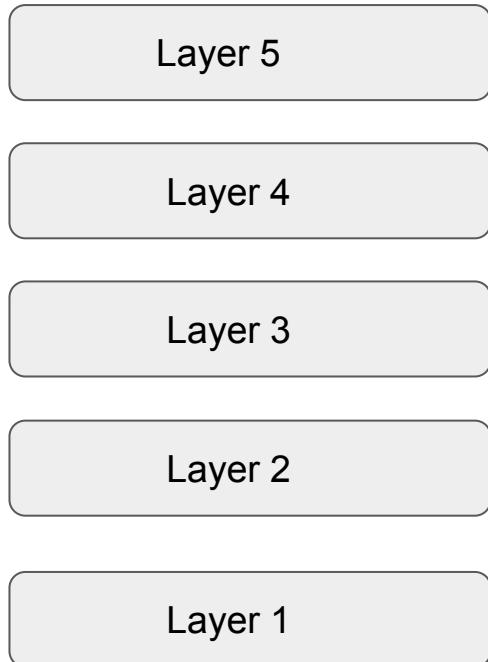
# Containers and Layers

The major difference between a container and an image is the top writable layer.

All writes to the container that add new or modify existing data are stored in this writable layer.



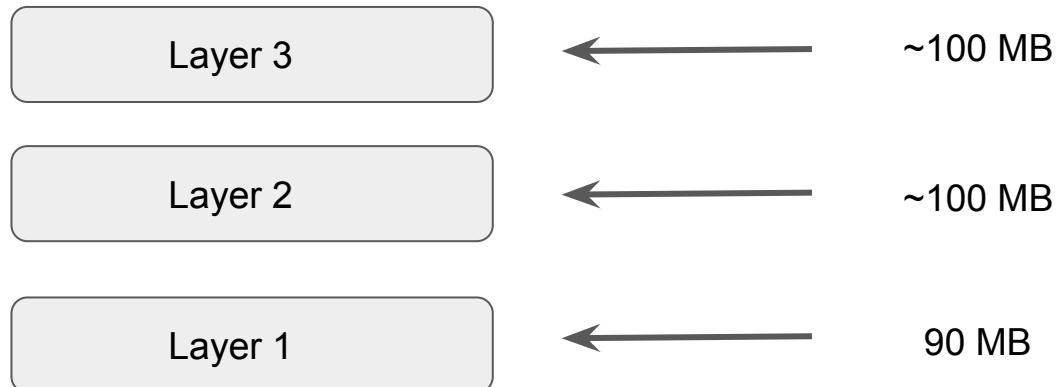
# Image and Layer



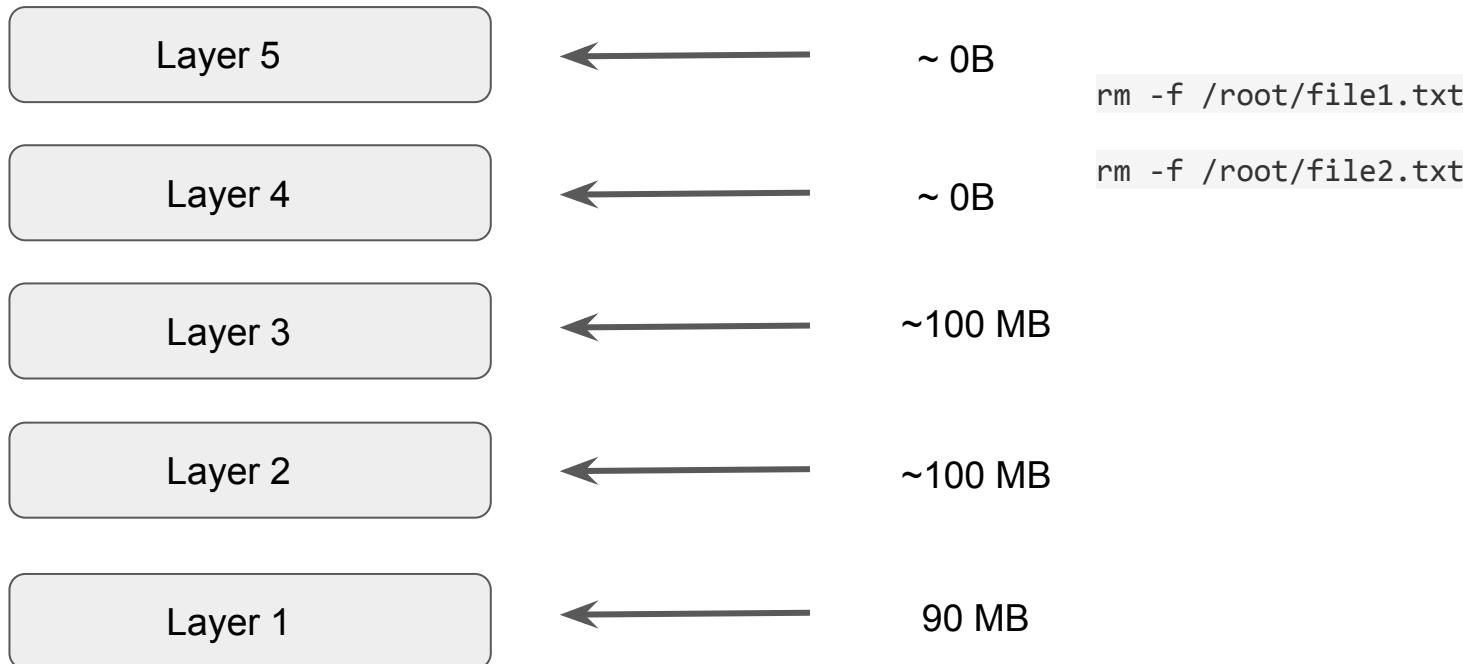
```
FROM ubuntu
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
RUN rm -f /root/file1.txt
RUN rm -f /root/file2.txt
```

# First Three Instructions

```
FROM ubuntu  
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100  
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
```



# ALL Instruction



# Point to Note

We have new updates coming up in next 3-4 weeks on the Exam Preparation Section.

---

# Important Pointers - Domain 1

Core Concepts

# Important Pointer - Part 1

Know very well on how to create POD manifest via CLI

Practice the below command very well:

```
kubectl run nginx --image=nginx --port=80 --restart=Never --dry-run -o yaml > pod.yaml
```

# Important Pointer - Part 2

Be aware that you cannot modify everything within the live object.

In situation were you have to modify certain aspects of POD object, then three steps are required:

<b>Steps</b>	<b>Description</b>	<b>Commands</b>
1	Store the object YAML and store it locally.	kubectl get pod [POD-NAME] -o yaml > pod.yaml
2	Delete the Live Object	kubectl delete pod [POD-NAME]
3	Modify Object manifest and apply	kubectl apply -f pod.yaml

---

# Important Pointers - Domain 2

POD Design

# Important Pointer - Part 1

Be very familiar with labels and selectors

Know on how you can apply to an object (both via manifest file as well as live modification)

Network Policies also work based on labels and selectors. For troubleshooting them, you need to be familiar with this topic.

<b>Useful Commands</b>	<b>Description</b>
<code>kubectl get pods --show-labels</code>	Show pods along with their labels.
<code>kubectl label pod busybox key=value</code>	Add a label to the pod.

# Important Pointer - Part 2

Be aware of deployments, creation and modification aspects.

Understand the importance of labels and selectors in deployments.

Have a glimpse about maxSurge and maxUnavailable in deployments and how to modify them.

Be very thorough with rolling updates and rollbacks.

# Important Pointer - Part 3

Be very familiar with creating Jobs and CronJobs.

Don't worry about setting Cron date/time, that is out of scope for exams.

Dedicate time to understand `activeDeadlineSeconds` parameter within CronJob.

---

# Important Pointers - Domain 4

Configuration

# Important Pointer - Part 1

Be very familiar with the basics of creating ConfigMaps.

Know on how you can mount ConfigMaps to container on specific path via volumes.

# Important Pointer - Part 2

Have a basic idea of Security Context.

Pay special attention to runAsUser, runAsGroup and fsGroup.

Do not worry about SELinux aspect as that is not part of the exams.

# Important Pointer - Part 3

Have good idea of “Resource Quotas”.

You should know about requests and limits and how to set them at POD level.

Know on what will happen if Nodes does not match the minimum request values.

# Important Pointer - Part 4

Having a good understanding of “Kubernetes Secrets” is important.

Know on how you can create secret, mount it as environment variables, and volumes.

Be also familiar with environment variables.

## Important Pointer - Part 5

You should be aware of how you can associate service accounts with existing deployments.

Know that each namespace has its own default service account.

---

# Important Pointers - Domain 5

Observability

# Important Pointer - Part 1

Understanding Liveness and Readiness Probes are very important for exams.

Know the difference between both of them and when you should use one.

Exam question will not mention “implement liveness/readiness” probe, instead a use-case would be given and you will have to know which one to implement.

Be aware of scenario on editing liveness/readiness probe of existing live POD object.

Be aware of troubleshooting questions related to this.

# Important Pointer - Part 2

You should be aware about container logging.

The kubectl logs command should be good enough for CKAD.

Also, be aware about how you can fetch the events associated with a given POD.

```
kubectl describe pod [POD-NAME]
```

```
kubectl get events -o wide --field-selector=involvedObject.name=nginx
```

# Important Pointer - Part 3

Metric Server is pre-installed in the exams, so no need to worry about that.

Be familiar on how you can see CPU/Memory usage of a specific pods and nodes.

- kubectl top pods
- kubectl top nodes

# Important Pointer - Part 4

There will be many debugging related questions in exams.

Be prepared with debugging aspects related to various areas like:

- Network Policies
- Services
- Liveness/Readiness probes.
- Deployments

---

# Important Pointers - Domain 6

State persistency

# Important Pointer - Part 1

Be familiar with PV and PVC.

For PV, pay careful attention to hostPath and EmptyDir

Exam question will not say that you need to use emptydir, instead it will give you a use-case and you will have to figure it out yourself.

For example, create a pod, attach a volume to it on the path of /mounts in such a way that volume should be deleted once pod is removed.