

# Certified Kubernetes Administrator



## Gaurav

RHCSA | RHCE | Red Hat Certified Specialist in OpenShift Administration | CKA | CKAD | SCA | SCE | SCA+ HA  
| ANSIBLE | DOCKER | OPENSHIFT | OPEN SOURCE TECHNOLOGIES

# Course Agenda

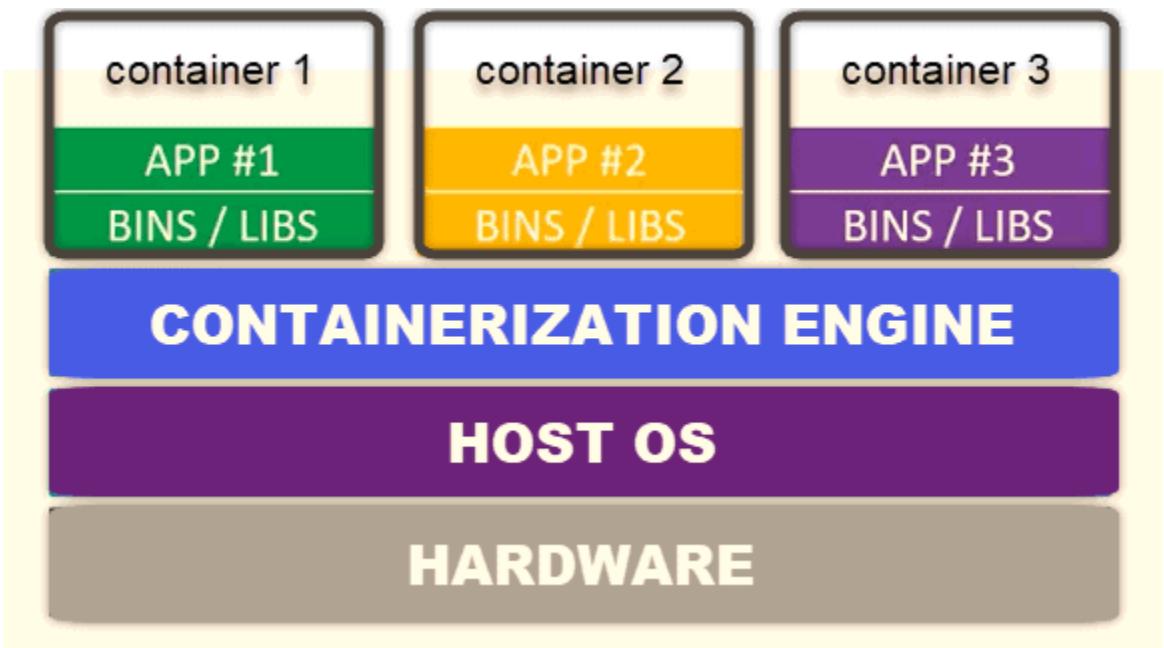
- Core Concepts
- Installation, Configuration and Validation
- Managing Resources
- Manual Scheduling
- Application Lifecycle Management
- Storage & Container Networking
- Environment Variable
- Storage

# Course Agenda

- Security
- Networking
- Cluster Maintenance
- Logging and Monitoring
- Troubleshooting

# What is Container?

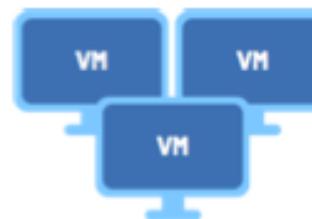
A container is an environment that runs an application that is not dependent on the operating system. It isolates the app from the host by virtualizing it. This allows users to created multiple workloads on a single OS instance.



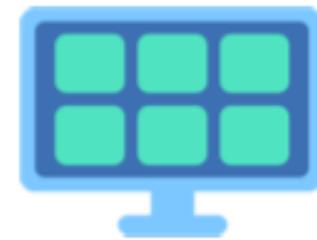
# Bare Metal vs Container vs Virtual Machine



Bare Metal



Virtual machines



Containers

Code

App Container

Language Runtime

Operating System

Hardware

Code

App Container

Language Runtime

Operating System

Hardware

Code

App Container

Language Runtime

Operating System

Hardware

# Why Container?

- Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images.
- Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.

# Container Orchestration

# Scenario of Guard

- You are owner of a building and you have 5 spots where people can enter your building and you want 5 security guards guarding the spots. All good till now.
- Now consider one of the guard was out of service for 2 hours due to some personal reasons. Now as a building owner its your responsibility to guard or employ another guard replacing the existing. Do you like to be manually interrupted from your task to look after who is out and whom to replace.
- No, no one likes to be. Now the solution could be, go to a third party vendor who provides 24\*7 availability of the guards. Its the responsibility of the vendor to make 24\*7 availability based on the requirements.

# What Problem Does Kubernetes Solve?

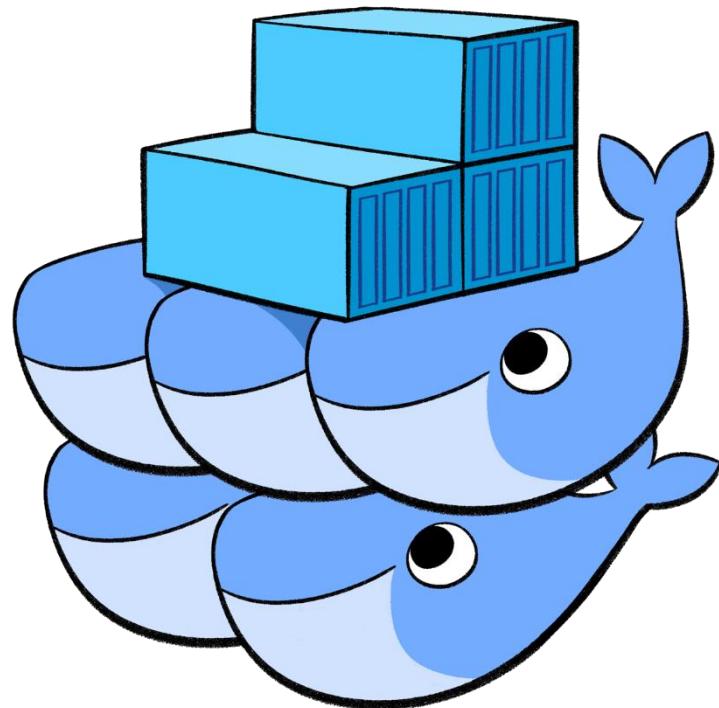
Now let's replace the characters.

- Building -> Your application
- Owner -> The application owner
- Guards -> Containers
- Vendor -> Kubernetes
- Configuration (replica)-> That we make a deal with vendor to maintain appropriate conf 24\*7.

# What is Container Orchestration?

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux containers so in this case we don't want downtime so container orchestration will give this facility.

# Container Orchestration Tools



**vs**



# Installation and Cluster Configuration

## DOCKER SWARM

- Setting up the cluster is simple
- Requires only 2 commands

## KUBERNETES

- Setting up the cluster is challenging and complicated

# User Interface

## DOCKER SWARM

- There is no GUI available
- Requires a third party tool

## KUBERNETES

- Provide a GUI  
(KUBERNETES DASHBOARD)

# Scalability

## DOCKER SWARM

- Quick container deployment and scaling even in very large cluster
- Scaling up is 5X faster than Kubernetes
- Autoscaling is not possible

## KUBERNETES

- Provides strong guarantees to cluster states at the expense of speed
- Scaling up is easy
- Supports auto-scaling

# Load Balancing

## DOCKER SWARM

- Automated internal loadbalancing through any node in the cluster

## KUBERNETES

- Enabling load balancing requires manual service configuration

# Data Volumes

## DOCKER SWARM

- Simple shared local volumes
- Storage volumes can be shared with any other container in the node.

## KUBERNETES

- Volumes shared within pods
- Storage volumes can be shared only between containers within the same pod.

# Logging and Monitoring

## DOCKER SWARM

- 3<sup>rd</sup> party logging and monitoring tools required

## KUBERNETES

- In-built logging and monitoring tools in place

# Rolling Updates and Rollbacks

## DOCKER SWARM

- Automated rollbacks is not available.

## KUBERNETES

- Automated rollbacks in case of a failure.

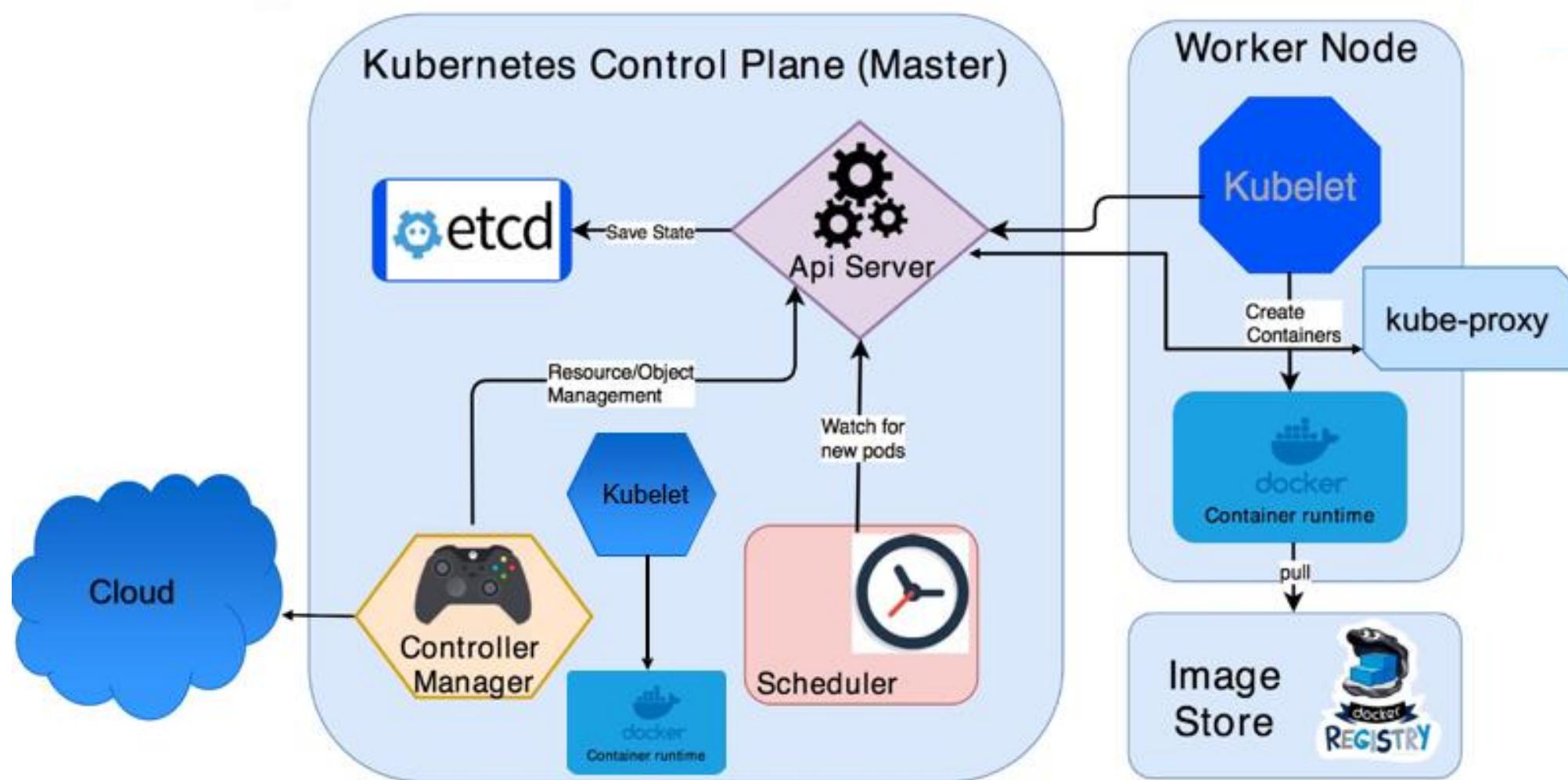
# What is Kubernetes?

Kubernetes is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing and scaling containerized applications.

# About Kubernetes

- Open-source cluster management tool
- Managed by Cloud Native Computing Foundation (CNCF); originally created by Google
- Under active deployment by a well-supported community.
- Can run on both bare metal and on various cloud providers.

# Kubernetes Architecture



# Kube-Apiserver

The Kubernetes API is the front end of the Kubernetes control plane, handling internal and external requests. The API server determines if a request is valid and, if it is then processes it.

## **Kube Apiserver work:**

- 1) Authenticate User
- 2) Validate Request
- 3) Retrieve Data
- 4) Update ETCD
- 5) Scheduler
- 6) Kubelet

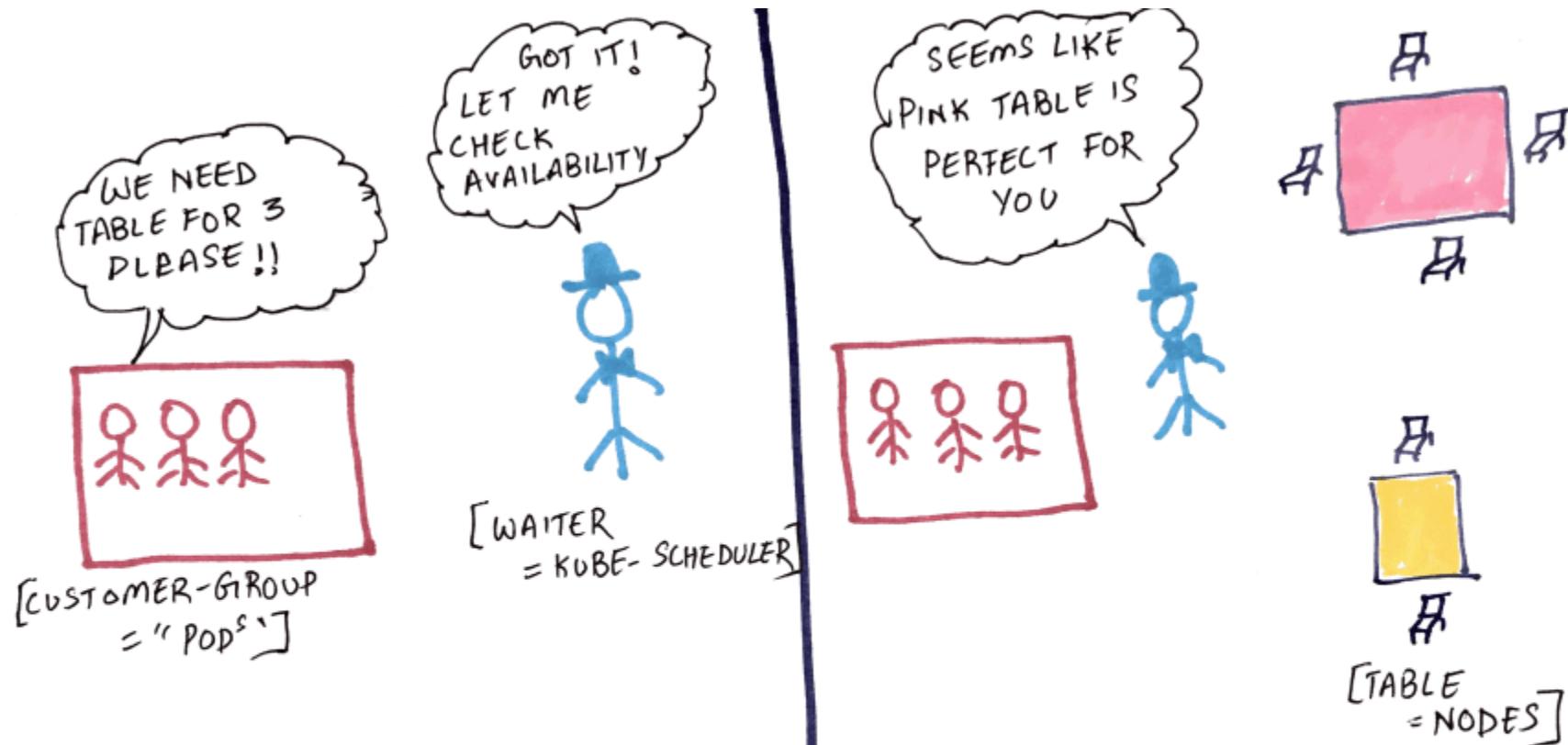
# ETCD

Configuration data and information about the state of the cluster lives in etcd, a key-value store database. Fault-tolerant and distributed, etcd is designed to be the ultimate source of truth about your cluster.

# Kube-Scheduler

The scheduler considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster. Then it schedules the pod to an appropriate compute node.

# How Kube-Scheduler Works?



# Kube-Controller-Manager

Controllers take care of actually running the cluster, and the Kubernetes controller-manager contains several controller functions in one. One controller consults the scheduler and makes sure the correct number of pods is running. If a pod goes down, another controller notices and responds. A controller connects services to pods, so requests go to the right endpoints.

# Controllers

- 1) Deployment Controller
- 2) CronJob Controller
- 3) Service Account Controller
- 4) Node Controller
- 5) Namespace Controller
- 6) Job Controller
- 7) Statefulset Controller
- 8) PV-Binder-Controller
- 9) Endpoint-Controller
- 10) ReplicaSet
- 11) Replication-Controller

# Container Runtime

To run the containers, each compute node has a container runtime engine. Docker is one example, but Kubernetes supports other Open Container Initiative-compliant runtimes as well, such as rkt and CRI-O.

# Kubelet

Each compute node contains a kubelet, a tiny application that communicates with the control plane. The kubelet makes sure containers are running in a pod. When the control plane needs something to happen in a node, the kubelet executes the action.

# Kube-Proxy

Each compute node also contains kube-proxy, a network proxy for facilitating Kubernetes networking services. The kube-proxy handles network communications inside or outside of your cluster—relying either on your operating system's packet filtering layer, or forwarding the traffic itself.

# Kubernetes Installation

# OS Requirements

One or more machines running one of:

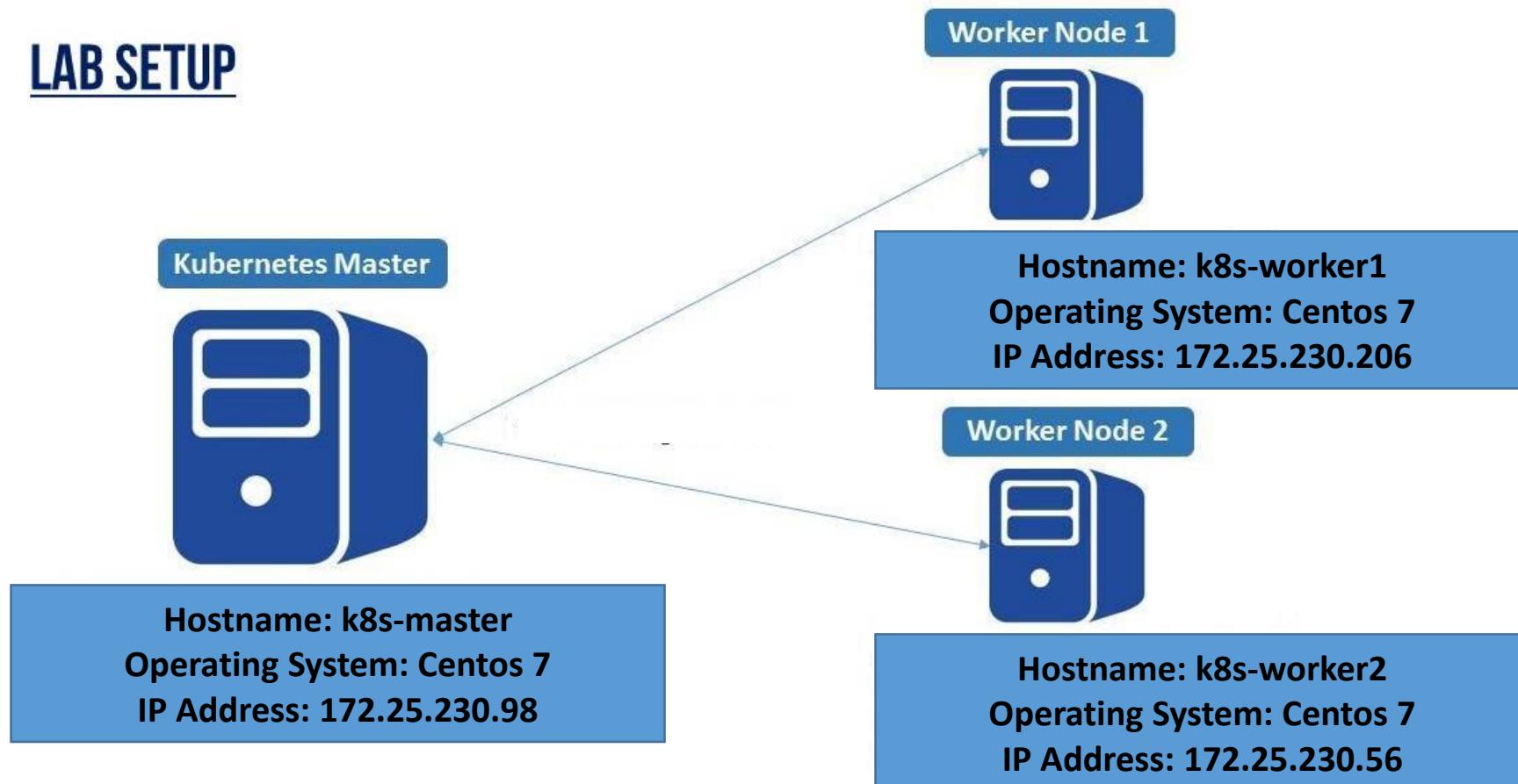
- Ubuntu 16.04+
- Debian 9+
- CentOS 7+
- Red Hat Enterprise Linux (RHEL) 7+
- Fedora 25+
- HypriotOS v1.0.1+
- Flatcar Container Linux (tested with 2512.3.0)

# Hardware Requirement

- 2 GB or more of RAM per machine
- 2 CPUs or more.

# Lab Environment

## LAB SETUP



# Prerequisites

- Set Hostname  
**#hostnamectl set-hostname <name>**
- Configure Local DNS
- Disable SWAP
- Disable SELINUX
- Add certain ports in the firewalld or disable firewalld
- Configure sysctl

# Configure Local DNS (On All Nodes)

```
[root@k8s-master ~]# cat /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
172.25.230.98 k8s-master
172.25.230.206 k8s-worker1
172.25.230.56 k8s-worker2
```

# Disable SWAP (On All Nodes)

```
[root@k8s-master ~]# cat /etc/fstab | grep swap  
#/dev/mapper/centos-swap swap          swap      defaults      0 0  
[root@k8s-master ~]# swapoff -a
```

# Disable SELINUX (On All Nodes)

```
[root@k8s-master ~]# cat /etc/sysconfig/selinux | grep SELINUX=permissive  
SELINUX=permissive  
[root@k8s-master ~]# setenforce 0  
[root@k8s-master ~]# getenforce  
Permissive
```

# Components Port Numbers

## Control-plane node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self

## Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services	All

# Disable Firewalld (On All Nodes)

```
[root@k8s-master ~]# systemctl stop firewalld
[root@k8s-master ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

# Configure sysctl(On All Nodes)

```
[root@k8s-master ~]# modprobe overlay
[root@k8s-master ~]# modprobe br_netfilter
[root@k8s-master ~]# cat /etc/sysctl.d/kubernetes.conf
net.bridge.bridge-nf-call-ip6tables=1
net.bridge.bridge-nf-call-iptables=1
net.ipv4.ip_forward=1
[root@k8s-master ~]# sysctl --system
```

# Configure Kubernetes Repository (On All Nodes)

```
#vim /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
name=kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
```

# Kubernetes Utilities

- `kubeadm` : the command to bootstrap the cluster.
- `kubelet` : the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- `kubectl` : the command line util to talk to your cluster.

# Install Kubernetes Packages (On All Nodes)

```
[root@k8s-master ~]# #yum install kubectl kubelet kubeadm -y
[root@k8s-master ~]# systemctl start kubelet
[root@k8s-master ~]# systemctl enable kubelet
[root@k8s-master ~]# kubectl version --short
Client Version: v1.20.2
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[root@k8s-master ~]# █
```

# Install Container Runtime (On All Nodes)

```
[root@k8s-master ~]# #curl -L -o /etc/yum.repos.d/libcontainers-stable.repo https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/CentOS_7-devel:kubic:libcontainers:stable.repo  
[root@k8s-master ~]#  
[root@k8s-master ~]# #curl -L -o /etc/yum.repos.d/crio.repo https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:1.20/CentOS_7-devel:kubic:libcontainers:stable:cri-o:1.20.repo
```

---

```
[root@k8s-master ~]# #yum install cri-o -y  
[root@k8s-master ~]# #systemctl start crio  
[root@k8s-master ~]# #systemctl enable crio
```

# Configure Kubelet with systemd (On All Nodes)

---

```
[root@k8s-master ~]# cat /etc/sysconfig/kubelet
KUBELET_EXTRA_ARGS="--cgroup-driver=systemd"
[root@k8s-master ~]# systemctl restart kubelet
```

# Initialize Kubernetes Cluster

**To Initialize Control-Plane (On Master) :**

```
#kubeadm init --pod-network-cidr=10.244.0.0/16
```

**To Configure User (On Master):**

```
#export KUBECONFIG=/etc/kubernetes/admin.conf
```

**To join worker nodes:**

**COPY JOIN COMMAND AND PASTE ON WORKER NODES**

# Install Calico Network Solution (On Master)

**If 50 nodes or less:**

```
#kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

**If more than 50 nodes:**

```
#kubectl apply -f https://docs.projectcalico.org/manifests/calico-typha.yaml
```

Doc: <https://docs.projectcalico.org/getting-started/kubernetes/self-managed-onprem/onpremises>

# Verify Installation

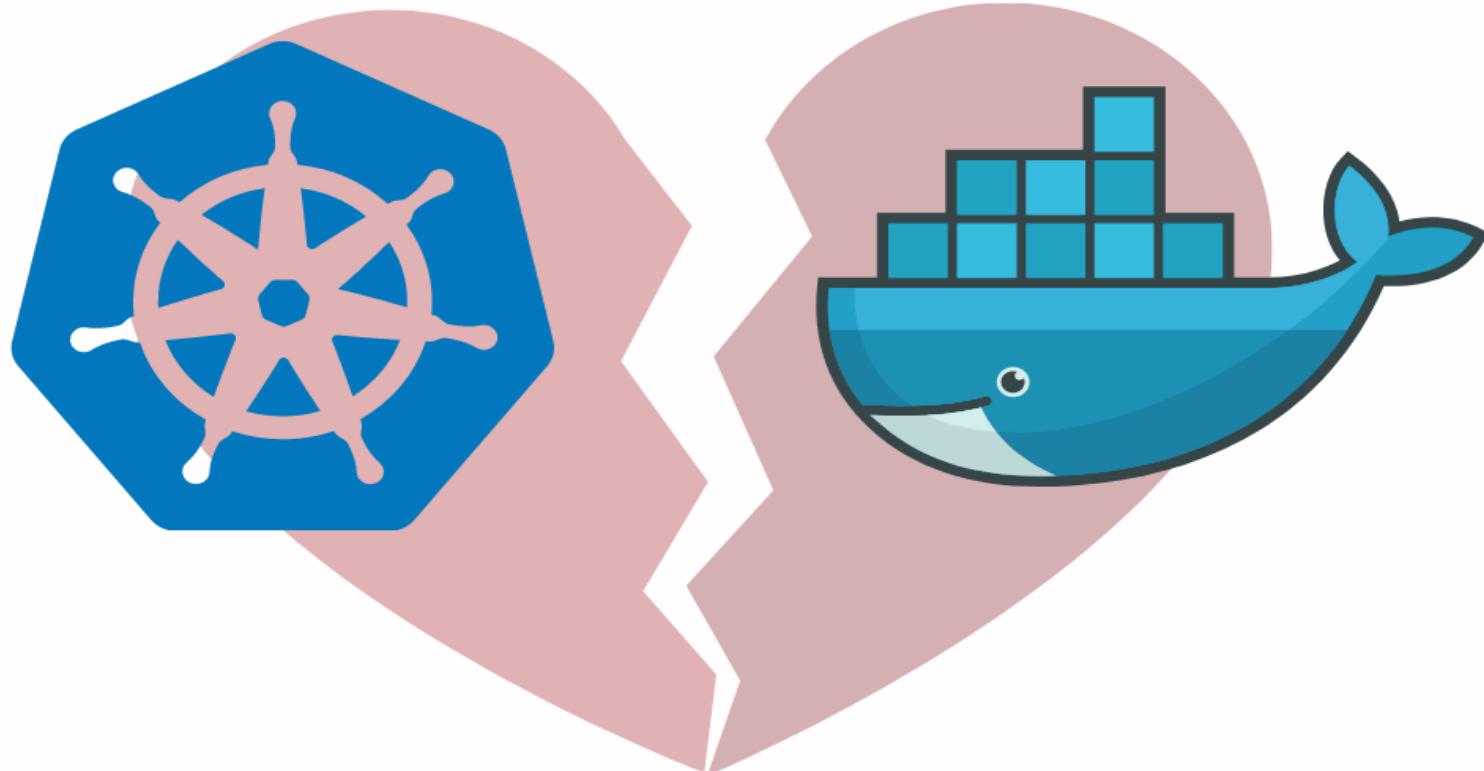
```
[root@k8s-master ~]# kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-86bddfcff-spcd4	1/1	Running	0	3m55s
kube-system	calico-node-2gjdf	1/1	Running	0	3m55s
kube-system	calico-node-nc86b	1/1	Running	0	3m55s
kube-system	calico-node-tq4l5	1/1	Running	0	3m55s
kube-system	coredns-74ff55c5b-xg87g	1/1	Running	0	15m
kube-system	coredns-74ff55c5b-xt5qq	1/1	Running	0	15m
kube-system	etcd-k8s-master	1/1	Running	0	15m
kube-system	kube-apiserver-k8s-master	1/1	Running	0	15m
kube-system	kube-controller-manager-k8s-master	1/1	Running	0	15m
kube-system	kube-proxy-kjpms	1/1	Running	0	15m
kube-system	kube-proxy-x2pxt	1/1	Running	0	12m
kube-system	kube-proxy-xp7fq	1/1	Running	0	11m
kube-system	kube-scheduler-k8s-master	1/1	Running	0	15m

# List Nodes

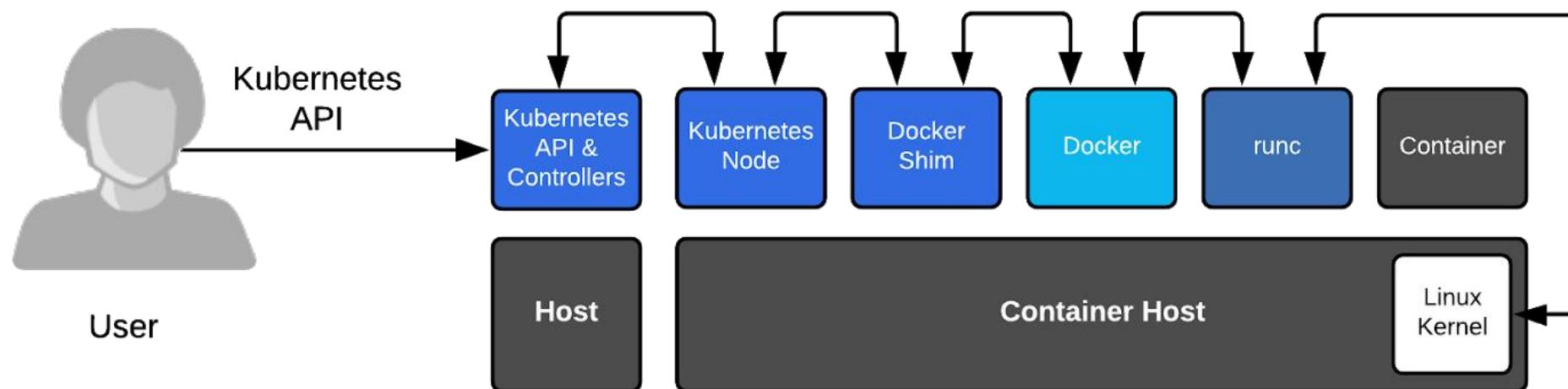
```
[root@k8s-master ~]# kubectl get nodes
NAME        STATUS   ROLES          AGE    VERSION
k8s-master   Ready    control-plane,master   16m    v1.20.2
k8s-worker1  Ready    <none>         12m    v1.20.2
k8s-worker2  Ready    <none>         12m    v1.20.2
```

# Kubernetes is Deprecating Docker

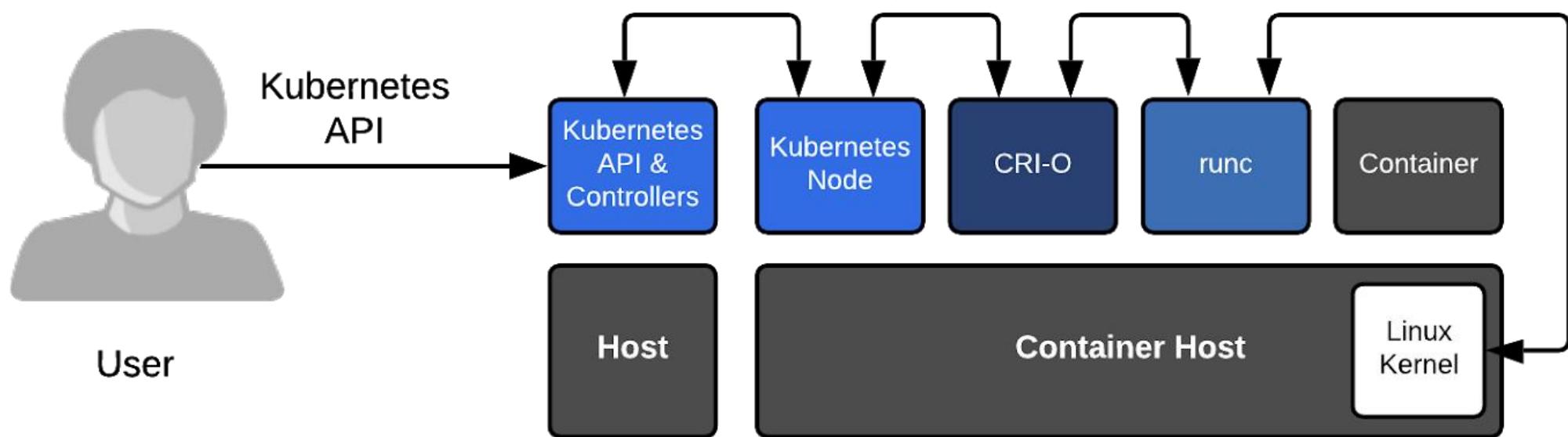


# Kubernetes Architecture with Docker

**Kubernetes only talks in CRI and talking to Docker requires a bridge service.**



# Kubernetes Architecture with CRI-O



# Why is Dockershim being deprecated?

Maintaining dockershim has become a heavy burden on the Kubernetes maintainers. The CRI standard was created to reduce this burden and allow smooth interoperability of different container runtimes. Docker itself doesn't currently implement CRI, thus the problem.

# WHY CRI-O?



# cri-o

LIGHTWEIGHT CONTAINER RUNTIME FOR KUBERNETES



## DESIGNED

Optimized for Kubernetes



## STABLE

Committed to passing Kubernetes tests



## ANY IMAGE, ANY REGISTRY

Pull from *any* compliant registry; run *any* OCI-compliant container

# Bash Completion

```
#yum install bash-completion -y  
#cd ~/.kube  
#kubectl completion bash > kube.sh  
#source kube.sh
```

To make it permanent:-

```
#vim $HOME/.bashrc
```

Add a entry

```
source $HOME/.kube/kube.sh
```

# Nodes

A node is a worker machine in Kubernetes, previously known as a minion. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components.

# Node Details

```
#kubectl describe node <node name>
```

**A node's status contains the following information:**

- IP Addresses
- Conditions
- Capacity and Allocation
- Information

# Node Conditions

Ready

`True` if the node is healthy and ready to accept pods, `False` if the node is not healthy and is not accepting pods, and `Unknown` if the node controller has not heard from the node in the last `node-monitor-grace-period` (default is 40 seconds)

DiskPressure

`True` if pressure exists on the disk size--that is, if the disk capacity is low; otherwise `False`

MemoryPressure

`True` if pressure exists on the node memory--that is, if the node memory is low; otherwise `False`

PIDPressure

`True` if pressure exists on the processes—that is, if there are too many processes on the node; otherwise `False`

NetworkUnavailable

`True` if the network for the node is not correctly configured, otherwise `False`

# Kubectl

```
#kubectl run <name> --image=<name>  
#kubectl cluster-info  
#kubectl get nodes  
#kubectl describe node <insert-node-name-here>
```

# Pods

A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it. For example, keeping a database container and data container in the same pod.

**There are two types of Pods:**

- Single container pod
- Multi container pod

# Pod Characteristics

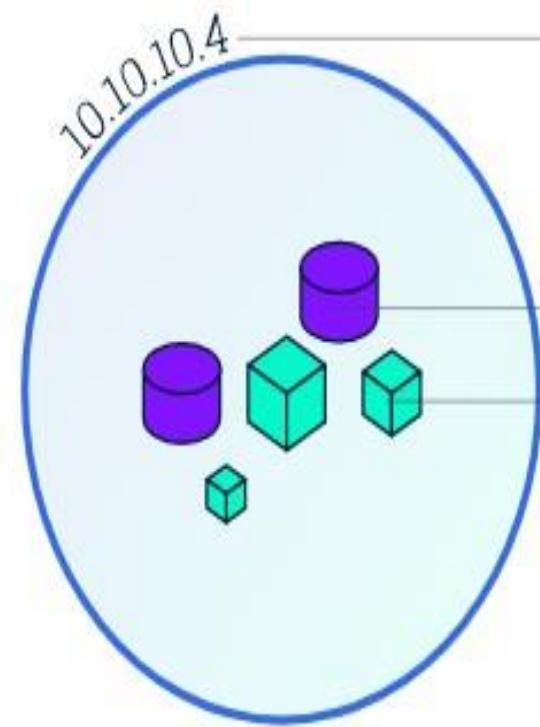
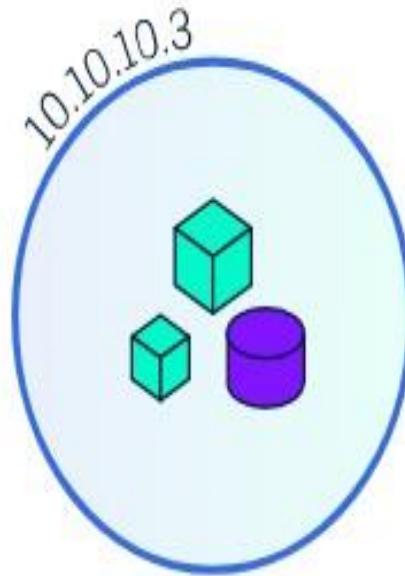
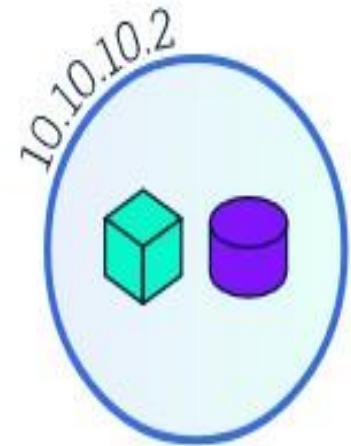
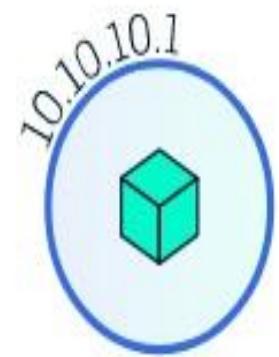
Pods provide two kinds of shared resources for their constituent containers: networking and storage.

- 1)Networking->** Each Pod is assigned a unique IP address. Every container in a Pod shares the network namespace, including the IP address and network ports.
- 2)Storage ->** A Pod can specify a set of shared storage Volumes. All containers in the Pod can access the shared volumes, allowing those containers to share data.

# Pod Characteristics

Pods provide two kinds of shared resources for their constituent containers: networking and storage.

- 1)Networking->** Each Pod is assigned a unique IP address. Every container in a Pod shares the network namespace, including the IP address and network ports.
- 2)Storage ->** A Pod can specify a set of shared storage Volumes. All containers in the Pod can access the shared volumes, allowing those containers to share data.



Pod 1

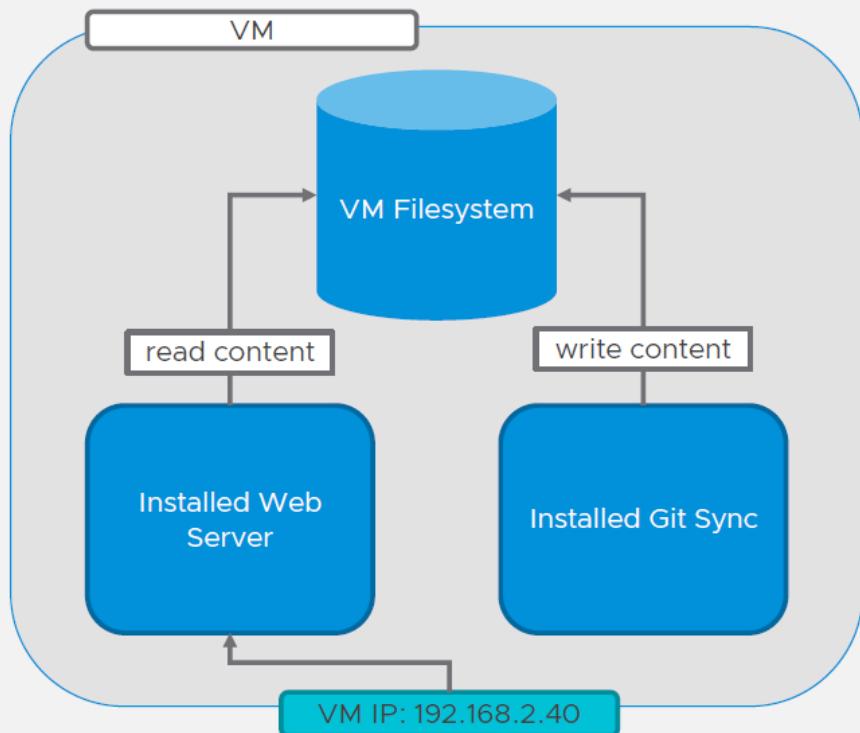
Pod 2

Pod 3

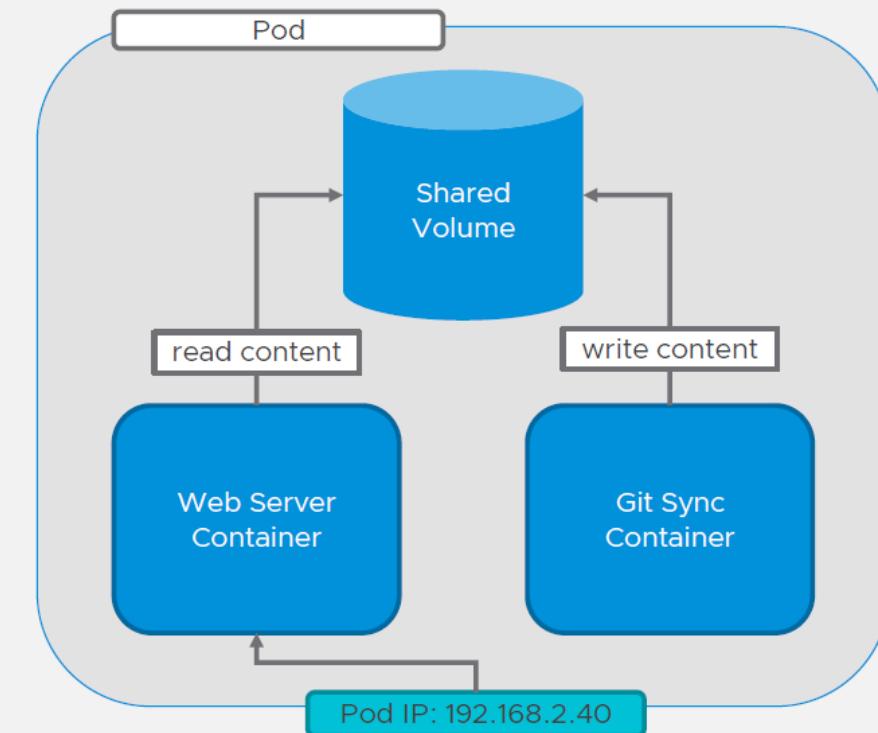
Pod 4

# Similar Concepts

## Virtualization



## Kubernetes



# Managing Pods with Commands

```
#kubectl run <podname> --image=nginx  
#kubectl get pods  
#kubectl get pods -A  
#kubectl get pods -o wide  
#kubectl describe pods <podname>  
#kubectl explain <resource> like pod,deployment  
#kubectl exec -it <podname> -- bash
```

# YAML Introduction

**Four fields are required to create yaml file:**

1. **apiVersion** - Which version of the Kubernetes API you're using to create this object
2. **kind** - What kind of object you want to create
3. **metadata** - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
4. **spec** - What state you desire for the object

# API Version

apiVersion: version of API depend on the resource.

Kind	Version
Pod	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

**To check API Version:**

```
#kubectl explain <resource name>
```

# Create Pod with YAML

```
#vim pod.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-first-pod
spec:
  containers:
    - name: mycontainer
      image: quay.io/app-sre/nginx
      ports:
        - containerPort: 80
```

```
#kubectl create -f pod.yml
#kubectl delete -f pod.yml
```

# Image Pull Policy

**Three types of Image Pull Policy:**

- 1) Always**
- 2) IfNotPresent**
- 3) Never**

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: abc
    image: quay.io/gauravkumar9130/nginxdemo
    imagePullPolicy: IfNotPresent
```

# Labels and Selectors

**Labels** : Labels enable end users to map their own, custom, organizational structures onto system resources in a loosely coupled fashion.

**Selectors** : Used to filter the tags

# Example Of Label

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: development
spec:
  containers:
    - name: mycontainer
      image: quay.io/app-sre/nginx
      ports:
        - containerPort: 80
```

# Labels Commands

**To list labels:**

```
#kubectl get pods <podname>--show-labels
```

**To set label:**

```
#kubectl label pod <podname> <key>=<value>
```

**To delete label:**

```
#kubectl label pod <podname> <key>-
```

**To overwrite label:**

```
#kubectl label --overwrite pods <podname> <key>=<value>
```

# Selector

Selector is used to filter the labels.

## Two types of Selectors:

- Equity-Based Selector
- Set-Based Selector

---

```
[root@master kube]# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
mypod     1/1     Running   0          7m56s  env=development
mypod2    1/1     Running   0          7m35s  env=production
mypod3    1/1     Running   0          7m2s   env=marketting
[root@master kube]#
```

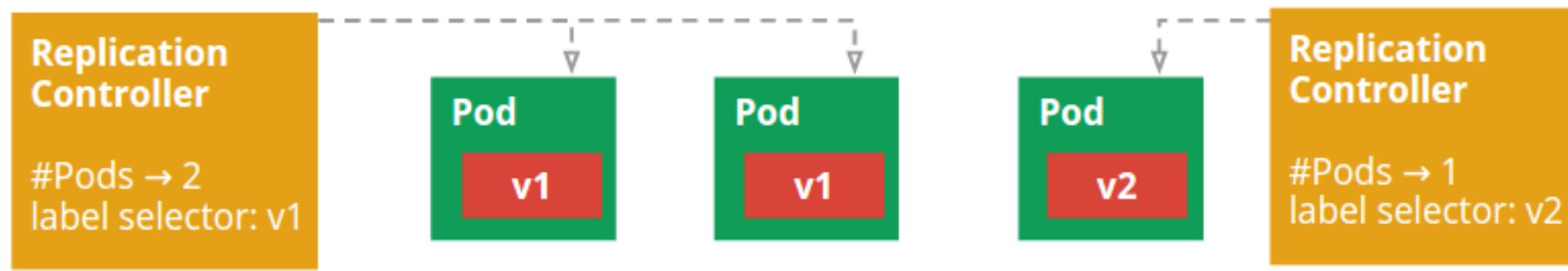
# Equality-Based Selector

```
[root@master kube]# kubectl get pods --selector env=marketting
NAME      READY   STATUS    RESTARTS   AGE
mypad3   1/1     Running   0          83s
[root@master kube]# kubectl get pods --selector env!=marketting
NAME      READY   STATUS    RESTARTS   AGE
mypad    1/1     Running   0          2m29s
mypad2   1/1     Running   0          2m8s
[root@master kube]# █
```

# Set-Based Selector

```
[root@master kube]# kubectl get pods --selector 'env in (development,marketing)'  
NAME      READY   STATUS    RESTARTS   AGE  
mypod     1/1     Running   0          3m42s  
mypod3    1/1     Running   0          2m48s  
[root@master kube]# █
```

# Replication Controllers



## Behavior

- Keeps Pods running
- Gives direct control of Pod #s

## Benefits

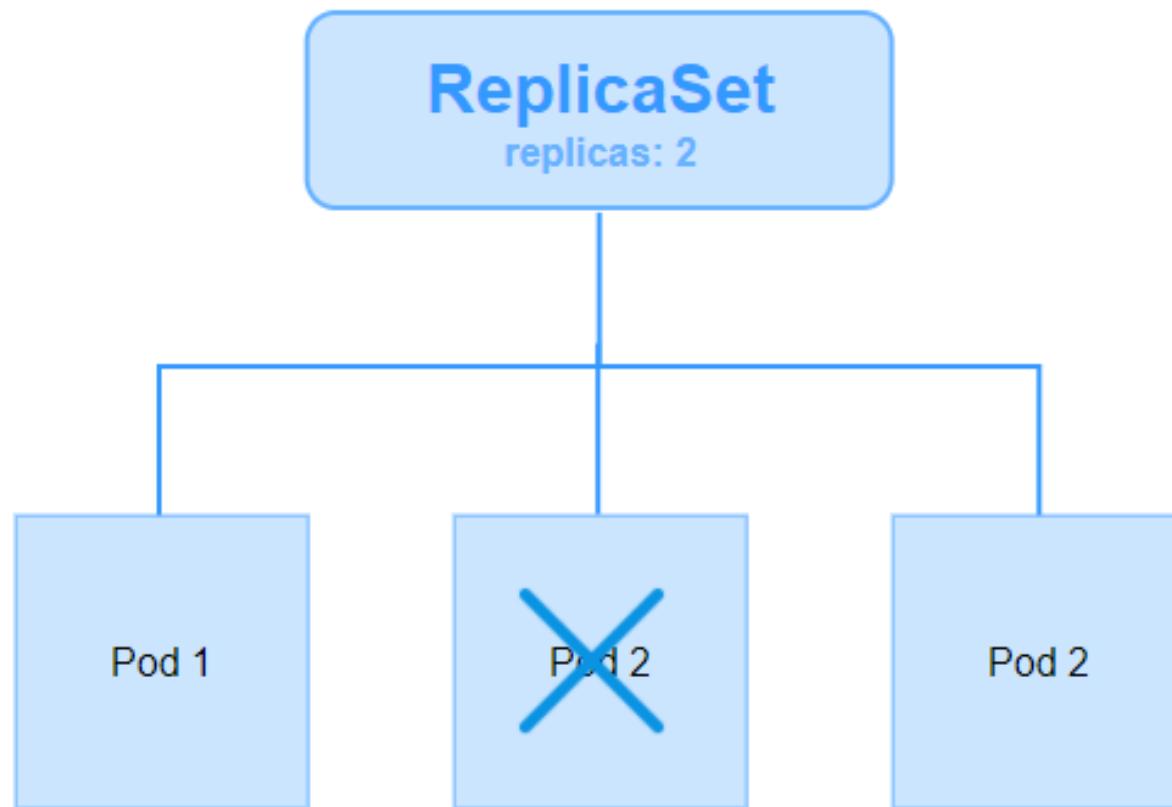
- Restarts Pods, desired state
- Fine-grained control for scaling

# Demo of Replication Controller

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: rc-web
spec:
  replicas: 3
  template:
    metadata:
      name: web
      labels:
        app: nginx
    spec:
      containers:
      - name: web
        image: quay.io/app-sre/nginx
```

**#kubectl get replicationcontroller**  
**OR**  
**#kubectl get rc**

# Replica Set



# Demo of Replica Set

**EXAMPLE 1**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: webserver
      labels:
        app: nginx
    spec:
      containers:
        - name: webcontainer
          image: quay.io/app-sre/nginx
      ports:
        - containerPort: 80
```

**EXAMPLE 2**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-web
spec:
  replicas: 3
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - app1
          - app2
  template:
    metadata:
      name: webserver
      labels:
        app: app1
    spec:
      containers:
        - name: webcontainer
          image: quay.io/app-sre/nginx
```

# Replication Controller VS Replica Set

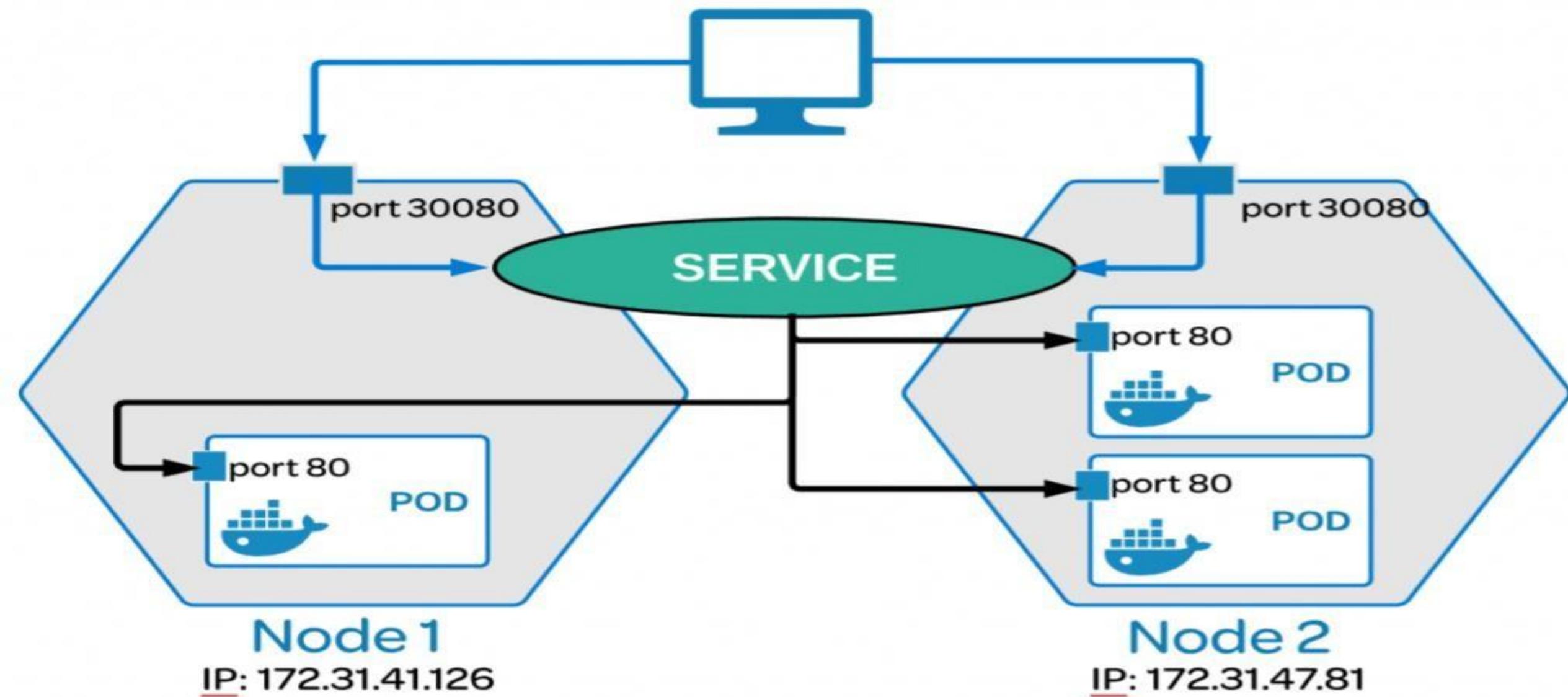
- Selector is optional in Replication Controller, but it's mandatory in Replica Set.
- Set-Based selector is not supported by Replication Controller, but it's supported by Replica Set.

# Service

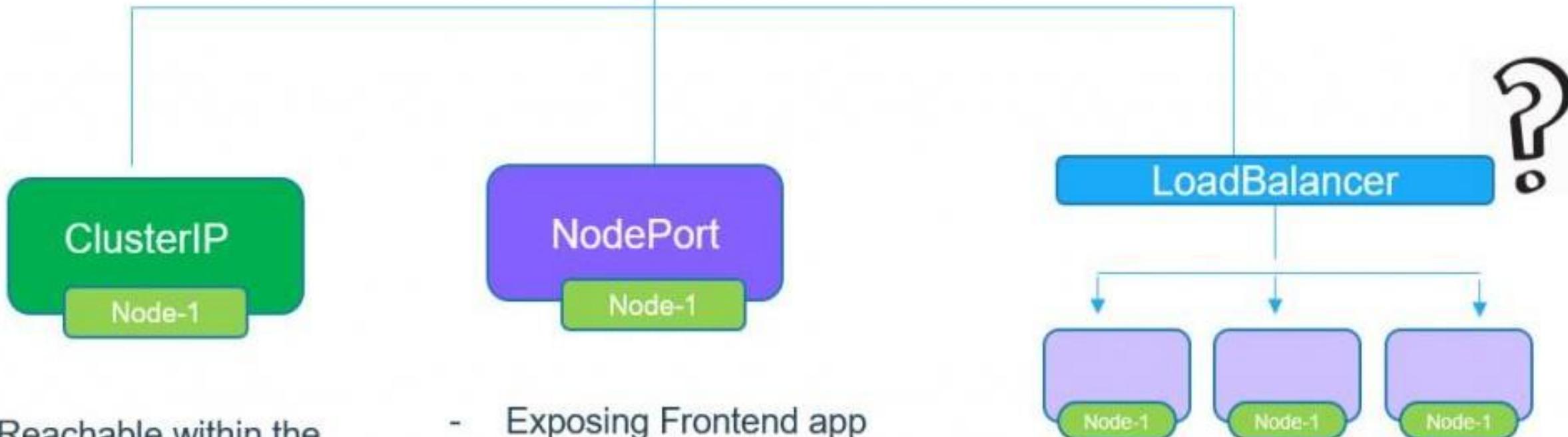
- Services are resources in Kubernetes (like pods, deployments, etc.) that provide a single point of access from the outside world, into your pod(s) which run your application.
- Each service has an IP address and a port that never change, and the client always knows where to find the pods due to the flat nat-less network.

# Kubernetes Service

A service allows you to dynamically access a group of replica pods.



## Types of Services

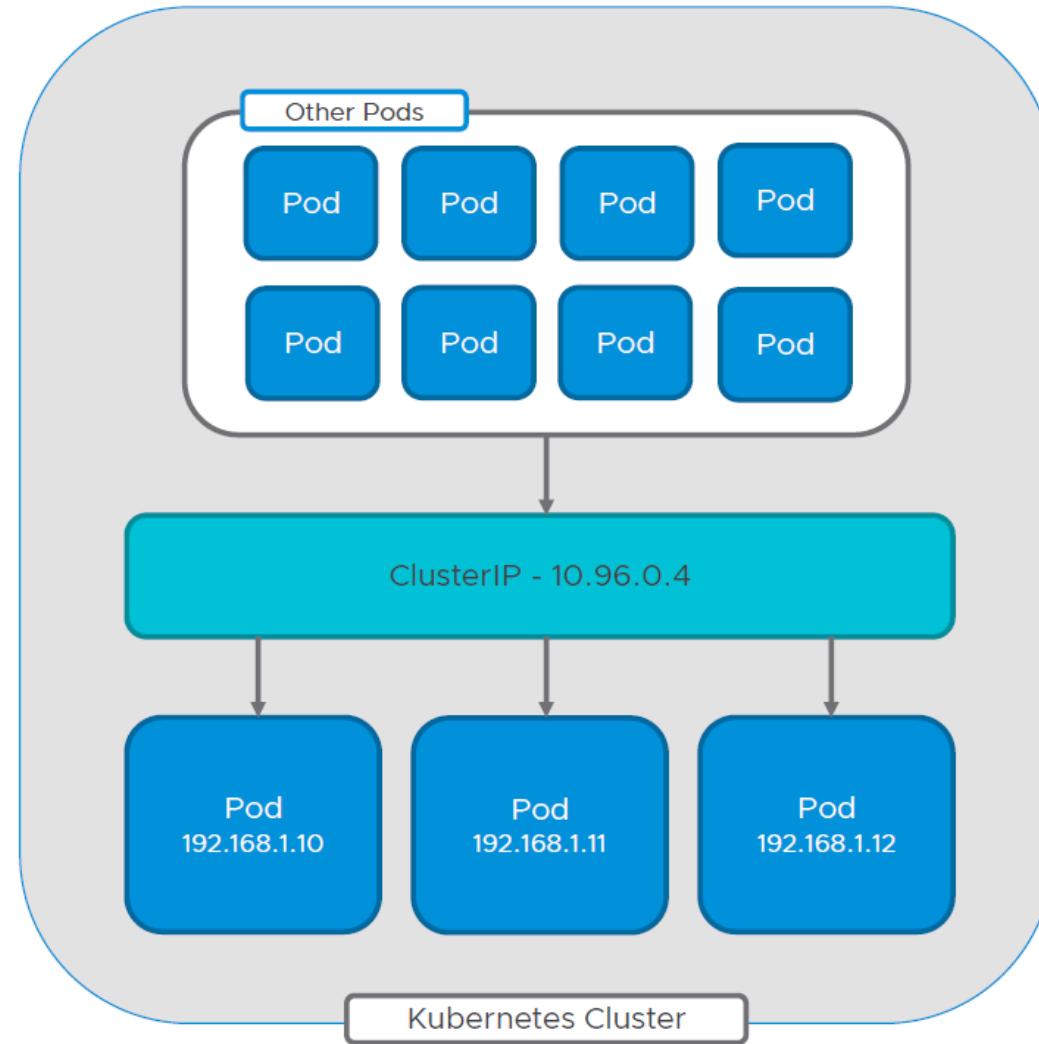


- Reachable within the cluster.
- Connects Frontend Pods to Backend Pods
- Exposing Frontend app to external world
- Equally distribute the loads

# Cluster IP

- ClusterIP:
  - Used for internal-facing services
  - Implementation
    - a virtual IP address that load balances requests to a set of backend pods
    - accessible anywhere within the cluster
    - not externally accessible

# Cluster IP



# Deploy Application

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      name: webserver
      labels:
        app: web
    spec:
      containers:
        -
          name: webcontainer
          image: quay.io/gauravkumar9130/nginxdemo
      ports:
        -
          containerPort: 80
```

# Demo of Cluster IP Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-web-service
spec:
  type: ClusterIP
  ports:
    - targetPort: 80   #container port number
      port: 8081       #bind port number with cluster ip
  selector:
    app: web
```

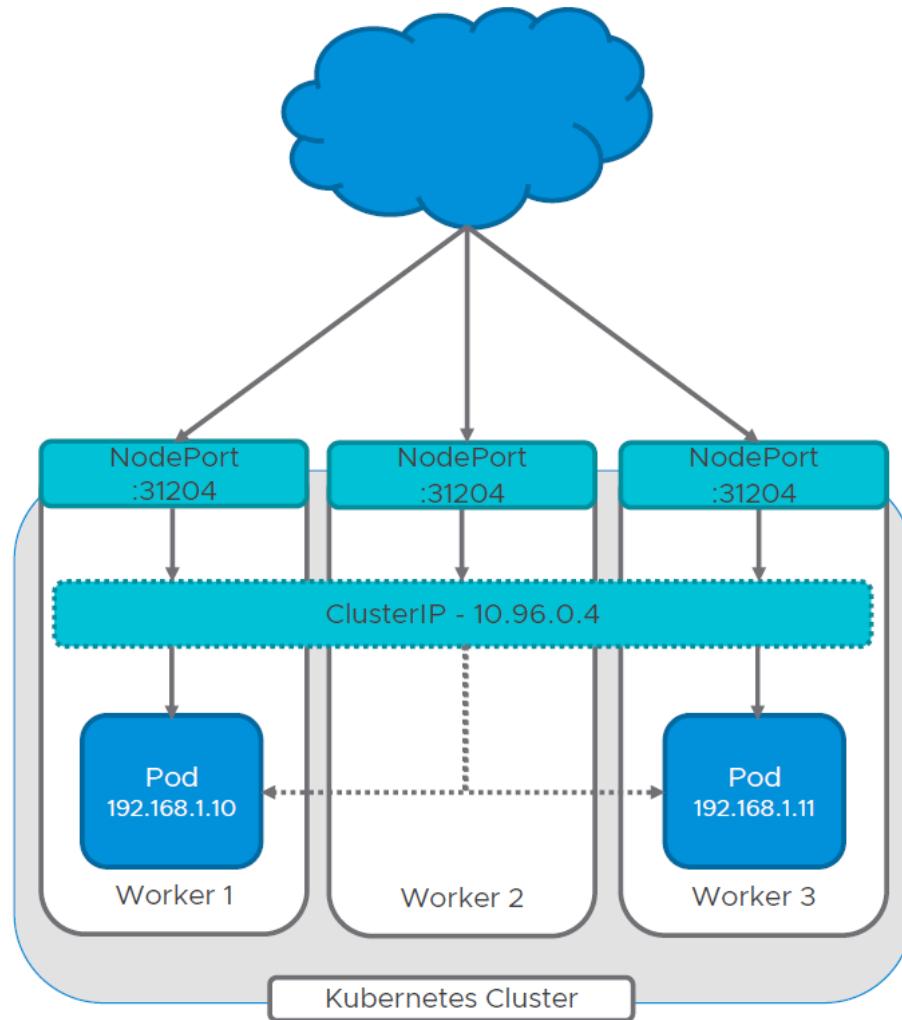
# Access Application

```
[root@master ~]# kubectl get svc
NAME           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes     ClusterIP  10.96.0.1      <none>          443/TCP      56m
my-web-service ClusterIP  10.111.137.181  <none>          8081/TCP    70s
[root@master ~]# curl 10.111.137.181:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

# Node Port

- NodePort:
  - Used for external facing services
  - Implementation
    - exposes a port on each node
    - externally accessible
    - leverages ClusterIP for load balancing to pods
  - You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

# Node Port



# Deploy Application

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      name: webserver
      labels:
        app: web
    spec:
      containers:
        -
          name: webcontainer
          image: quay.io/gauravkumar9130/nginxdemo
      ports:
        -
          containerPort: 80
```

# Demo of Node Port Service

```
apiVersion: v1
kind: Service
metadata:
  name: web-outside-service
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30008
  selector:
    app: web
```

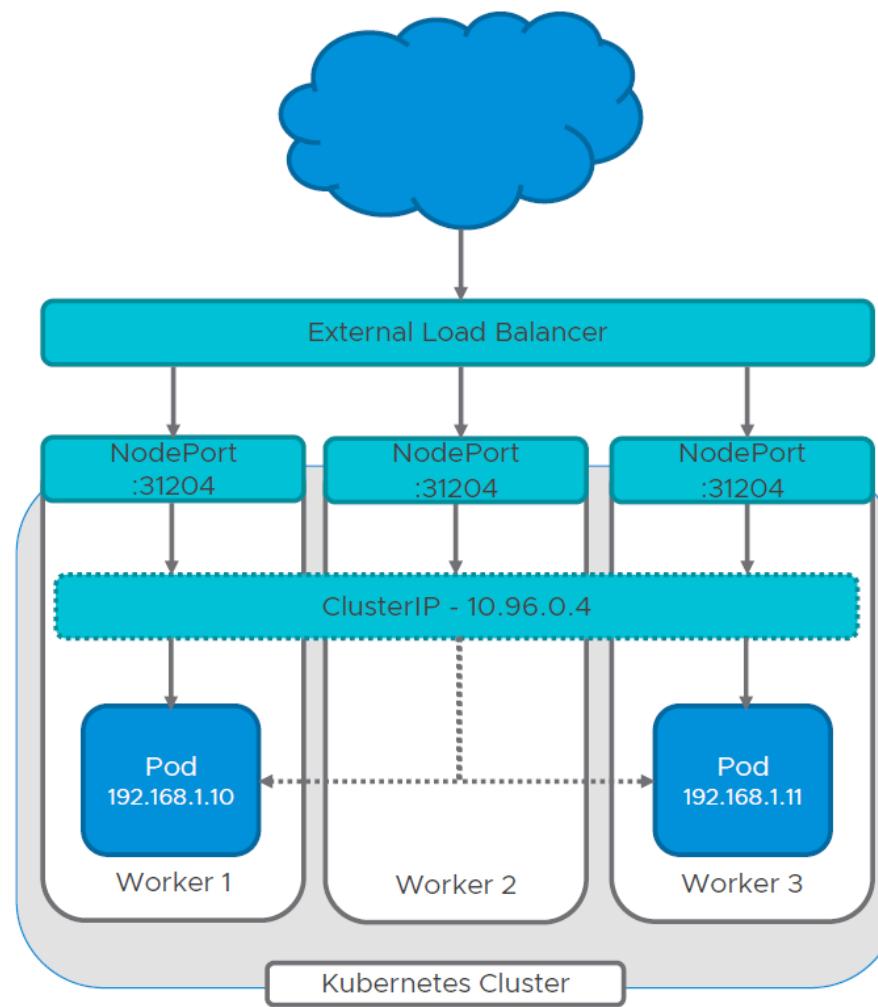
# Access Application

```
[root@master ~]# curl master:30008
Server address: 10.244.189.71:80
Server name: rs-web-w7kx8
Date: 12/Oct/2020:18:26:36 +0000
URI: /
Request ID: fc0f5ef2947395ab4b857e54a8eb6da3
[root@master ~]# curl worker1:30008
Server address: 10.244.235.141:80
Server name: rs-web-6wn7r
Date: 12/Oct/2020:18:26:43 +0000
URI: /
Request ID: 25b9ff3dc70255bf2fc521029a0e939a
[root@master ~]# curl worker2:30008
Server address: 10.244.189.71:80
Server name: rs-web-w7kx8
Date: 12/Oct/2020:18:26:47 +0000
URI: /
Request ID: 4ea79ac1658affc6a4e64c7fe54de5db
```

# Load Balancer

- LoadBalancer:
  - Creates and manages an external load balancer
  - Implementation
    - leverages NodePort for traffic ingress
    - leverages ClusterIP for load balancing to pods
    - plugins available for different Load Balancer implementations

# Load Balancer



# Manual Scheduling

# Kube-Scheduler

The scheduler considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster. Then it schedules the pod to an appropriate compute node.

# How Kube-Scheduler Works?

**Kube-Scheduler selects a node for the pod in a two step operation:**

**1) Filtering:**

Filter checks whether a candidate Node has enough available resource to meet a Pod's specific resource requests

**2) Ranking:**

Ranks the remaining nodes to choose the most suitable Pod placement.

# Manual Scheduling

```
apiVersion: v1
kind: Pod
metadata:
  name: worker1-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
node_name: worker1
```

# Verify

```
[root@master ~]# kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS	GATES
worker1-pod	1/1	Running	0	26s	10.244.235.142	worker1	<none>		<none>

# Taints and Tolerations

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

## Use Cases:-

- Dedicated Nodes
- Nodes with special hardware

# Taints and Tolerations

A

B

C

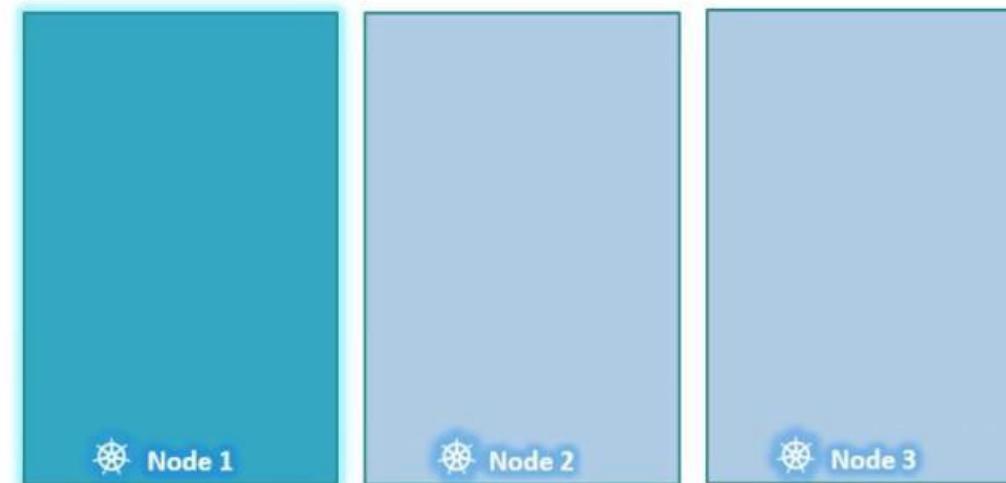
D

Toleration=blue

2

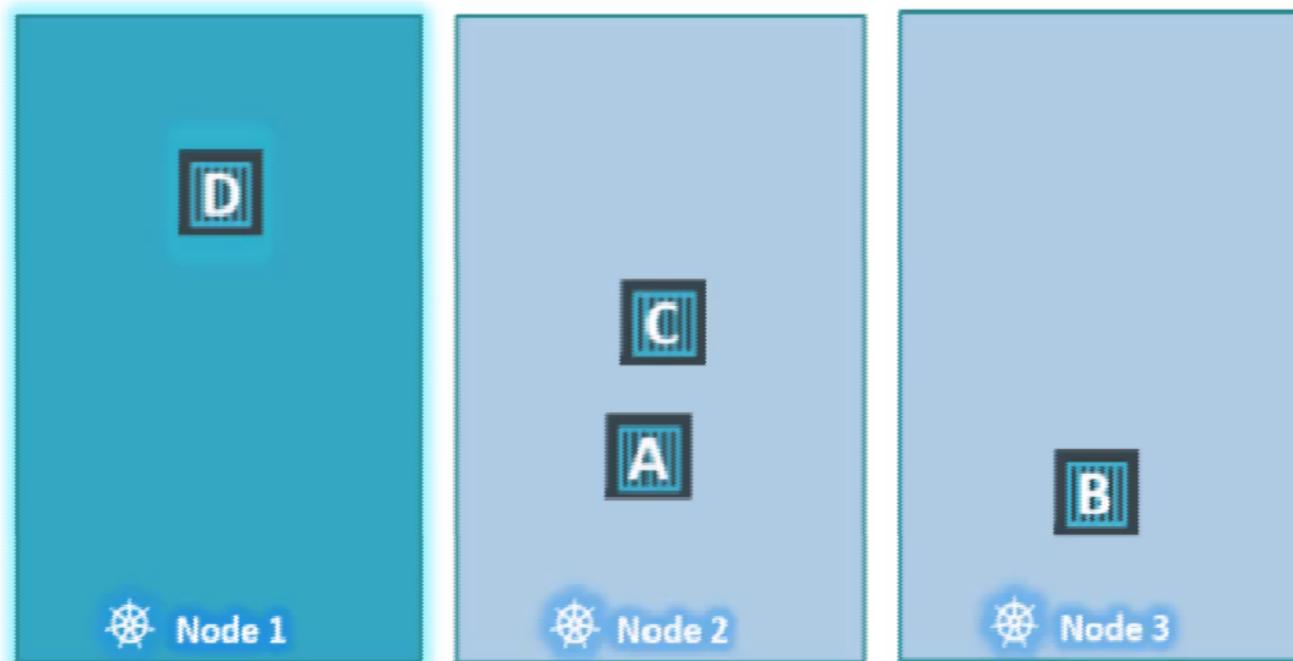
Taint=blue

1

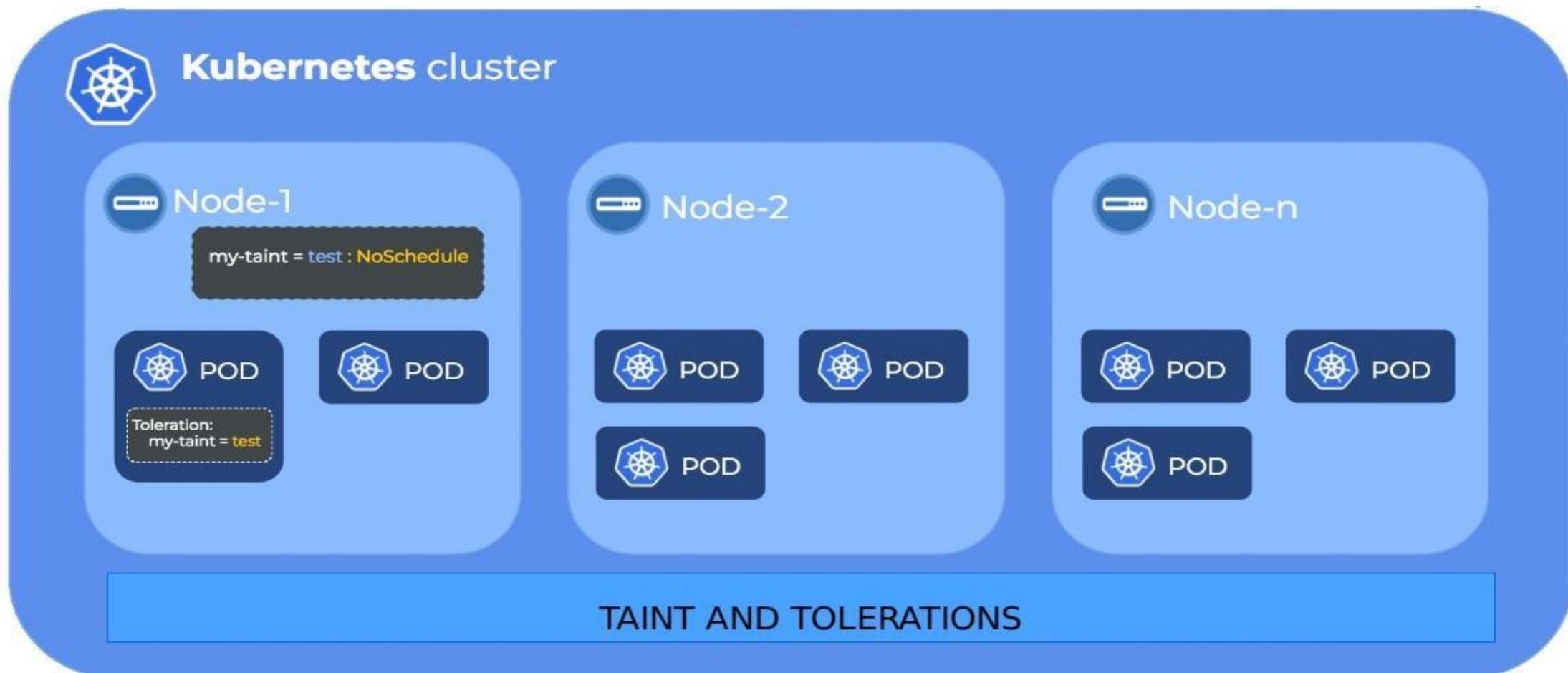


# Taints and Tolerations

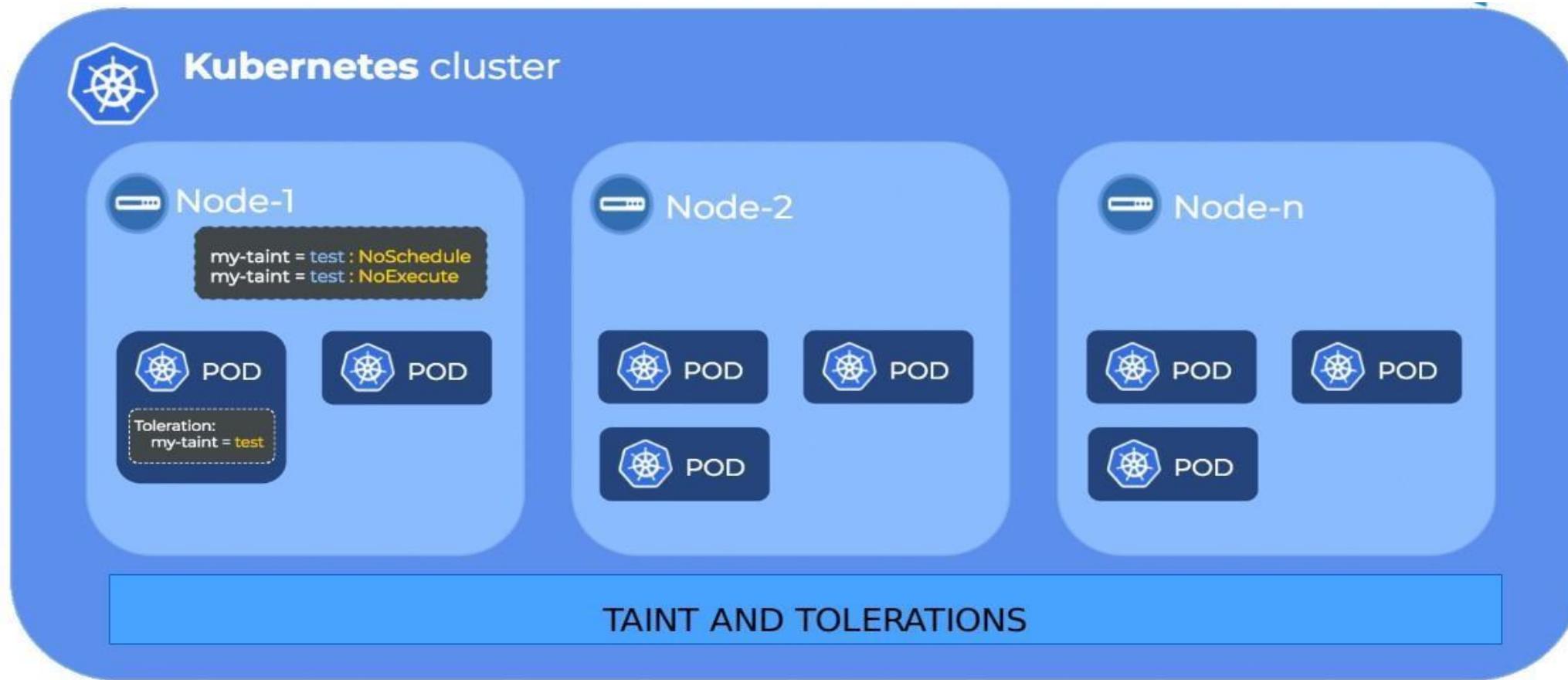
Taint=blue



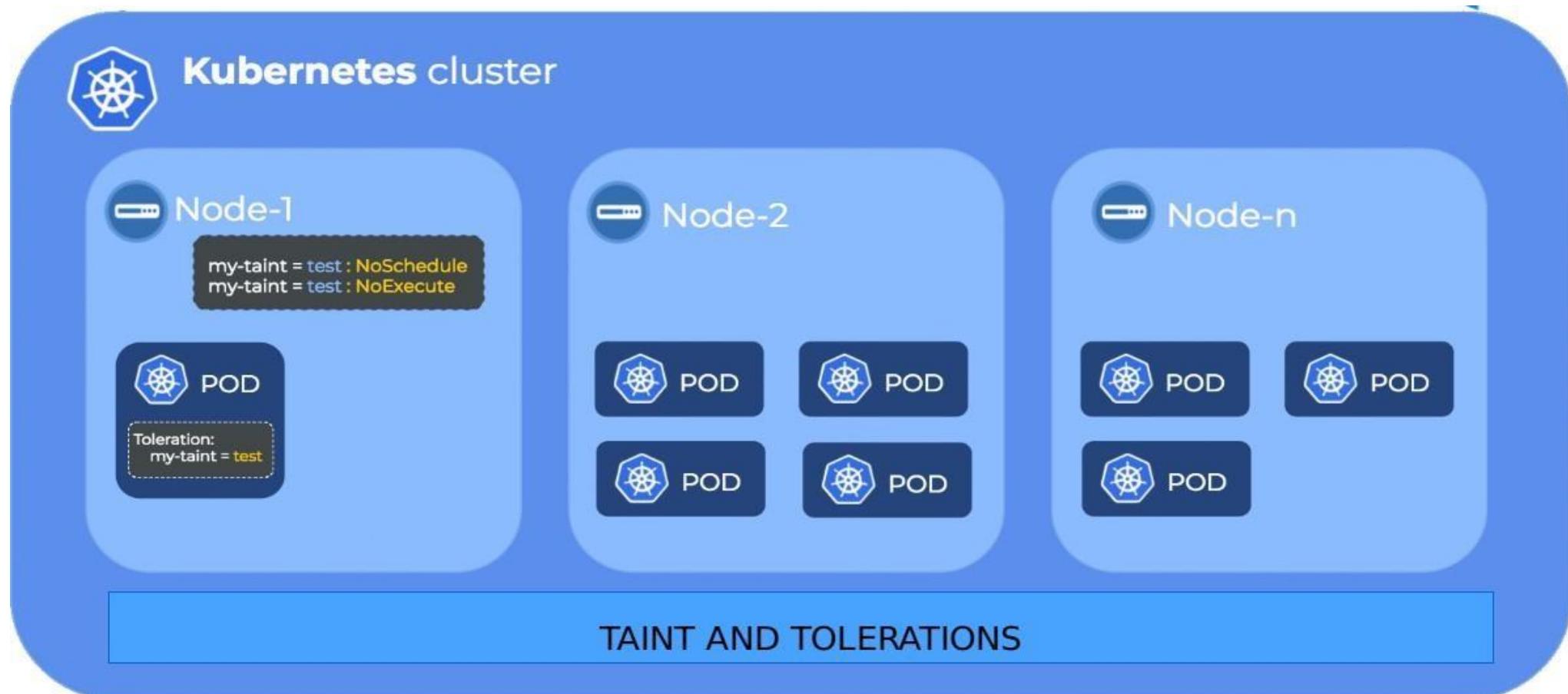
# Taint Effect: NoSchedule



# Taint Effect: NoExecute



# Taint Effect: NoExecute



# Taint on Node

**To set taint:**

```
#kubectl taint nodes <node name> <key>=<value>:<effect>
```

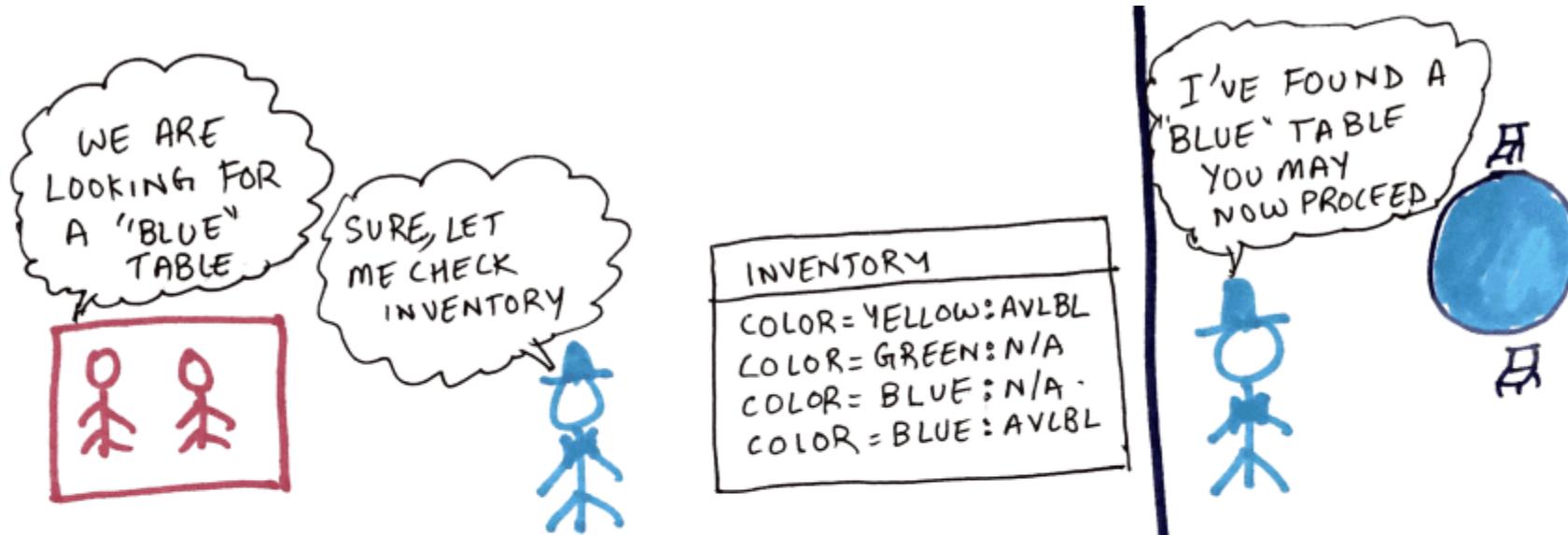
**To remove taint:**

```
#kubectl taint nodes <node name> <key>-
```

# Toleration on Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: quay.io/gauravkumar9130/nginxdemo
  tolerations:
  - key: "app"
    operator: "Equal"
    value: "blue"
    effect: "NoSchedule"
```

# Node Selector



# Node Selector (Label on Nodes)

**To set label:**

```
#kubectl label nodes <node name> <key>=<value>
```

**To delete label:**

```
#kubectl label nodes <node name> <key>-
```

# Node Selector

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: large-pod
    image: nginx
nodeSelector:
  size: large
```

# Node Affinity

- **requiredDuringScheduling**: means at the time of creation it is must to have label on node otherwise it will not create pod
- **preferredDuringScheduling**: means at the time of create it is not mandatory to have label on node if label is exist on node then it will create pod otherwise it will create pod on another available nodes.
- **ignoredDuringExecution**: it means if the pod is already exist on node and administrator changes the node label so it will ignore and it will be running on same node.
- **requiredDuringExecution**: it means if the pod is already exist on node and administrator changes the node label so it will be removed.

# Node Affinity

**Affinity types->**

- 1) requiredDuringSchedulingIgnoredDuringExecution
- 2) preferredDuringSchedulingIgnoredDuringExecution
- 3) requiredDuringSchedulingRequiredDuringExecution
- 4) preferredDuringSchedulingRequiredDuringExecution

# Node Affinity Demo

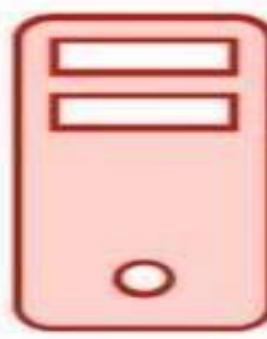
```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: quay.io/gauravkumar9130/nginxdemo
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "size"
            operator: "In"
            values:
            - "large"
            - "medium"
```

# Taint and Toleration VS Node Affinity

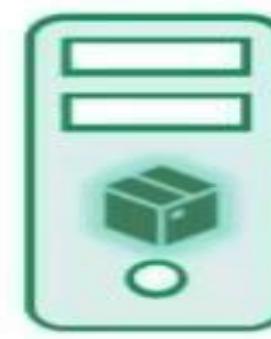
# Taint and Toleration



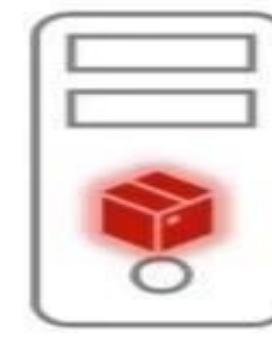
Blue



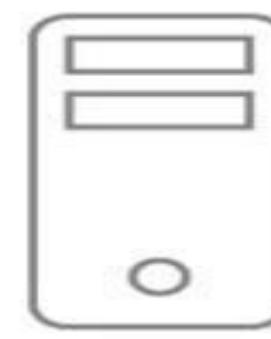
Red



Green



Other



Other



# Node Affinity



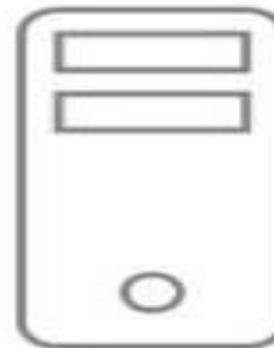
Blue



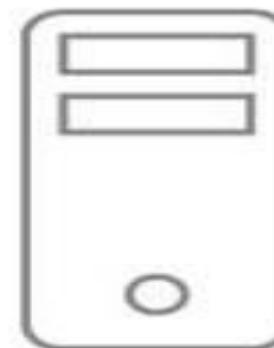
Red



Green



Other



Other



# Daemon Set

A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

**Some typical uses of a DaemonSet are:**

- running a cluster storage daemon, such as glusterd, ceph, on each node.
- running a logs collection daemon on every node, such as fluentd or filebeat.

# Demo of Daemon Set

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: monitoring-tool
spec:
  selector:
    matchLabels:
      app: monitoring
  template:
    metadata:
      labels:
        app: monitoring
    spec:
      containers:
        - name: monitoring-container
          image: nginx
```

# Verify

```
[root@master kube]# kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS	GATES
monitoring-tool-66zlx	1/1	Running	0	22s	10.244.235.143	worker1	<none>	<none>	<none>
monitoring-tool-jmfch	1/1	Running	0	22s	10.244.189.72	worker2	<none>	<none>	<none>

```
[root@master kube]#
```

# Deployments

- A Deployment provides declarative updates for pods and replicaset.
- You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

# Deployment Strategy – Built In

**Two types of built in strategies ->**

## **1) RollingUpdate**

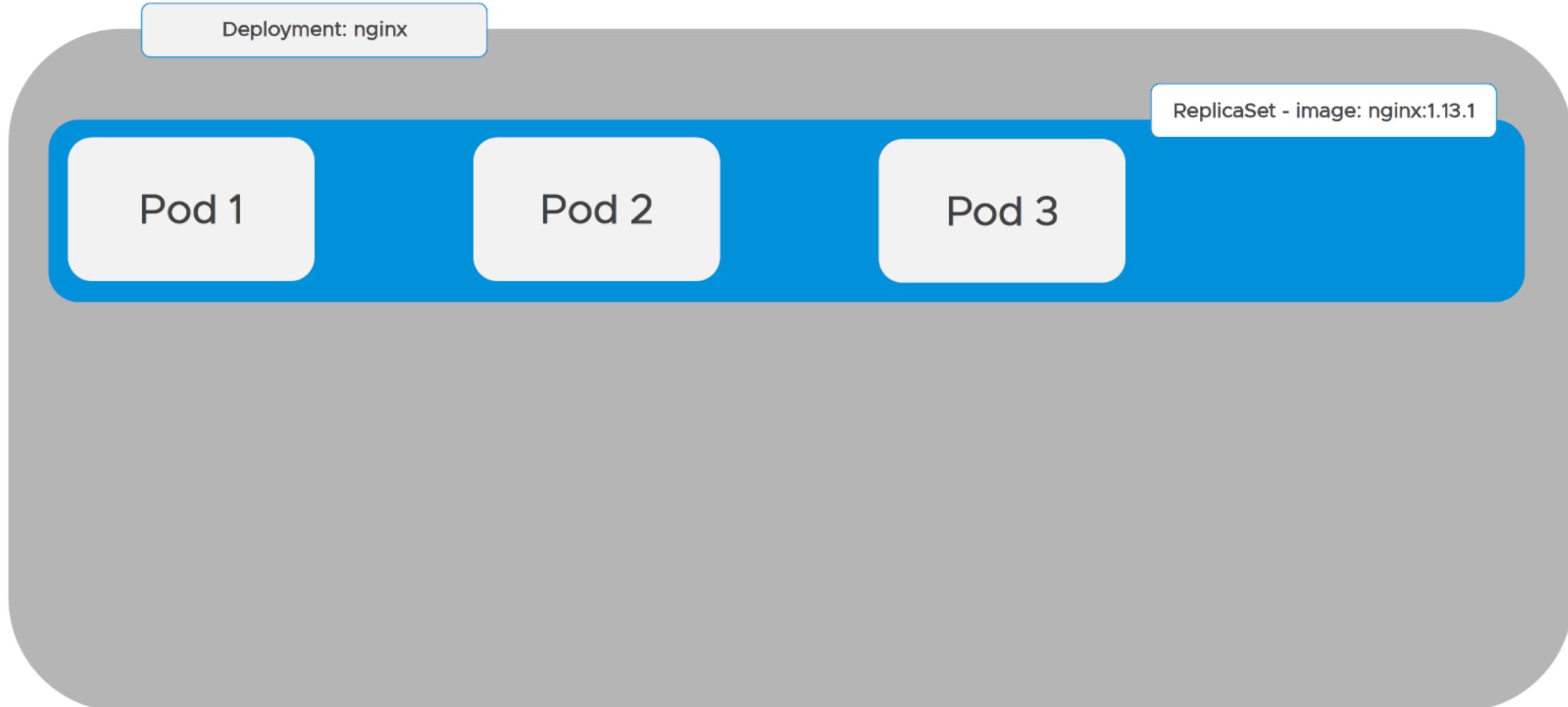
default if not defined

new ReplicaSet is created, then scaled up as the old ReplicaSet is scaled down

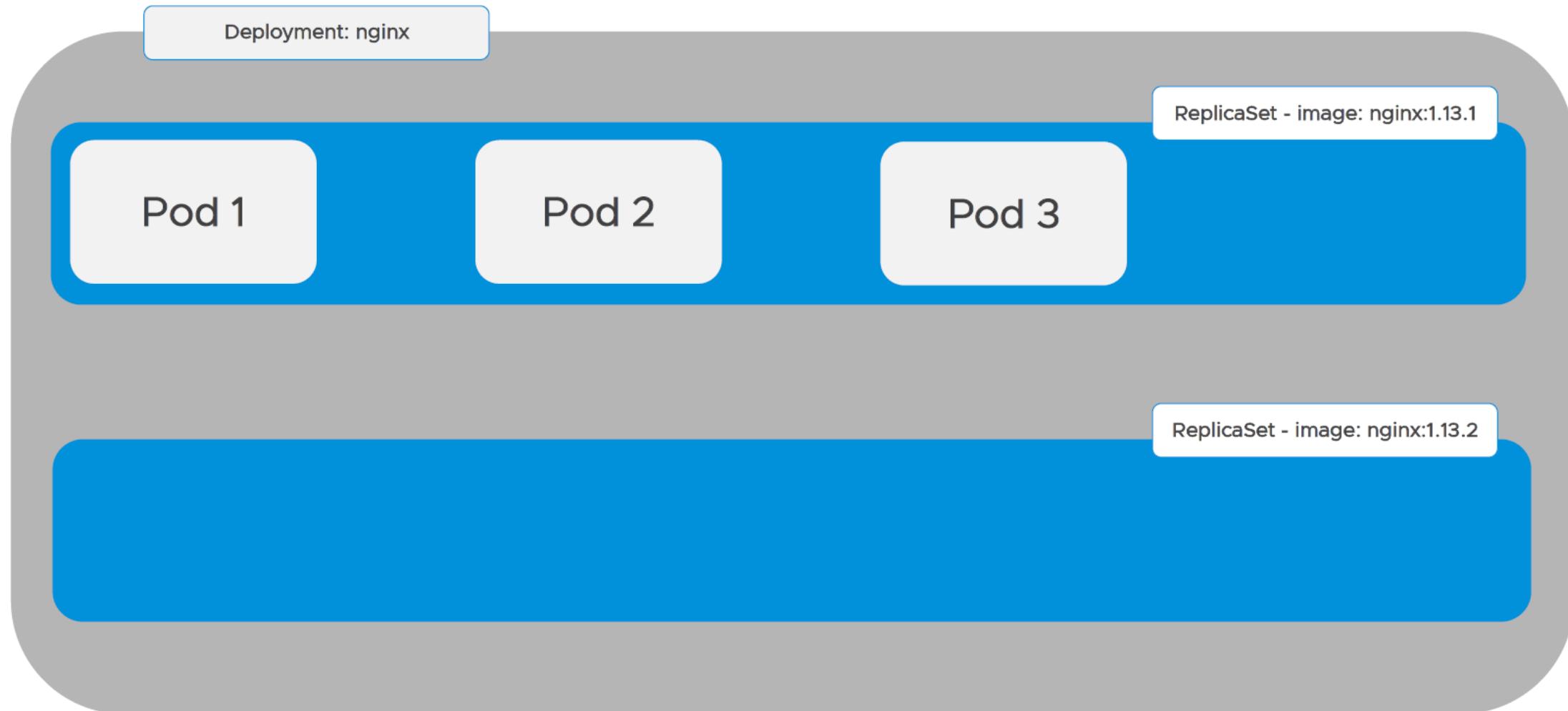
## **2) Recreate**

removes all existing pods in the existing ReplicaSet first  
then creates new pods in the new ReplicaSet

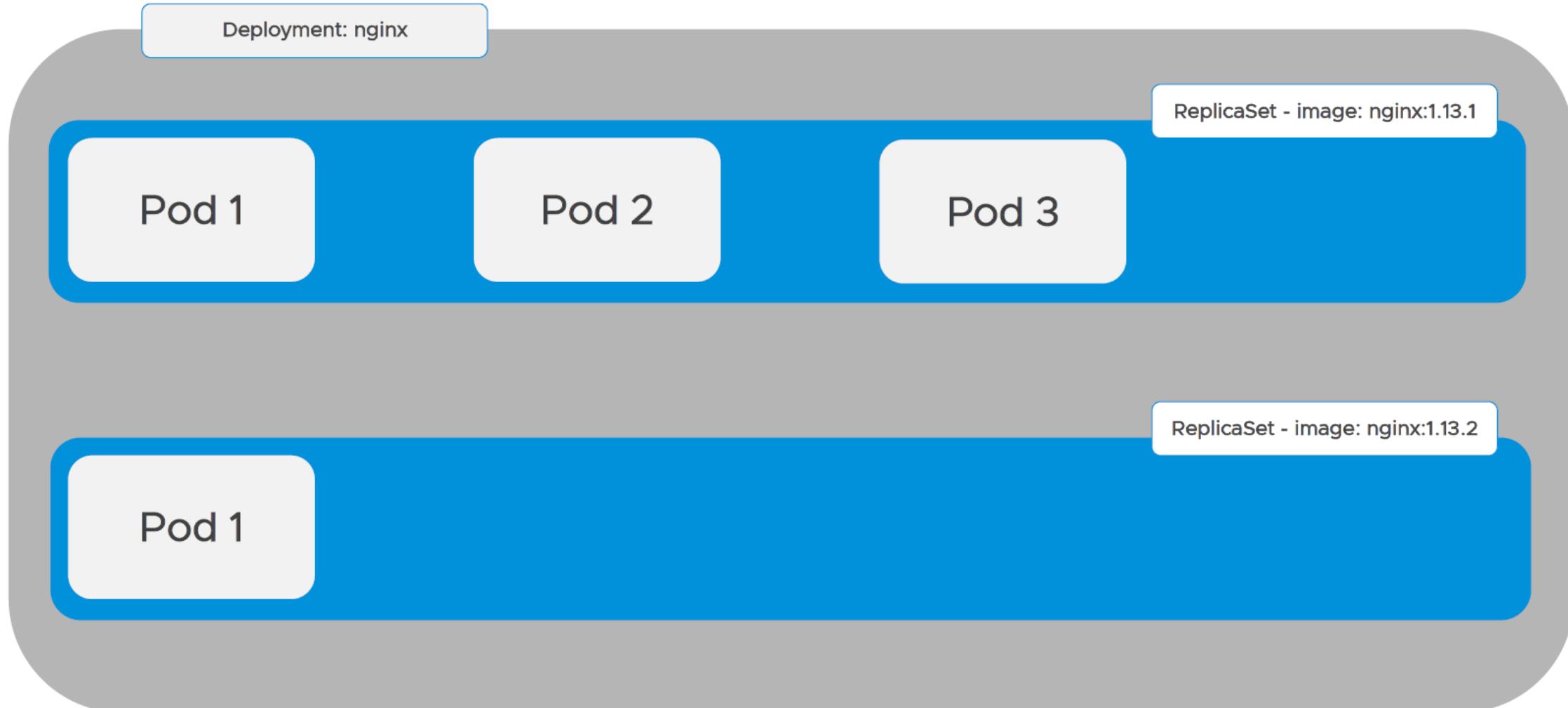
# Deployment Strategies – Rolling Update



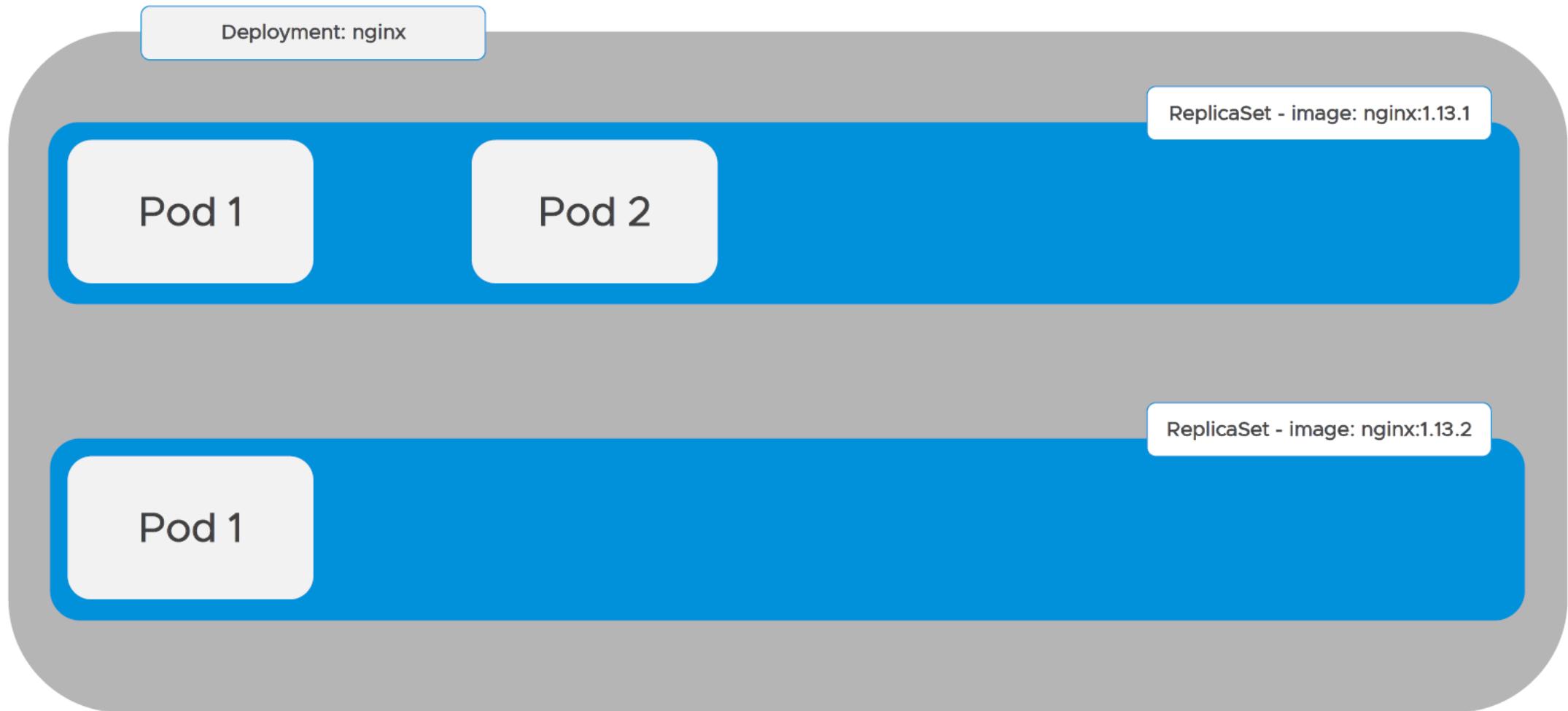
# Deployment Strategies – Rolling Update



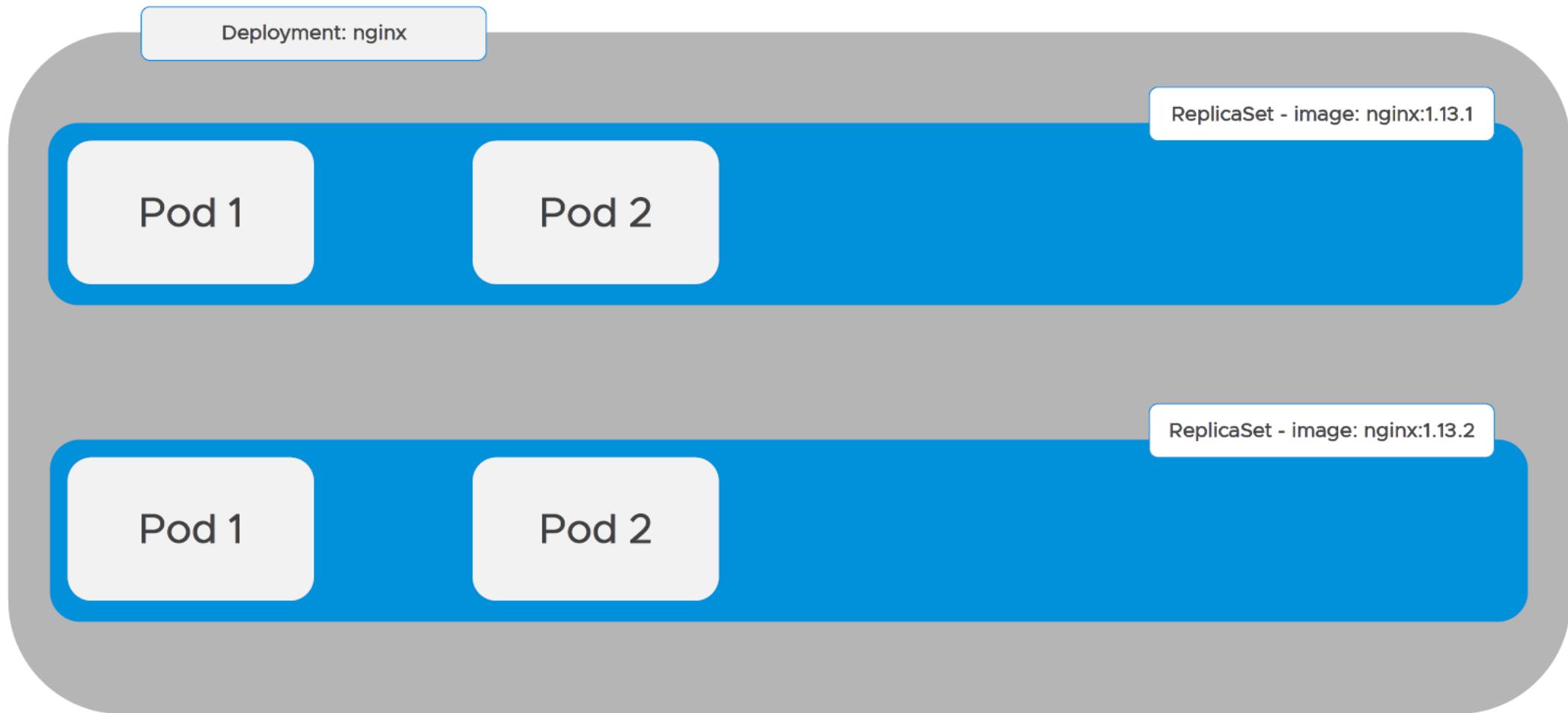
# Deployment Strategies – Rolling Update



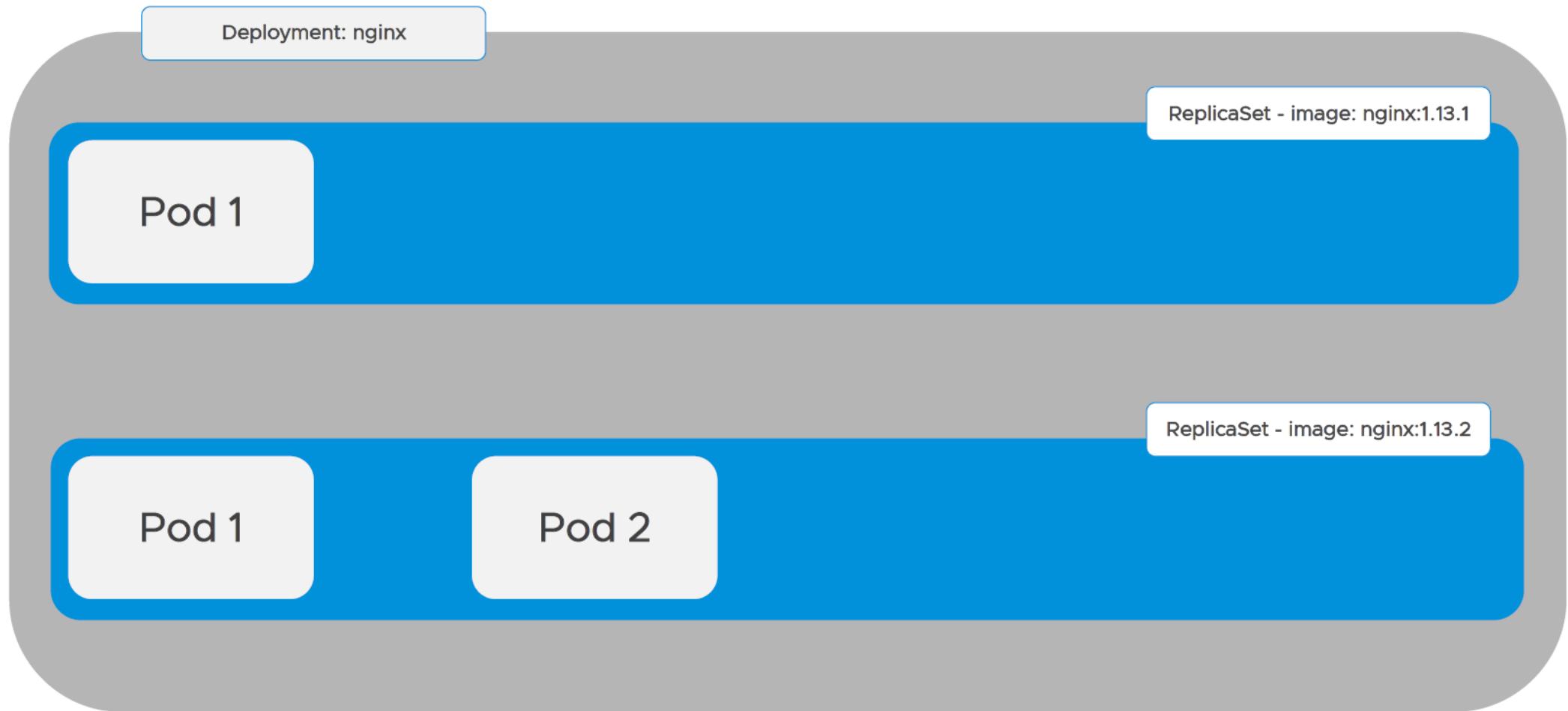
# Deployment Strategies – Rolling Update



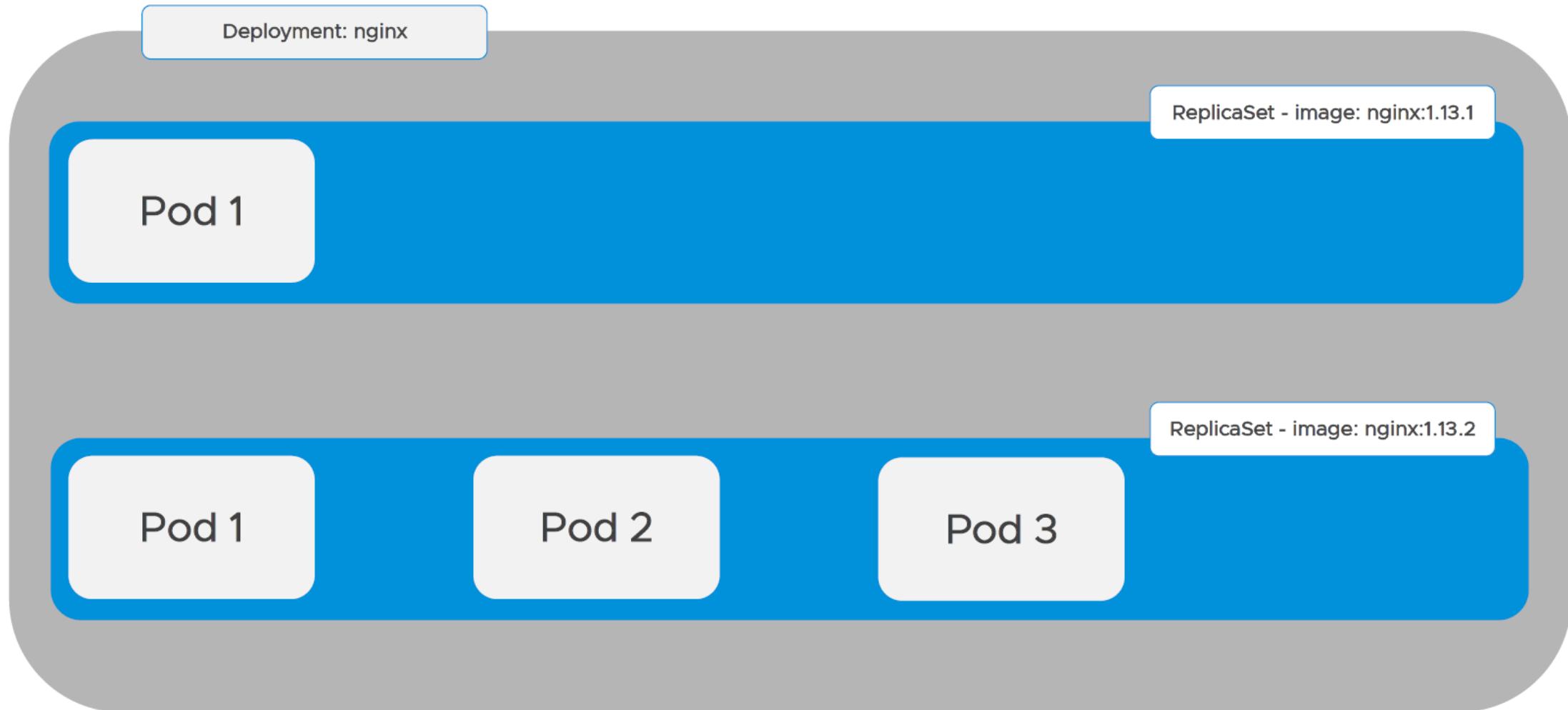
# Deployment Strategies – Rolling Update



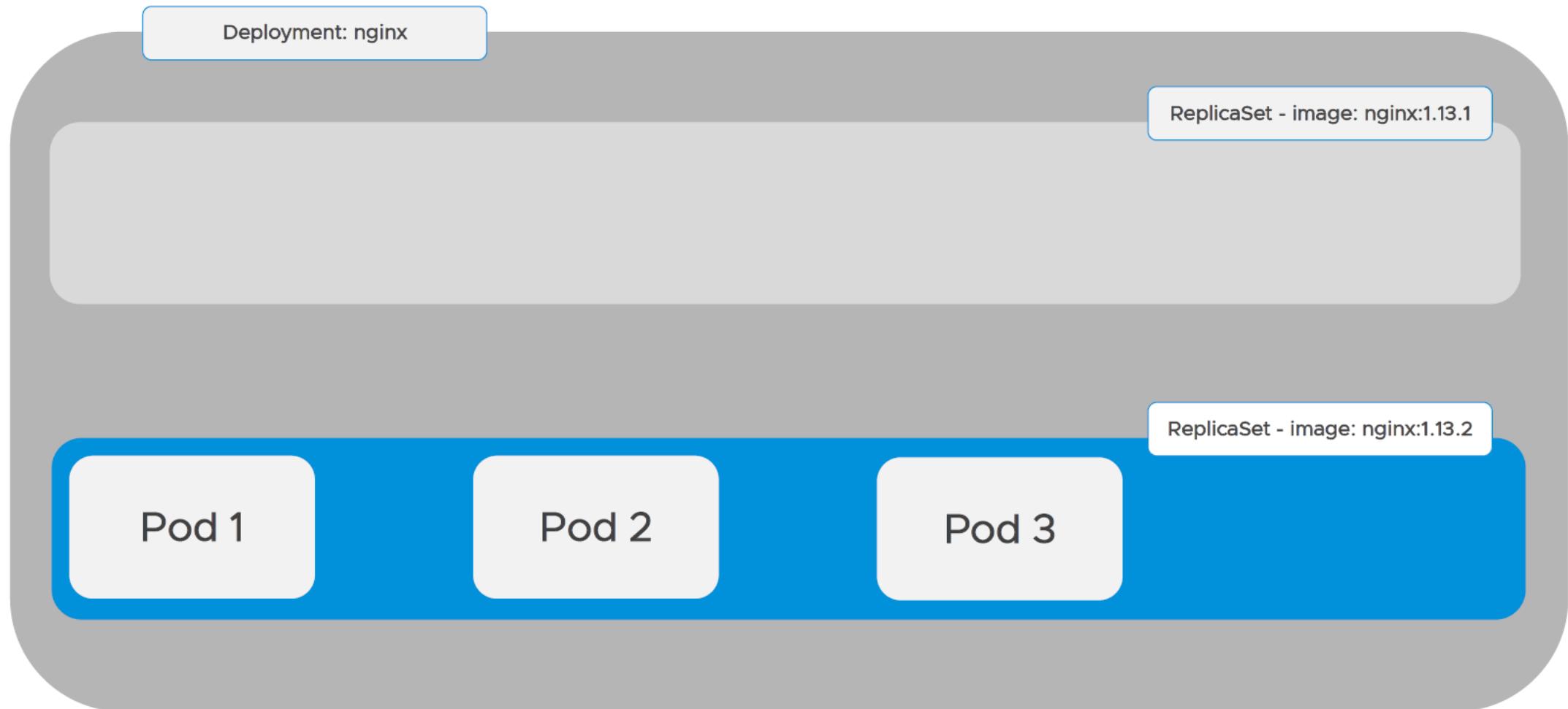
# Deployment Strategies – Rolling Update



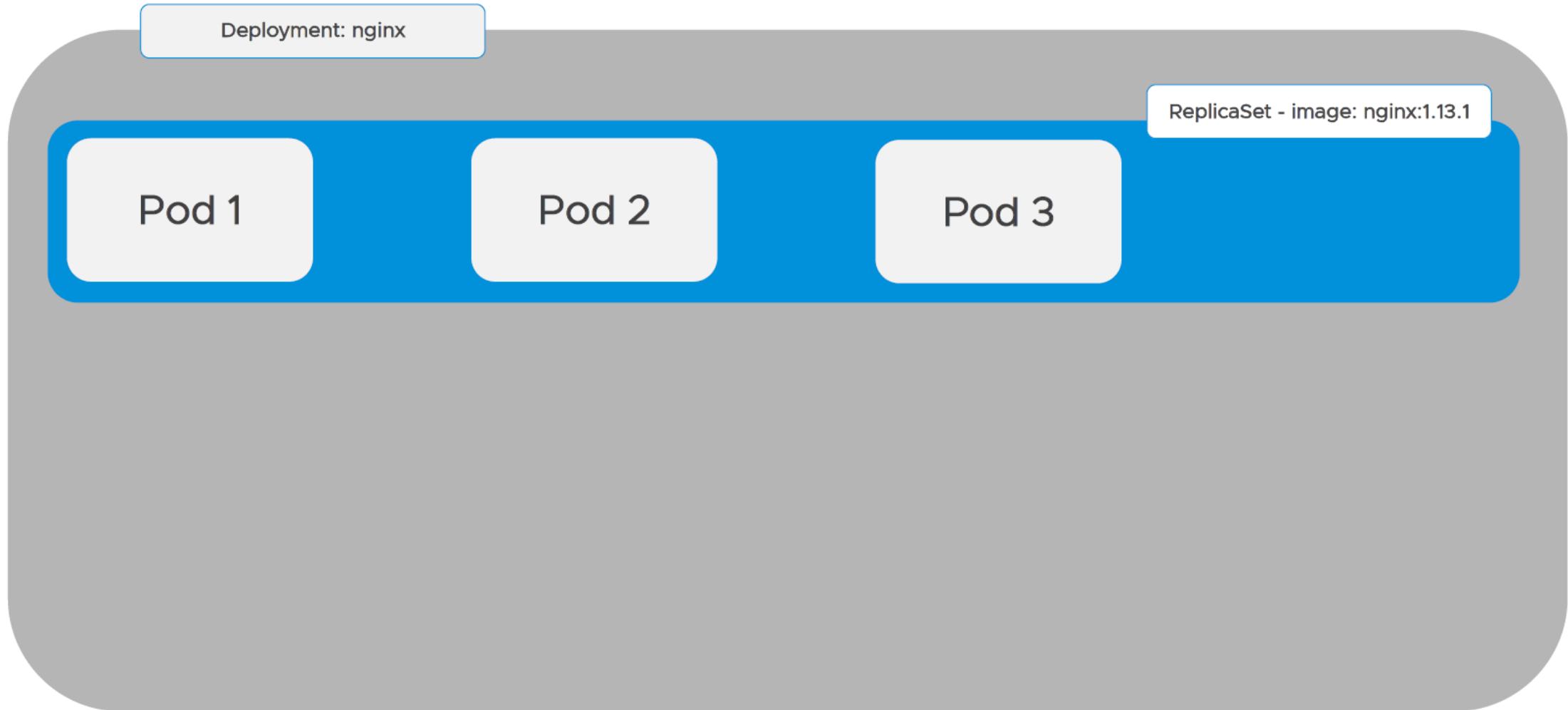
# Deployment Strategies – Rolling Update



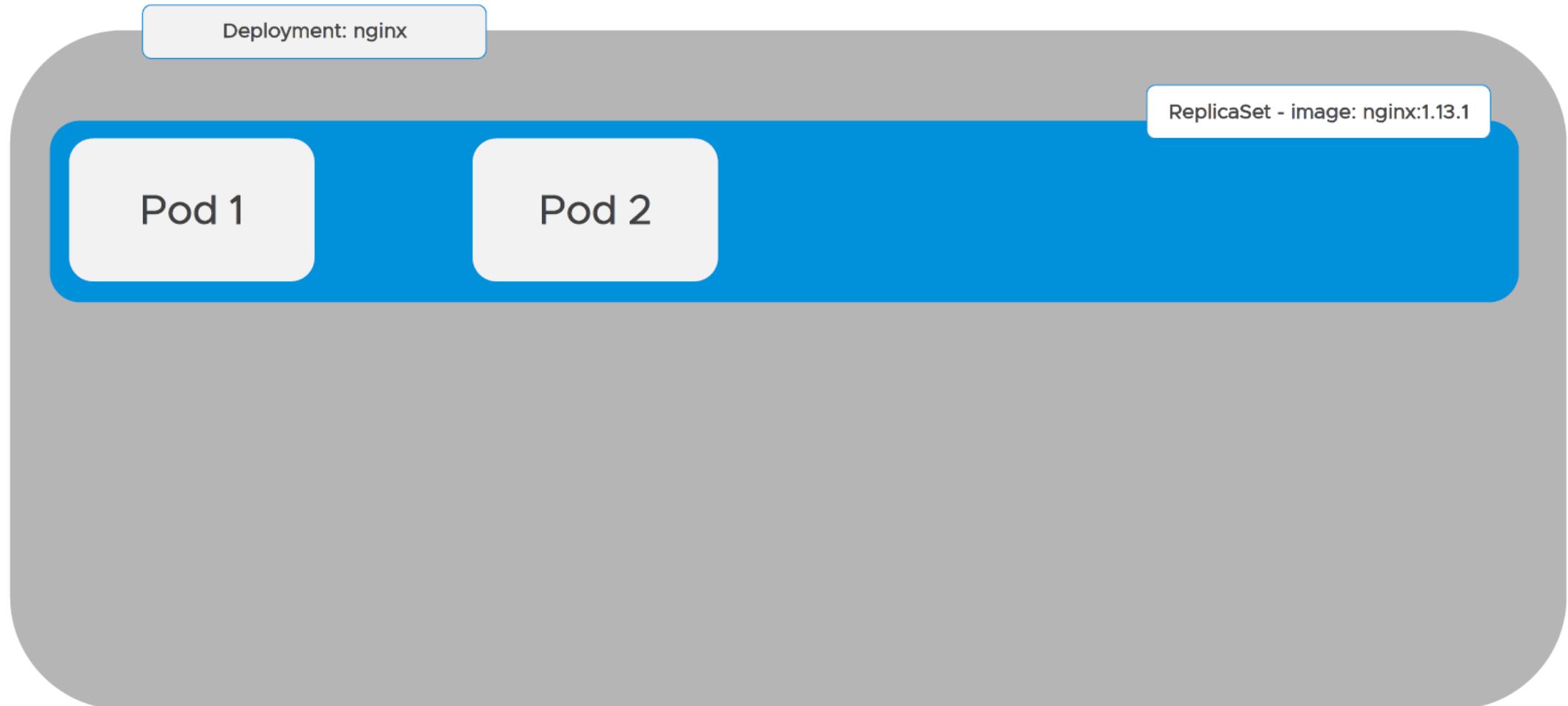
# Deployment Strategies – Rolling Update



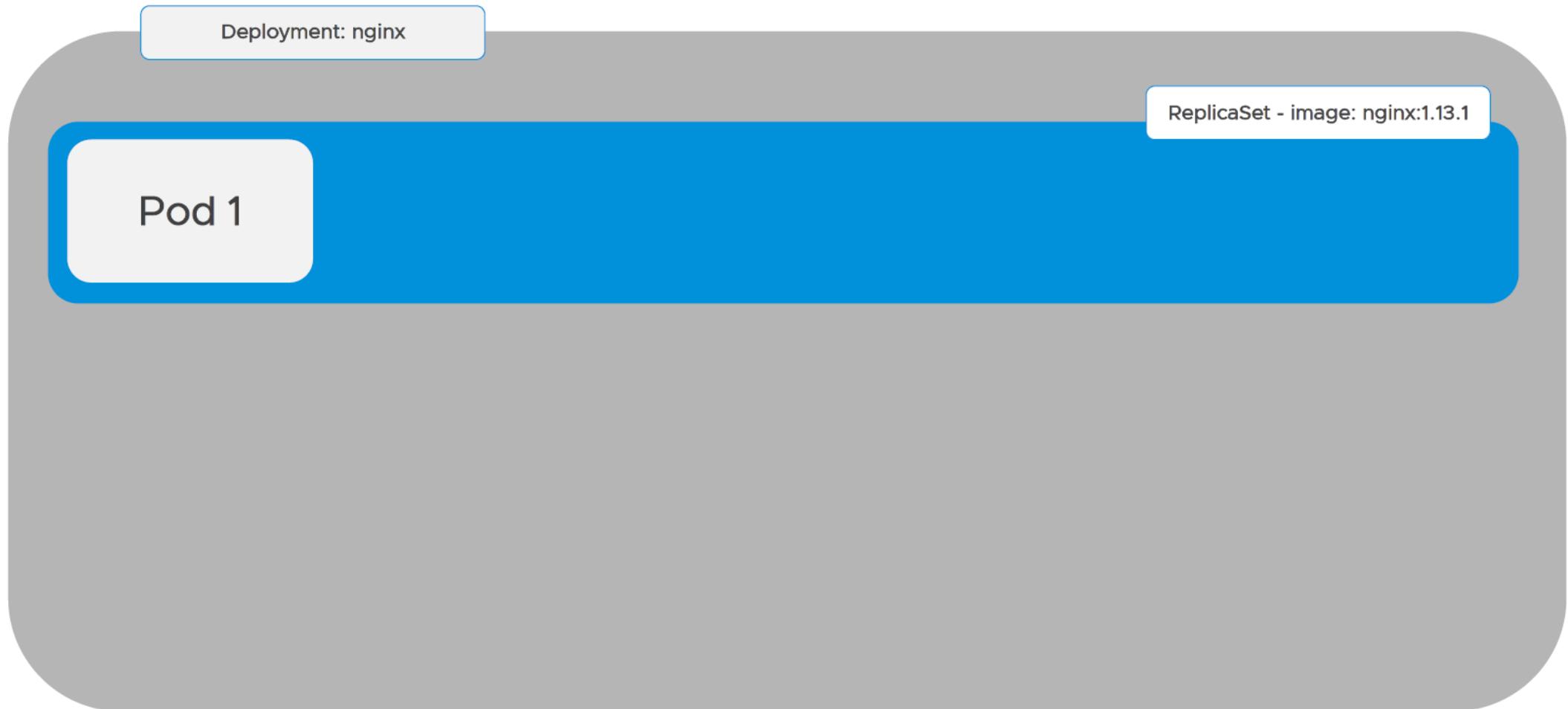
# Deployment Strategies – Recreate



# Deployment Strategies – Recreate



# Deployment Strategies – Recreate



# Deployment Strategies – Recreate

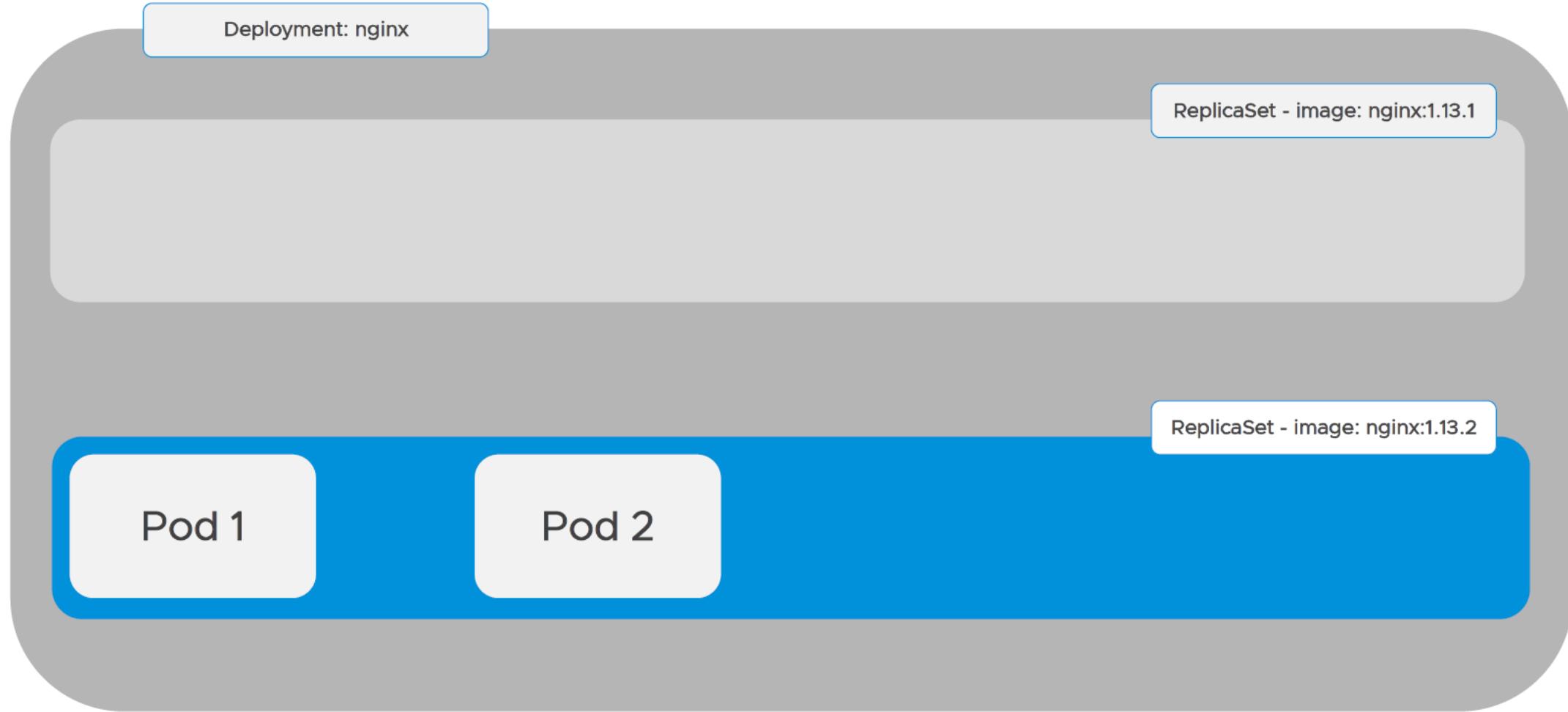
Deployment: nginx

ReplicaSet - image: nginx:1.13.1

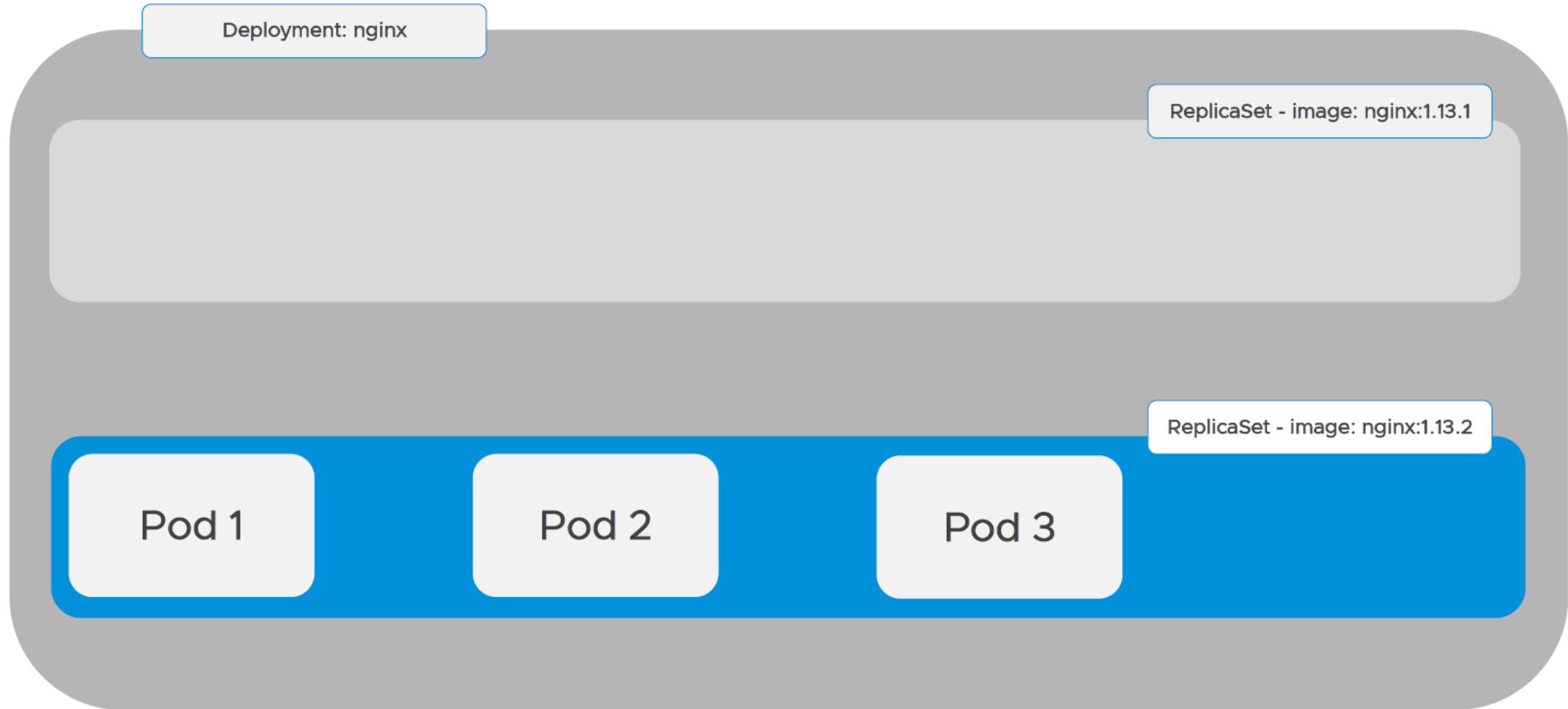
# Deployment Strategies – Recreate



# Deployment Strategies – Recreate



# Deployment Strategies – Recreate



# Demo of Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx-deployment
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
```

# Deployment Commands

**To update manually:**

```
#kubectl set image deployment/<deployment name> <container name>=<image name>:<tag>
```

**To list history of deployment:**

```
#kubectl rollout history deployment/<deployment name>
```

**To go to previous version:**

```
#kubectl rollout undo deployment/<deployment name>
```

**To go to particular version:**

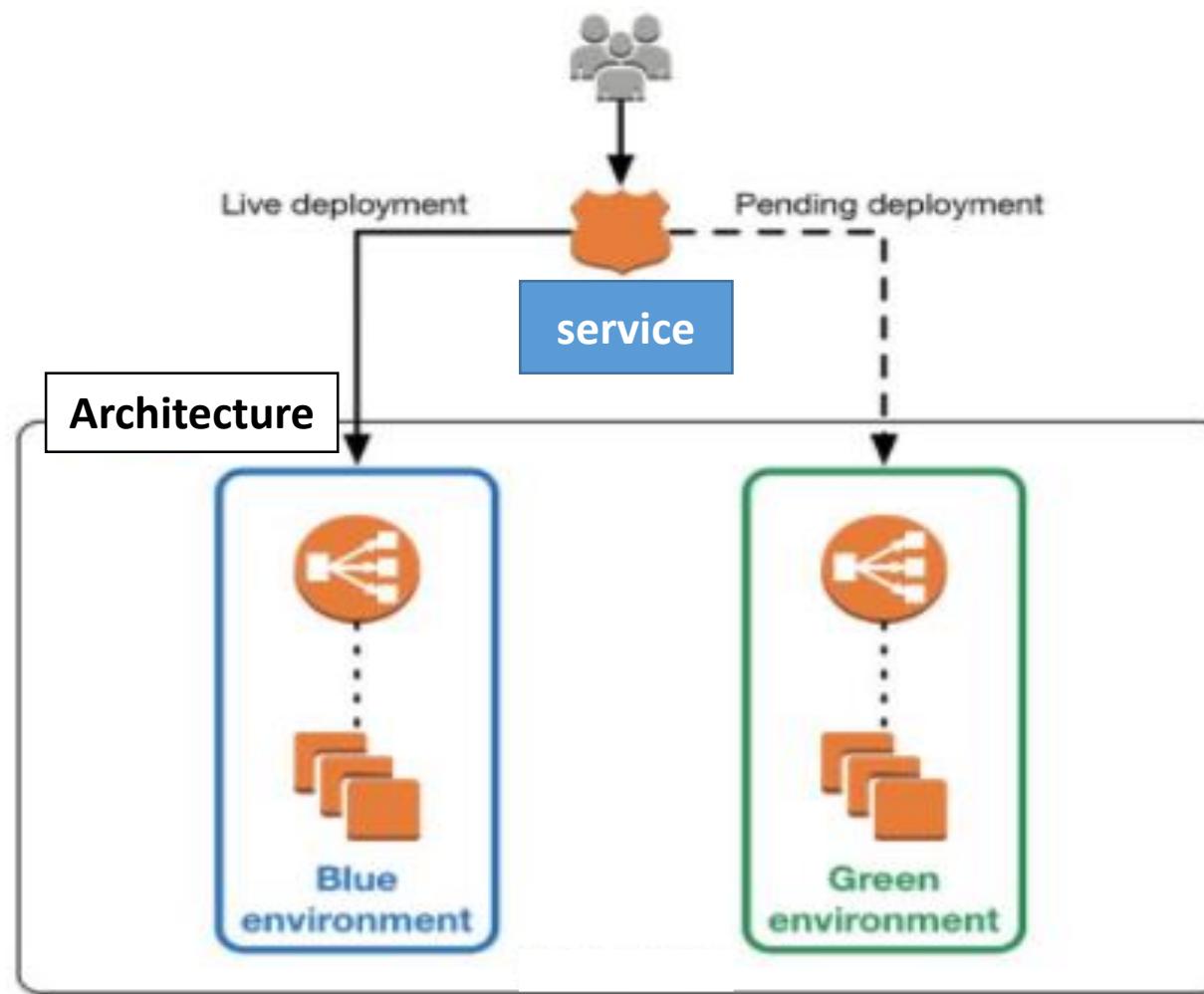
```
#kubectl rollout undo deployment/<deployment name> --to-revision=<number>
```

# Blue-Green Deployment

**Blue green deployment** is an application release model that gradually transfers user traffic from a previous version of an app or microservice to a nearly identical new release—both of which are running in production.

The old version can be called the blue environment while the new version can be known as the green environment.

# Example



# Deployment Blue Application

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blue-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
      version: blue
  template:
    metadata:
      labels:
        app: nginx
        version: blue
    spec:
      containers:
        - name: abc
          image: nginx
          imagePullPolicy: IfNotPresent
```

```
apiVersion: v1
kind: Service
metadata:
  name: mysvc
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: nginx
    version: blue
```

# Scenario

**Now we have a requirement to upgrade our application, so as we are using blue-green deployment, we will create a new deployment with new updated application and update the service selector.**

# Green Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
      version: green
  template:
    metadata:
      labels:
        app: nginx
        version: green
    spec:
      containers:
      - name: abc
        image: quay.io/gauravkumar9130/nginxdemo
        imagePullPolicy: IfNotPresent
```

Now update the service selector:

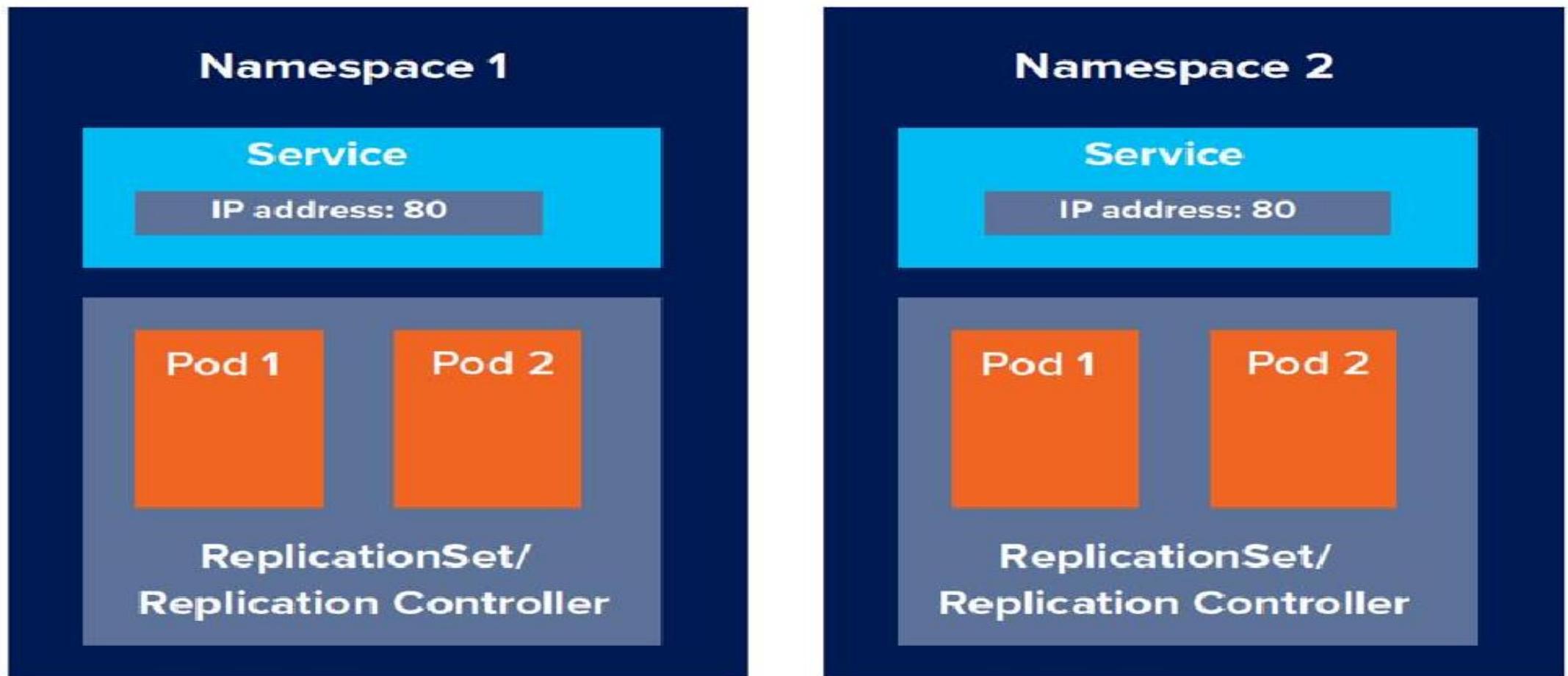
```
#kubectl edit svc mysvc
selector:
  app: nginx
  version: green
```

# Namespace

# Namespace

- Namespace provides an additional qualification to a resource name. This is helpful when multiple teams are using the same cluster and there is a potential of name collision. It can be as a virtual wall between multiple clusters.
- It offers an isolation for process interaction within OS.

# Namespace



# Create Namespace

---

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

OR

```
#kubectl create namespace dev
```

# List Namespace

```
[root@localhost ~]# kubectl get namespace
NAME          STATUS   AGE
default       Active   6h45m
dev           Active   7s
kube-node-lease Active   6h45m
kube-public   Active   6h45m
kube-system   Active   6h45m
[root@localhost ~]#
```

# To Add Pod in Namespace

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

~

```
[root@localhost ~]# kubectl get pods -n dev
NAME      READY   STATUS    RESTARTS   AGE
dev-pod   1/1     Running   0          13s
[root@localhost ~]#
```

# Namespace Commands

```
#kubectl get namespace
```

```
# kubectl describe namespace <Namespace name>
```

```
#kubectl delete namespace <Namespace name>
```

```
#kubectl get <resource> -A
```

**To switch from default namespace to another namespace:**

```
#kubectl config set-context $(kubectl config current-context) --  
namespace=<namespace name>
```

# Resource Quota for Namespace

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-quota
  namespace: dev
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: "5Gi"
    limits.cpu: "10"
    limits.memory: "10Gi"
```

~

```
[root@localhost ~]# kubectl get quota -n dev
NAME      AGE     REQUEST                                LIMIT
dev-quota  7s      pods: 1/10, requests.cpu: 0/4, requests.memory: 0/5Gi  limits.cpu: 0/10, limits.memory: 0/10Gi
[root@localhost ~]#
```

# Resource Limits for Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dev-app
  namespace: dev
spec:
  containers:
    - name: dev-app
      image: nginx
      ports:
        - containerPort: 80
      resources:
        requests:
          memory: "1Gi"
          cpu: "1"
        limits:
          memory: "2Gi"
          cpu: 2
```

Pod will be killed if it tries  
to use more than 2Gi  
memory.

Pod will be throttled if it  
uses more than 2 core.

Pod will be scheduled on a  
node with at least 1Gi  
memory and 1 CPU

# Environment Variable

# Types of Environment Variables

- Plain Key
- ConfigMap
- Secret

# Plain Key Environment Variable

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  containers:
    - name: mydb
      image: quay.io/gauravkumar9130/mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: root123
        - name: MYSQL_USER
          value: gaurav
        - name: MYSQL_PASSWORD
          value: gaurav123
```

# Config Map

A ConfigMap is a dictionary of configuration settings. This dictionary consists of key-value pairs of strings. Kubernetes provides these values to your containers. Like with other dictionaries (maps, hashes, ...) the key lets you get and set the configuration value.

```
#kubectl create configmap <config map name> --from-literal=<key>=<value>
```

# Demo of Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-app
data:
  MYSQL_ROOT_PASSWORD: root123
  MYSQL_USER: gaurav
  MYSQL_USER_PASSWORD: gaurav123
```

---

```
[root@master ~]# kubectl get configmaps
NAME      DATA   AGE
db-app    3      14s
```

# Inject ConfigMap in Pod

---

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  containers:
  - name: mydb
    image: quay.io/gauravkumar9130/mysql
  envFrom:
  - configMapRef:
      name: db-app
```

# Secret

Secrets can be defined as Kubernetes objects used to store sensitive data such as user name and passwords with encryption.

```
#kubectl create secret generic <secret name> --from-literal=<key>=<value>
```

# Create Secret with YAML

```
[root@master ~]# echo -n "root123" | base64  
cm9vdDEyMw==  
[root@master ~]# echo -n "gaurav" | base64  
Z2F1cmF2  
[root@master ~]# echo -n "gaurav123" | base64  
Z2F1cmF2MTIz  
[root@master ~]#  
apiVersion: v1  
kind: Secret  
metadata:  
  name: db-secret  
data:  
  MYSQL_ROOT_PASSWORD: cm9vdDEyMw==  
  MYSQL_USER: Z2F1cmF2  
  MYSQL_USER_PASSWORD: Z2F1cmF2MTIz
```

# Inject Secret in Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  containers:
  - name: mydb
    image: quay.io/gauravkumar9130/mysql
    envFrom:
    - secretRef:
        name: db-secret
```

# Mount Environment Variables as Volume

# Create ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  name: gaurav
  password: redhat
```

# Mount Config Map as Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: config-pod
spec:
  containers:
    - name: config
      image: nginx
      volumeMounts:
        - name: config-vol
          mountPath: /data
  volumes:
    - name: config-vol
      configMap:
        name: app-config
```

# Create Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  name: Z2F1cmF2
  password: cmVkaGF0
-
```

# Mount Secret as Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
    - name: secret
      image: nginx
      volumeMounts:
        - name: secret-vol
          mountPath: /data
  volumes:
    - name: secret-vol
      secret:
        secretName: app-secret
```

# Secret Environment Variable Mapping

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
    - name: secret
      image: nginx
      env:
        - name: CREDENTIALS
          valueFrom:
            secretKeyRef:
              name: app-secret
              key: password
```

# Storage

# Storage in Kubernetes

## Volumes

- Exposed at Pod level and backend different ways
- emptyDir: ephemeral scratch directory that lives for the life of pod

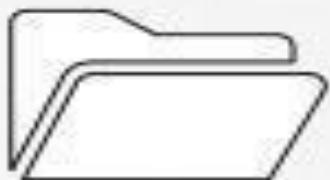
## Persistent Volumes

- Abstraction away from the storage provider(AWS, GCE)
- Have a lifecycle independent of any individual pod that uses it

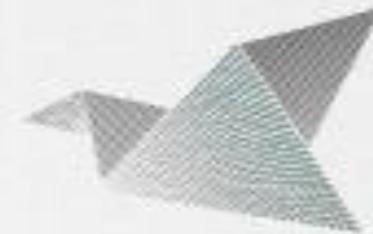
## Persistent Volume Claims

- Request for storage with specific details(size, access modes, etc.)

# Types of Kubernetes Volume



NFS

Flocker™  
by ClusterHQ™

ceph

SCALEIO



# EmptyDir

An emptyDir volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node

# Demo of EmptyDir

```
apiVersion: v1
kind: Pod
metadata:
  name: cache-pod
spec:
  containers:
    - name: cache-container
      image: quay.io/gauravkumar9130/nginxdemo
      volumeMounts:
        - name: cache-vol
          mountPath: /mytest
  volumes:
    - name: cache-vol
      emptyDir: {}
```

# Persistent Volumes

- Persistent Volumes can be provisioned either statically or dynamically.
  - Static: A pre-provisioned pool of volumes such as iSCSI or Fiber Channel
  - Dynamic: Volumes are created on demand by calling a storage provider's API such as Amazon EBS
- Persistent Volumes have a lifecycle independent of the pods that use them

# Access Modes

The access modes are:

- `ReadWriteOnce` -- the volume can be mounted as read-write by a single node
- `ReadOnlyMany` -- the volume can be mounted read-only by many nodes
- `ReadWriteMany` -- the volume can be mounted as read-write by many nodes

# Demo of Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-voll
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: "1Gi"
  hostPath:
    path: "/webvolumedata" #save data in the scheduled node
```

```
[root@localhost ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM   STORAGECLASS   REASON   AGE
pv-voll   1Gi        RWO          Retain           Available
[root@localhost ~]#
```

# Persistent Volumes Claim

- Created by users to request a persistent volume
- Users can request various properties such as capacity and access modes (e.g. can be mounted once read/write or many times read-only)

# Demo of Persistent Volume Claim

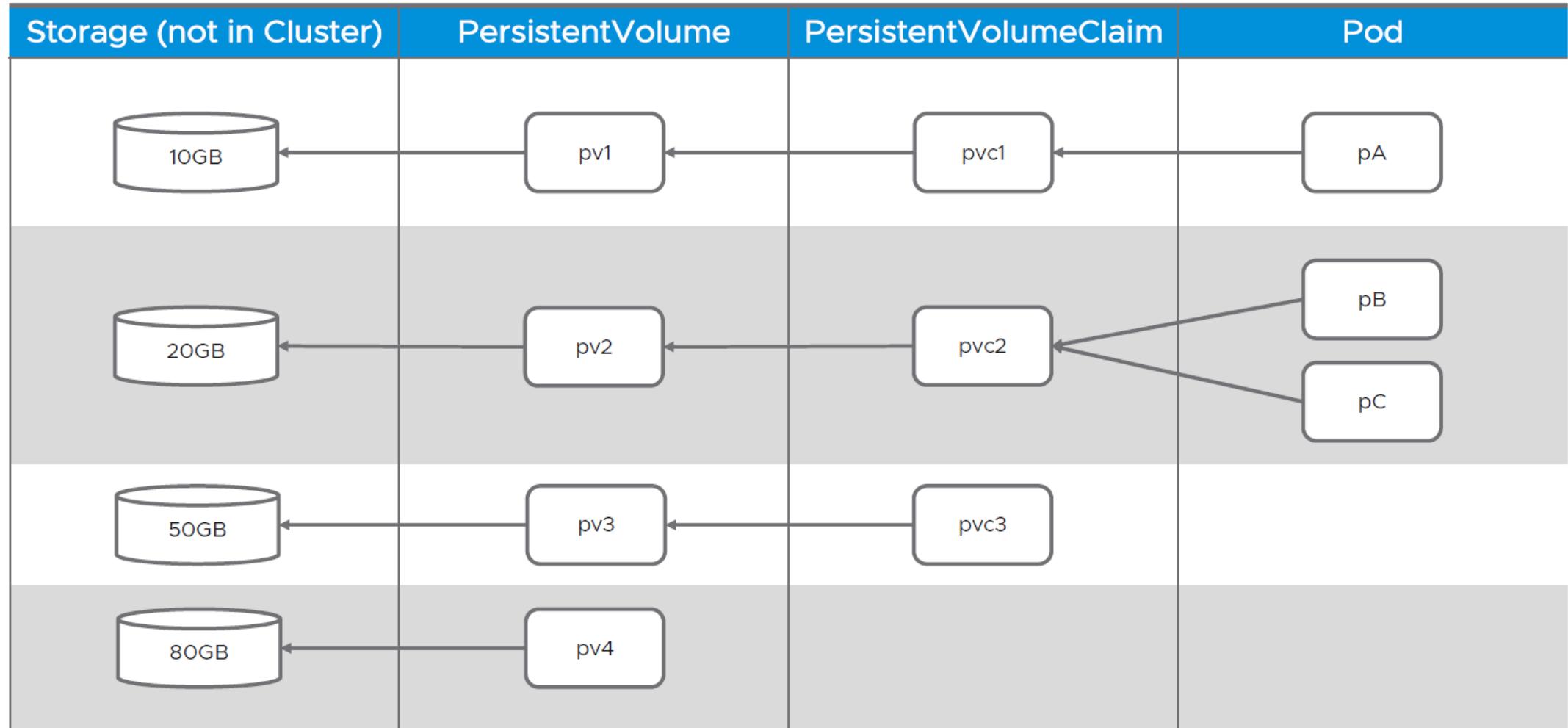
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-vol-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: "500Mi"
~
```

```
[root@localhost ~]# kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pv-vol-claim Bound   pv-vol1   1Gi        RWO          -            30s
[root@localhost ~]# █
```

# Use PVC in Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pv-pod
spec:
  containers:
    - name: my-web-persistent
      image: quay.io/app-sre/nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: mydatavol
          mountPath: "/var/www/html"
  volumes:
    - name: mydatavol
      persistentVolumeClaim:
        claimName: pv-vol-claim
```

# Storage – Relationship Examples



# Configure NFS as Storage

# Configure NFS Server

```
[root@nfs-server ~]# #yum install nfs-utils -y
[root@nfs-server ~]# mkdir /nfsshare
[root@nfs-server ~]# chmod 777 /nfsshare/
[root@nfs-server ~]# echo "/nfsshare *(rw,sync)" >> /etc/exports
[root@nfs-server ~]# systemctl restart nfs
[root@nfs-server ~]# systemctl enable nfs
[root@nfs-server ~]# systemctl stop firewalld
[root@nfs-server ~]# systemctl disable firewalld
[root@nfs-server ~]#
```

# Demo of Persistent Volume with NFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs
spec:
  storageClassName: nfs-storage
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 172.25.230.21
    path: /nfsshare
```

```
[root@k8s-master ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS        CLAIM           STORAGECLASS   REASON   AGE
pv-nfs    1Gi        RWX          Retain          Available
[root@k8s-master ~]#
```

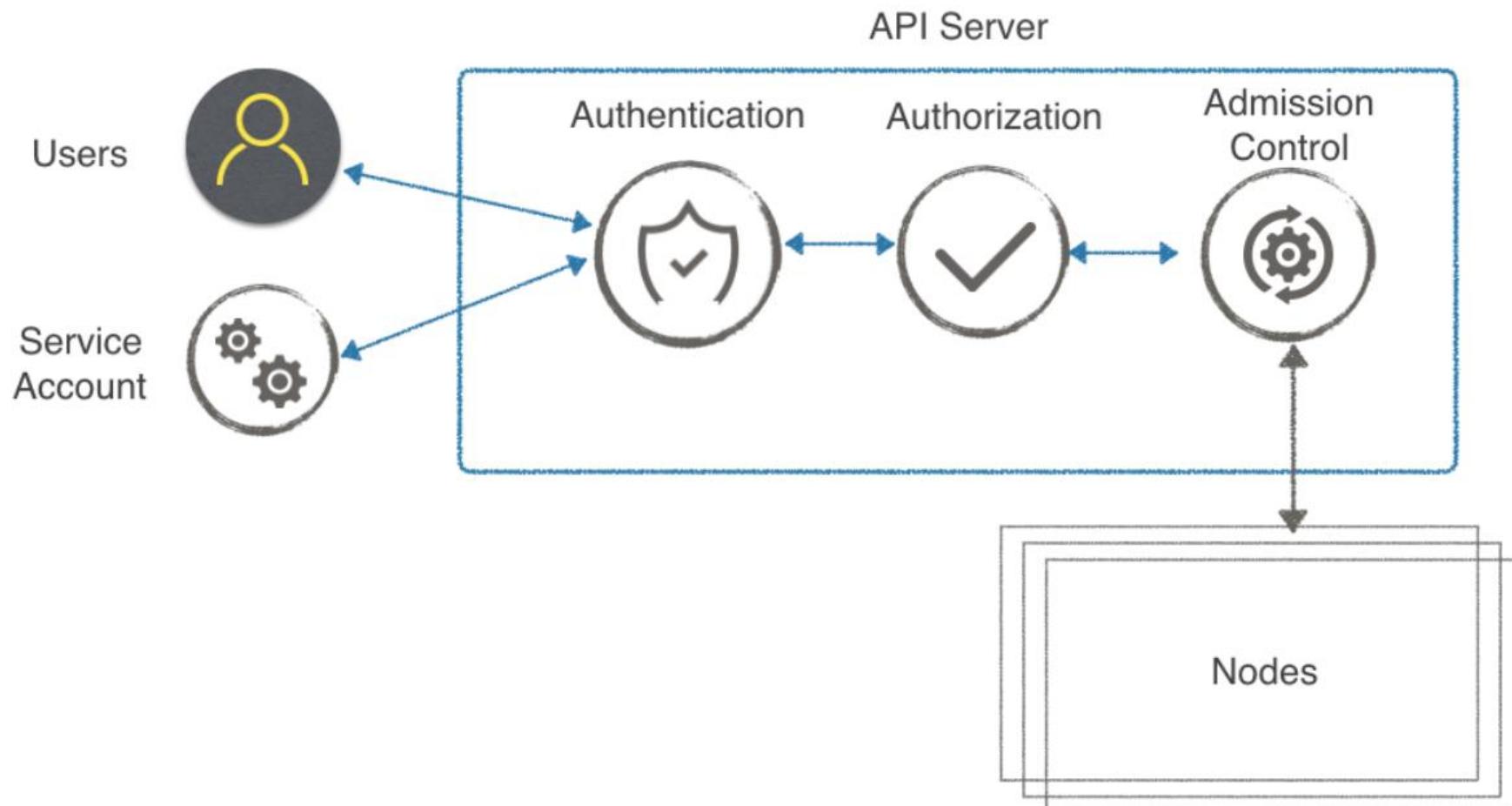
# Demo of Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc1
spec:
  storageClassName: nfs-storage
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 500Mi
```

```
[root@k8s-master ~]# kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-pvc1  Bound     pv-nfs   1Gi        RWX          nfs-storage   41s
[root@k8s-master ~]# █
```

# Security

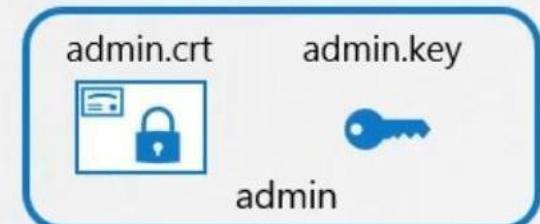
# Kubernetes Authentication





# CERTIFICATE AUTHORITY (CA)

## Client Certificates for Clients



scheduler.crt    scheduler.key



controller-  
manager.crt    controller-  
manager.key



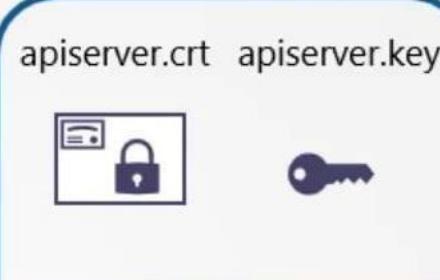
kube-proxy.crt    kube-proxy.key



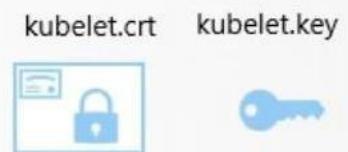
## Server Certificates for Servers



ETCD SERVER



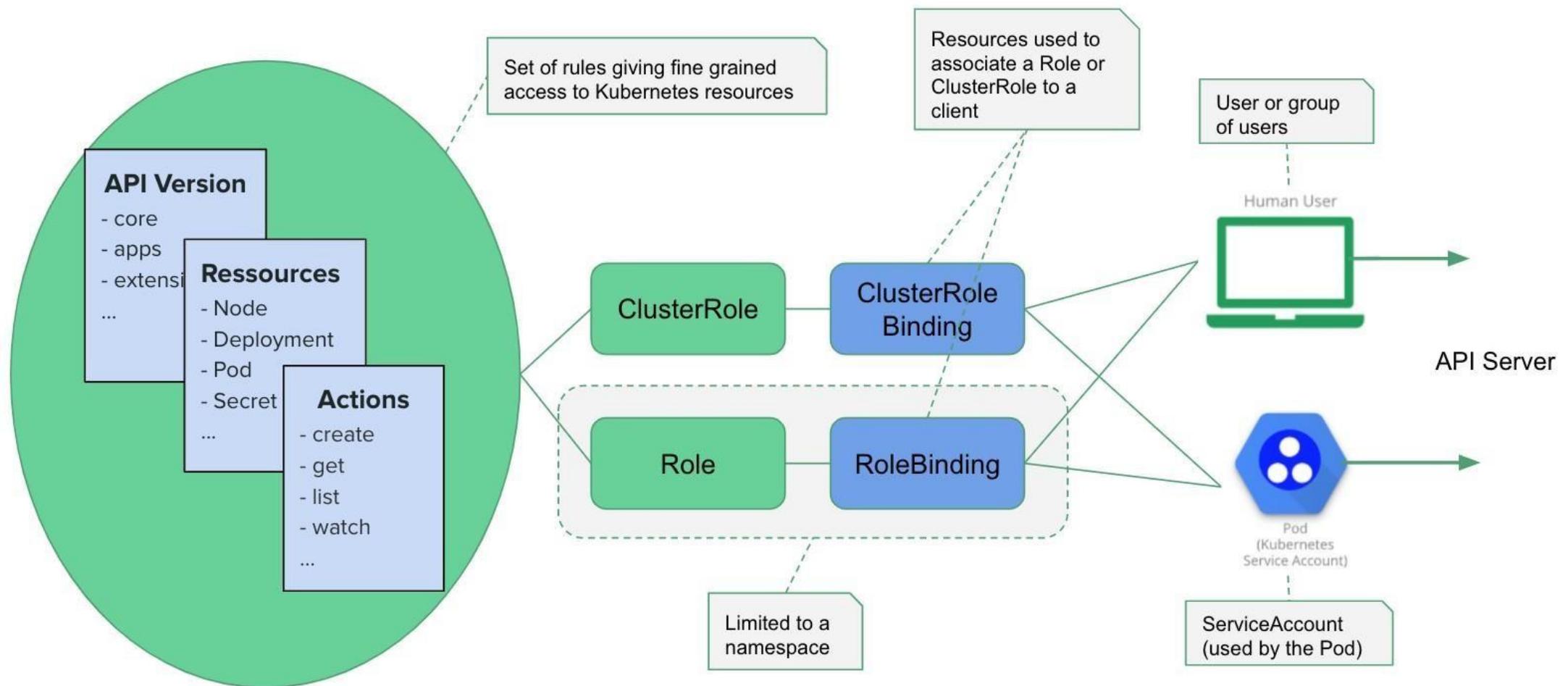
KUBE-API SERVER



KUBELET SERVER

# Managing Users in Kubernetes

# Types of Users in Kubernetes



# User Accounts VS Service Accounts

User Accounts	Service Accounts
User accounts for human	Service Accounts for processes, which run in pods
User accounts are intended to be global. Names must be unique across all namespaces of a cluster, future user resource will not be namespaced.	Service accounts are namespaced.
cluster's User accounts might be synced from a corporate database, where new user account creation requires special privileges and is tied to complex business processes.	Service account creation is intended to be more lightweight, allowing cluster users to create service accounts for specific tasks (i.e. principle of least privilege).

# Important

**When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace for authentication.**

```
#kubectl get sa
```

# Creating Service Account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sales-service-account
  namespace: sales
```

**#kubectl get sa -n sales**

**#kubectl describe sales-service-account -n sales**

# Describe Service Account

```
[root@master test]# kubectl describe sa sales-service-account -n sales
Name:                  sales-service-account
Namespace:             sales
Labels:                <none>
Annotations:           <none>
Image pull secrets:   <none>
Mountable secrets:    sales-service-account-token-zkb6s
Tokens:                sales-service-account-token-zkb6s
Events:                <none>
[root@master test]#
```

**Each service account creates a secrets token which is used to authenticate with API server.**

# Pod with Service Account

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: sales
spec:
  containers:
  - name: pod1
    image: nginx
  serviceAccountName: sales-service-account
```

# Create User

```
#git clone https://github.com/gauravkumar9130/kubeuser
```

```
#cd kubeuser
```

```
#vim user_script.sh
```

**Go to line no 35 and remove ens192 and type your connection name.**

```
#mkdir <username>
```

```
COPY SCRIPT FILE TO <username> DIRECTORY
```

```
#chmod a+x user_script.sh
```

```
#sh user_script.sh
```

# Authorization

In Kubernetes, you must be authenticated (logged in) before your request can be authorized (granted permission to access).

# Authorization Modes

- **Node Based**
  - grants permissions to kubelets based on the pods they are scheduled to run
- **Attribute Based Access Control (ABAC)**
  - Manual Manage Permissions in API
- **Role Based Access Control (RBAC)**
  - Roles and Role Bindings
- **Webhook**
  - manage third party authentication

# Kube API Server YAML

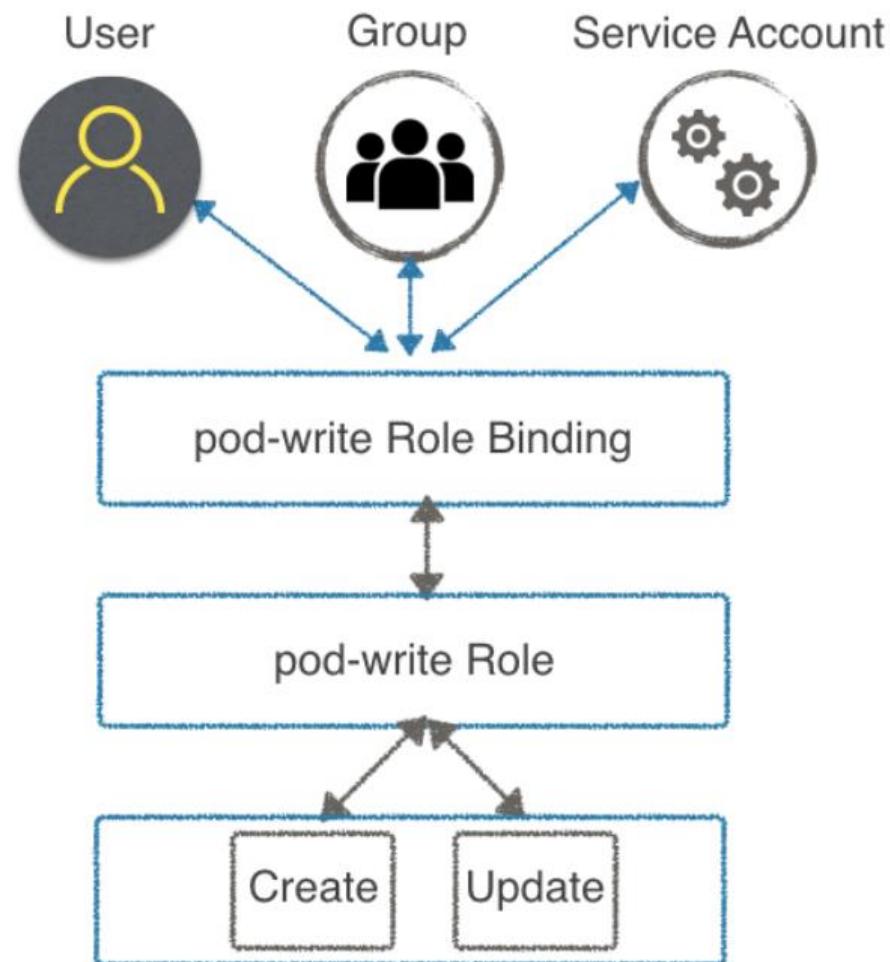
```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 172.25.230.143:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.25.230.143
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
```

# Role Based Access Control (RBAC)

# RBAC

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

# RBAC



# Create Role

**To create role:**

```
#kubectl create role <rolename> --verb=get,list --resource=pods,services
```

**To create role for list pods (only for pod1 and pod2):**

```
#kubectl create role <rolename> --verb=get,list --resource=pods --  
resource-name=pod1 --resource-name=pod2
```

# Manage Role

**To list roles:**

```
#kubectl get roles
```

**To describe role:**

```
#kubectl describe role <rolename>
```

**To delete role:**

```
#kubectl delete role <rolename>
```

# Create Role Binding (Assign Permission)

**To assign role to user:**

```
#kubectl create rolebinding <rolebindingname> --role=<rolename> --  
user=<username>
```

**To assign role to service account:**

```
#kubectl create rolebinding <rolebindingname> --role=<rolename> --  
serviceaccount=<nsname>:<sa name>
```

# Manage Role Binding

**To list roles:**

```
#kubectl get rolebinding
```

**To describe role:**

```
#kubectl describe rolebinding <rolebindingname>
```

**To delete role:**

```
#kubectl delete rolebinding <rolebindingname>
```

# Verify Permission

**To check current logged in user permission:**

```
#kubectl auth can-i <verb> <resource>
```

**To check permission for other user:**

```
#kubectl auth can-i <verb> <resource> --as <username>
```

# IMPORTANT NOTE

**Roles and Rolebindings are restricted to namespace.**

**If we want to give access cluster wide then we will use  
clusterrole and clusterrolebinding.**

# To check Resources

**To check all namespaced resources (role):**

```
#kubectl api-resources --namespaced=true
```

**To check all non-namespaced resources (clusterrole):**

```
#kubectl api-resources --namespaced=false
```

# Create Cluster Role

**To create cluster role:**

```
#kubectl create clusterrole <name> --verb=get,list --  
resource=pods,services
```

# Manage Cluster Role

**To list Cluster Roles:**

```
#kubectl get clusterroles
```

**To describe Cluster Role:**

```
#kubectl describe clusterrole <name>
```

**To delete Cluster Role:**

```
#kubectl delete clusterrole <name>
```

# Create Cluster Role Binding (Assign Permission)

**To assign Cluster Role to user:**

```
#kubectl create clusterrolebinding <name> --clusterrole=<clusterrolename> --  
user=<username>
```

**To assign Cluster Role to service account:**

```
#kubectl create clusterrolebinding <name> --clusterrole=<clusterrolename> --  
serviceaccount=<nsname>:<sa name>
```

**To assign Cluster Admin Role:**

```
#kubectl create clusterrolebinding <name> --clusterrole=cluster-admin --  
user=<username>
```

# Manage Cluster Role Binding

**To list Cluster Role Binding:**

```
#kubectl get clusterrolebinding
```

**To describe Cluster Role Binding:**

```
#kubectl describe clusterrolebinding <clusterrolebindingname>
```

**To delete Cluster Role Binding:**

```
#kubectl delete clusterrolebinding <clusterrolebindingname>
```

# Security Context

# Security Context

Security Context facilitates management of access rights, privileges, and permissions for processes and filesystems in Kubernetes.

# Default Docker Capabilities

**CHOWN, DAC\_OVERRIDE, FSETID, FOWNER, MKNOD, NET\_RAW,  
SETGID, SETUID, SETFCAP, SETPCAP, NET\_BIND\_SERVICE, SYS\_CHROOT,  
KILL, AUDIT\_WRITE**

Doc: <https://opensource.com/business/15/3/docker-security-tuning>

# Default CRI-O Capabilities

**CHOWN, DAC\_OVERRIDE, FSETID, FOWNER, SETGID , SETUID, SETPCAP,  
NET\_BIND\_SERVICE, KILL**

Doc: <https://github.com/cri-o/cri-o/blob/master/docs/crio.conf.5.md>

# Define Capabilities

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: abc
    image: quay.io/gauravkumar9130/nginxdemo
    securityContext:
      capabilities:
        add: ["NET_RAW"]
```

# Verify Capabilities

```
[root@k8s-worker2 ~]# crictl inspect f9402dd292e41 | grep -A 10 capabilities
  "capabilities": {
    "bounding": [
      "CAP_NET_RAW",
      "CAP_CHOWN",
      "CAP_DAC_OVERRIDE",
      "CAP_FSETID",
      "CAP_FOWNER",
      "CAP_SETGID",
      "CAP_SETUID",
      "CAP_SETPCAP",
      "CAP_NET_BIND_SERVICE",
[root@k8s-worker2 ~]# █
```

# Network Policy

# Network Policy

Network policies are Kubernetes resources that control the traffic between pods and/or network endpoints. They use labels to select pods and specify the traffic that is directed toward those pods using rules.

**The most popular CNI plugins with network policy support are:**

- Weave
- Calico
- Cilium
- Kube-router
- Romana

# Isolated and Non-Isolated Pods

- Pods are non-isolated (default)
  - Accept traffic from any source
- Pods become isolated by:
  - Defining a NetworkPolicy that selects them in a Namespace
  - Pod will reject any connections that are not explicitly allowed by a NetworkPolicy
  - Other Pods in the Namespace that are not selected by any NetworkPolicy will continue to accept all traffic

# Network Policy Example - Inbound

```
apiVersion:  
networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: test-network-policy  
  namespace: default  
spec:  
  podSelector:  
    matchLabels:  
      tier: backend  
  policyTypes:  
  - Ingress
```

```
ingress:  
  - from:  
    - namespaceSelector:  
        matchLabels:  
          project: gowebapp  
    - podSelector:  
        matchLabels:  
          app: gowebapp  
  ports:  
  - protocol: TCP  
    port: 3306
```

# pods in gowebapp namespace OR are labeled gowebapp

# Network Policy Example – Inbound (Improved)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
  - Ingress
```

```
ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: gowebapp
  - from:
    - podSelector:
        matchLabels:
          tier: backend
  ports:
  - protocol: TCP
    port: 3306
```

# pods in gowebapp namespace AND are labeled with tier backend

# Network Policy Example – Inbound and Outbound

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
    - Ingress
    - Egress
```

```
ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: gowebapp
  - from:
    - podSelector:
        matchLabels:
          tier: backend
  ports:
    - protocol: TCP
      port: 3306
egress: #allow the DB to connect to LDAP for auth
  - to
    - ipBlock:
        cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 389
```

# Networking

# Custom DNS For Container

```
apiVersion: v1
kind: Pod
metadata:
  name: custom-dns-pod
spec:
  hostAliases:
    - ip: "127.0.0.1"
      hostnames:
        - "localhost"
        - "gaurav.example.com"
    - ip: "172.25.230.111"
      hostnames:
        - "remote.example.com"
  containers:
    - name: dns-hosts
      image: nginx
```

# Verify

```
[root@master ~]# kubectl exec -it custom-dns-pod -- bash
root@custom-dns-pod:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.244.235.132  custom-dns-pod

# Entries added by HostAliases.
127.0.0.1      localhost      gaurav.example.com
172.25.230.111  remote.example.com
root@custom-dns-pod:/# █
```

# Container Network Interface (CNI)

Kubernetes does not provide any default network implementation, rather it only defines the model and leaves to other tools to implement it. We can use **Calico** Solution.

**Note:** Currently Calico is the only CNI plugin that the kubeadm project performs e2e tests against. If you find an issue related to a CNI plugin you should log a ticket in its respective issue tracker instead of the kubeadm or kubernetes issue trackers.

Doc: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>

# Network Solutions

Doc: <https://kubernetes.io/docs/concepts/cluster-administration/networking/>

# Ingress

Ingress is an API object that manages external access to the services in a cluster (typically HTTP).

Ingress isn't a type of Service, but rather an object that acts as a reverse proxy and single entry-point to your cluster that routes the request to different services. The most basic Ingress is the NGINX Ingress Controller, where the NGINX takes on the role of reverse proxy, while also functioning as SSL.

Traffic



Ingress

foo.mydomain.com

mydomain.com/bar

Other

Service

Service

Service

Pod

Pod

Pod

Pod

Pod

Pod

Pod

Pod

Pod

Kubernetes cluster

# How to Deploy Ingress

```
yum install -y git
git clone https://github.com/nginxinc/kubernetes-ingress
cd kubernetes-ingress/
git checkout v1.8.1
cd deployments/
kubectl apply -f common/ns-and-sa.yaml
kubectl apply -f common/default-server-secret.yaml
kubectl apply -f common/nginx-config.yaml
kubectl apply -f rbac/rbac.yaml
kubectl apply -f daemon-set/nginx-ingress.yaml
```

# Verify Installation

```
[root@k8s-master ~]# kubectl get pods -n nginx-ingress
NAME                  READY   STATUS    RESTARTS   AGE
nginx-ingress-dwb4j  1/1     Running   0          52s
nginx-ingress-n96br  1/1     Running   0          52s
[root@k8s-master ~]# █
```

# Hotel Application

# Application

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hotel
spec:
  replicas: 5
  selector:
    matchLabels:
      app: hotel
  template:
    metadata:
      labels:
        app: hotel
    spec:
      containers:
        - name: hotel-container
          image: quay.io/gauravkumar9130/mywebapp
```

# Service for Hotel Application

```
apiVersion: v1
kind: Service
metadata:
  name: hotel-svc
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: hotel
```

# Ingress for Hotel Application

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hotel-ingress
spec:
  rules:
    - host: hotel.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: hotel-svc
                port:
                  number: 80
```

# Ingress for Hotel Application

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hotel-ingress
spec:
  rules:
    - host: hotel.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: hotel-svc
                port:
                  number: 80
```

# Verify

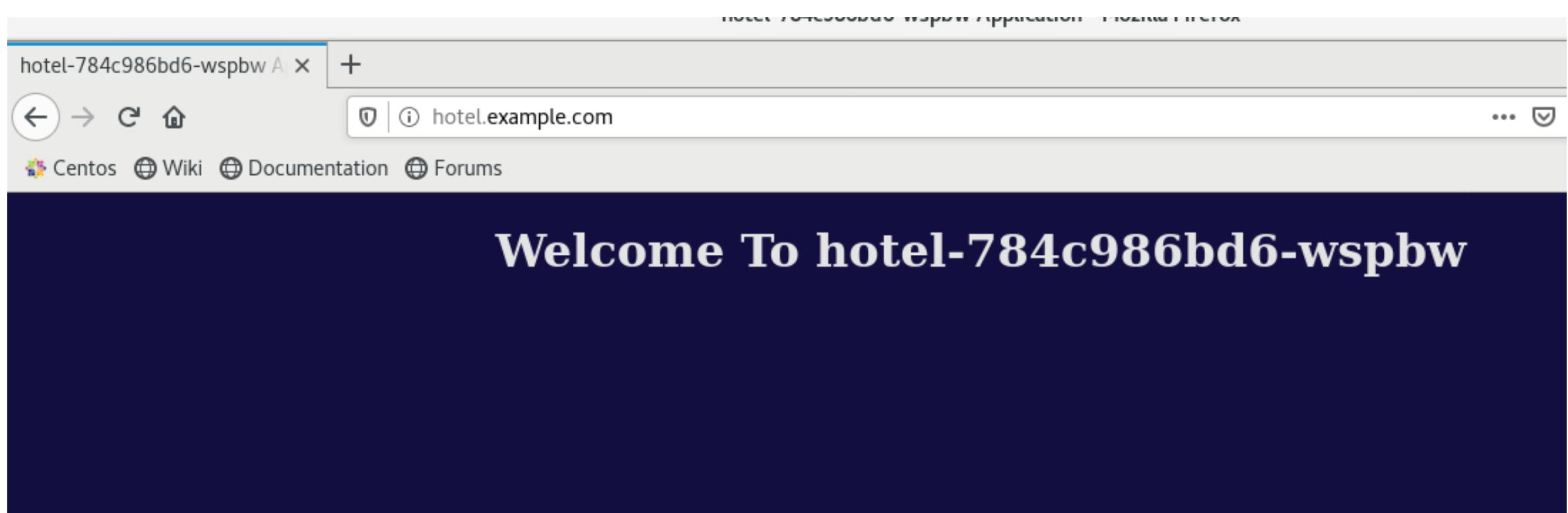
```
#kubectl get pods,svc
```

**Add entry in host file because of we have no DNS Server:**

```
#vim /etc/hosts
```

```
<ip of worker> hotel.example.com
```

# Verify



# Metal Load Balancer

# Why?

- Kubernetes does not offer an implementation of network load-balancers (Services of type LoadBalancer) for bare metal clusters.
- LoadBalancers will remain in the “pending” state indefinitely when created.

# Metal Load Balancer

- MetallLB hooks into your Kubernetes cluster, and provides a network load- balancer implementation. In short, it allows you to create Kubernetes services of type “LoadBalancer” in clusters.

# Prerequisites

```
#kubectl edit configmap kube-proxy -n kube-system
```

```
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: "ipvs"
ipvs:
    strictARP: true
```

Doc: <https://metallb.universe.tf/installation/>

# Installation by Manifests

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/metallb.yaml
# On first install only
kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64
128)"
```

Doc: <https://metallb.universe.tf/installation/>

# Create ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
```

Doc: <https://metallb.universe.tf/configuration>

# Create ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 172.25.230.1-172.25.230.5
```

Doc: <https://metallb.universe.tf/configuration>

# Components in Manifests

- The `metallb-system/controller` deployment. This is the cluster-wide controller that handles IP address assignments.
- The `metallb-system/speaker` daemonset. This is the component that speaks the protocol(s) of your choice to make the services reachable.
- Service accounts for the controller and speaker, along with the RBAC permissions that the components need to function.

# Verify

```
[root@k8s-master ~]# kubectl get pods -n metallb-system
NAME                  READY   STATUS    RESTARTS   AGE
controller-65db86ddc6-62lr9   1/1     Running   0          4m22s
speaker-8xdfm           1/1     Running   0          4m22s
speaker-dxfpc           1/1     Running   0          4m22s
speaker-nr998            1/1     Running   0          4m22s
[root@k8s-master ~]#
```

# Deploy Application

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapplication
spec:
  replicas: 5
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web-container
          image: quay.io/gauravkumar9130/mywebapp
```

# Deploy Service

```
apiVersion: v1
kind: Service
metadata:
  name: web-svc
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: web
  type: LoadBalancer
```

# Verify

```
[root@k8s-master ~]# kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1       <none>          443/TCP       21h
web-svc      LoadBalancer   10.109.169.216  172.25.230.1   80:32264/TCP  14s
[root@k8s-master ~]# █
```

# Logging and Monitoring

# Logs

**What are the kinds of logs we want to capture?**

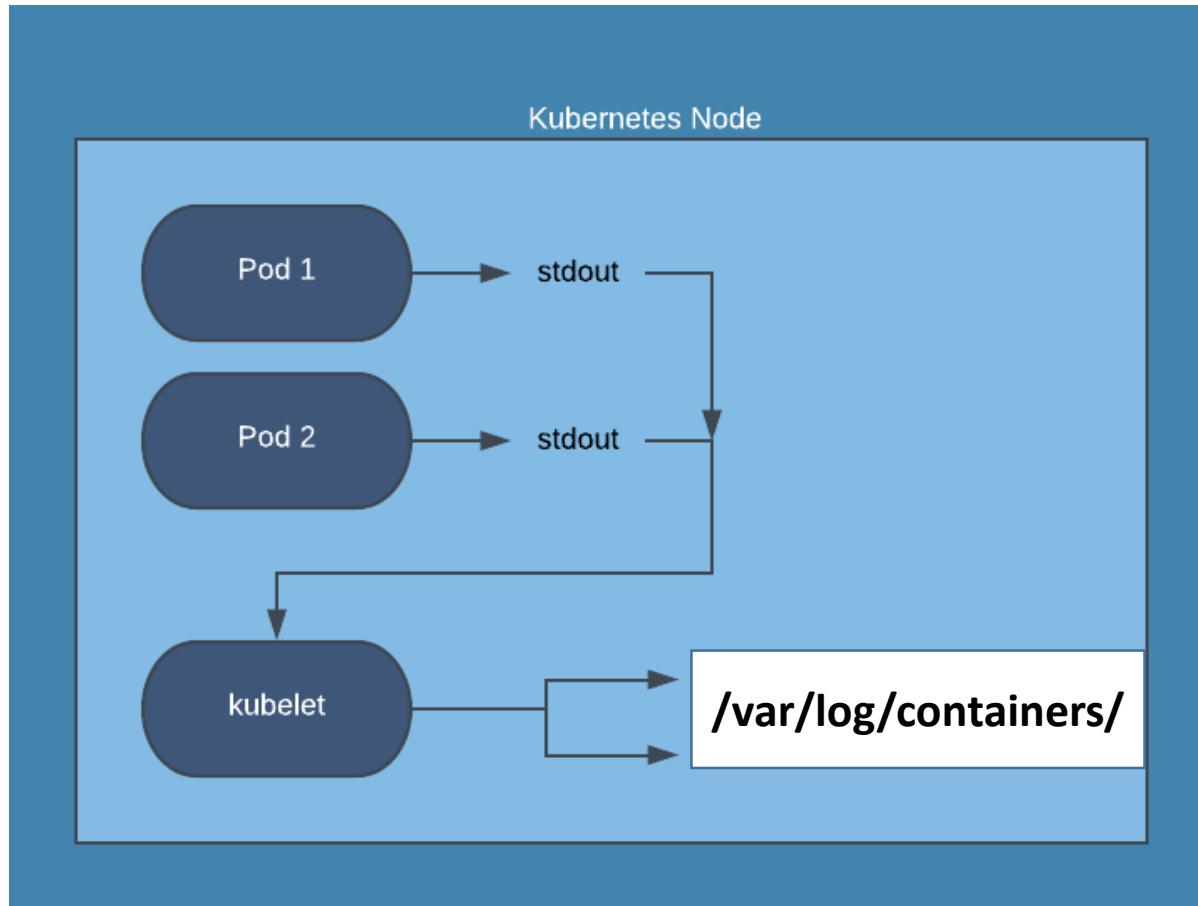
- Container Logs
- Host OS Logs
- Control Plane Logs
- Event Messages

# Check Logs

```
#kubectl logs <podname>
```

```
#kubectl logs <podname> -c <container name>
```

# How Kubernetes Logging Works?



# Monitoring Tools

- **DIY - Free**
  - Kube-state-metrics- collects metrics from Kubernetes API
  - Prometheus
  - metricbeat – standard metric monitoring that is Kubernetes aware
  - Ganglia
  - And more.....
- **Third Party Services – Paid**
  - Datadog
  - Honeycomb
  - Stackdriver with google monitoring
  - And more...

# Prometheus

## Download and Install Prometheus:-

```
#git clone https://github.com/gauravkumar9130/prometheus.git  
#cd prometheus  
#kubectl create -f prometheus.yaml  
#kubectl create -f kube-state-metrics-configs/.
```

# Verify

```
[root@k8s-master ~]# kubectl get pods -n monitoring
NAME                               READY   STATUS    RESTARTS   AGE
prometheus-deployment-86969d558c-rbvrg   1/1     Running   0          55s
[root@k8s-master ~]# kubectl get svc -n monitoring
NAME         TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
prometheus-service   NodePort    10.105.48.175    <none>        8080:30000/TCP   60s
[root@k8s-master ~]# █
```

# Prometheus Console

The screenshot shows the Prometheus Console interface. At the top, there is a header bar with the title "Prometheus Time Series" and a "localhost:30000/graph" URL. Below the header are links for "Centos", "Wiki", "Documentation", and "Forums". The main area contains a query editor with a checkbox for "Enable query history" and a text input field for "Expression (press Shift+Enter for newlines)". Below the expression input is a blue "Execute" button and a dropdown menu with the placeholder "- insert metric at cursor -". There are two tabs at the bottom: "Graph" (which is selected) and "Console". At the bottom of the screen, there are navigation controls for "Moment" (with arrows for previous and next), a search bar, and a footer with the Prometheus logo.

Prometheus Time Series X +

localhost:30000/graph

Centos Wiki Documentation Forums

Prometheus Alerts Graph Status Help

Enable query history

Expression (press Shift+Enter for newlines)

Execute - insert metric at cursor -

Graph Console

<> Moment >>

# Deploy Metrics Server

**Download and Install Prometheus:-**

```
#git clone https://github.com/gauravkumar9130/metrics-server.git
```

```
#cd metrics-server
```

```
#kubectl create -f .
```

# Verify

```
[root@k8s-master ~]# kubectl top node
NAME          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master    208m        10%   2250Mi        28%
k8s-worker1   142m        7%   2372Mi        30%
k8s-worker2   164m        8%   2032Mi        25%
[root@k8s-master ~]# kubectl top pod
NAME      CPU(cores)   MEMORY(bytes)
myapp    0m           1Mi
[root@k8s-master ~]# █
```

# Deploy Kubernetes Dashboard

```
#kubectl apply -f  
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml
```

---

```
[root@k8s-master ~]# kubectl get deployments -n kubernetes-dashboard  
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE  
dashboard-metrics-scraper   1/1      1           1          5m37s  
kubernetes-dashboard       1/1      1           1          5m37s  
[root@k8s-master ~]# █
```

# Deploy Kubernetes Dashboard

```
[root@k8s-master ~]# kubectl get sa -n kubernetes-dashboard
NAME                  SECRETS   AGE
default               1          113s
kubernetes-dashboard  1          113s
[root@k8s-master ~]# █
```

**Give cluster admin access to kubernetes-dashboard SA:**

```
#kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --
service-account=kubernetes-dashboard:kubernetes-dashboard
```

# Deploy Kubernetes Dashboard

**Create node port service for Kubernetes Dashboard:**

```
#kubectl expose deployment kubernetes-dashboard --type=NodePort --  
name=dashboard-svc -n kubernetes-dashboard
```

```
[root@k8s-master ~]# kubectl get svc -n kubernetes-dashboard  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)        AGE  
dashboard-metrics-scraper  ClusterIP  10.97.126.17 <none>       8000/TCP      9m12s  
dashboard-svc    NodePort   10.98.118.205  <none>       8443:31349/TCP 78s  
kubernetes-dashboard  ClusterIP  10.108.18.51   <none>       443/TCP      9m12s  
[root@k8s-master ~]# █
```

Kubernetes Dashboard x +

← → ⟳ HomeAs ... ⋮ 🔗 ☆

Centos Wiki Documentation Forums

https://localhost:31349/#/login

## Kubernetes Dashboard

Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the [Authentication](#) section.

Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the [Configure Access to Multiple Clusters](#) section.

Enter token \*

---

Sign in

# Deploy Kubernetes Dashboard

```
[root@k8s-master ~]# kubectl get secrets -n kubernetes-dashboard
NAME                           TYPE              DATA  AGE
default-token-pr9mp           kubernetes.io/service-account-token 3      11m
kubernetes-dashboard-certs    Opaque            0      11m
kubernetes-dashboard-csrf     Opaque            1      11m
kubernetes-dashboard-key-holder Opaque            2      11m
kubernetes-dashboard-token-7rhjk kubernetes.io/service-account-token 3      11m
[root@k8s-master ~]# █
```

# Deploy Kubernetes Dashboard

```
[root@k8s-master ~]# kubectl describe secrets kubernetes-dashboard-token-7rhjk -n kubernetes-dashboard
Name:           kubernetes-dashboard-token-7rhjk
Namespace:      kubernetes-dashboard
Labels:         <none>
Annotations:   kubernetes.io/service-account.name: kubernetes-dashboard
               kubernetes.io/service-account.uid: 9c0d3697-afa9-45bb-ae9b-8a28895aaf22
Type:          kubernetes.io/service-account-token

Data
=====
token:          eyJhbGciOiJSUzI1NiIsImtpZCI6IjN6dTlfMDdMWmpoMGswcEdZXzZGYzNYdnFJTlB0YV9LQl9r0Us0c3dlcTgifQ.e
yJpc3Mi0iJrdWJlc5ldGVzL3NlcZpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pbY9zZXJ2aWNLYWNjb3VudC9uYW1lc3BhY2Ui0iJrd
WJlc5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaw8vc2VydmljZWfjY291bnQvc2VjcmV0Lm5hbWUi0iJrdWJlc5ldGVzLWRhc
2hib2FyZC10b2tlbi03cmhqayIsImt1YmVybmV0ZXMuaw8vc2VydmljZWfjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUi0iJrdWJlc
m5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaw8vc2VydmljZWfjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjljMGQzNjk3L
WFmYTktNDViYi1hZTlilThhMjg40TVhYWYyMiIsInN1YiI6InN5c3RlbTpzZXJ2aWNLYWNjb3VudDprdWJlc5ldGVzLWRhc2hib2FyZ
DprdWJlc5ldGVzLWRhc2hib2FyZCJ9.ZhkXGyiIIxvWCAYw8vQLFGznEaHhu6-1RvxctuhAbxfj6Dbi-7bAt3zf-RJ1y0AyaMqcp_oA
2pPwHhqp5Y3vRfzNqsZkJIwsqglfNMDzN6pmgScpzw3uzPashP0s6UkHhC9sIBxV87fnFT0fKmbu5HowtHtBppmwyatFwb3XTPlNHycy
yeFBVzWz8ol-0uIhfkQdijfkglTrp2DWkCgw0k7oNwpzXYsrHdfLZ333Y0vaTU758PC0En2zTpexIKmXM6oQZ29Hte8mfccym6bsj0WY
c5u6IJpKK7zi_X9qAiXNZtk0-mpwc38NN06q-5GnuE1P8XyUTAHej_FdgS-XSw
ca.crt:        1066 bytes
namespace:     20 bytes
[root@k8s-master ~]# █
```

Kubernetes Dashboard X

Firefox Privacy Notice — X +

! ! https://localhost:31349/#/overview?namespace=default

Centos Wiki Documentation Forums

 kubernetes

default ▼

Search

## ☰ Overview

**Cluster**

- Cluster Roles
- Namespaces
- Nodes
- Persistent Volumes
- Service Accounts N
- Storage Classes

**Workloads**

- Workloads N
- Cron Jobs
- Daemon Sets

**CPU Usage**

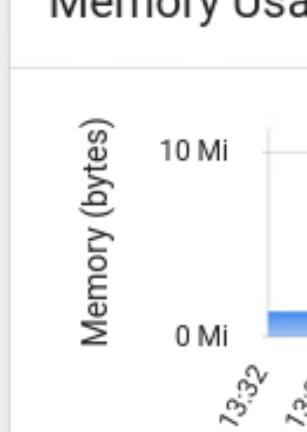


0.01

0

13:32 13:33 13:34 13:35 13:36 13:37 13:38 13:39 13:40 13:41 13:42 13:43 13:44 13:45

**Memory Usage**



10 Mi

0 Mi

13:32 13:33 13:34 13:35 13:36 13:37 13:38 13:39 13:40 13:41 13:42 13:43 13:44 13:45

# Cluster Maintenance

# OS Upgrade

**Before doing os upgradation first drain->**

```
#kubectl drain worker1 --force --ignore-daemonsets
```

Technically containers are not removed(in case of replica) they are gracefully terminated and recreated on another node and our node1 become unschedulable i.e no container can be created on it then we reboot node1 still it is in unscheduable state to make it schedulable

```
#kubectl uncordon worker1
```

# Kubernetes Version

```
[root@k8s-master ~]# kubectl get nodes
NAME        STATUS   ROLES          AGE   VERSION
k8s-master   Ready    control-plane,master   22h   v1.20.2
k8s-worker1  Ready    <none>         22h   v1.20.2
k8s-worker2  Ready    <none>         22h   v1.20.2
[root@k8s-master ~]#
```

**v1 -> major**

**20-> minor (added features and functionalities)**

**2 -> patch (bug fixes)**

# Upgrade Strategies

- 1) upgrade all nodes at once -> but then your pods will be down.**
- 2) upgrade one at a time-> first drain then upgrade. in this case there will be no downtime.**
- 3) add a new upgraded node -> then transfer one nodes data to upgraded node then delete oldest then do same with all**

# Version Compatibility

**All components are same version:**

- Kube-apiserver
- Kube-controller-manager
- Kube-scheduler
- Kubelet
- Kube-proxy
- Kubectl

**Some components have different version:**

- etcd
- coredns

# Version Compatibility

**NOTE: We need to make sure that kube-apiserver version is not lower than other components version.**

# Version Compatibility

## Kube-API Server (v1.20)

Kube-Controller Manager  
(v1.19 OR v1.20)

Kube-Scheduler  
(V1.19 OR V1.20)

Kubelet  
(V1.18 OR V1.19 OR V1.20)

Kube-Proxy  
(V1.18 OR V1.19 OR V1.20)

# NOTE

**NOTE: We can upgrade only one version.**

**Suppose we are on v1.18 and we want to upgrade to v1.20  
so first we need to upgrade to v1.19 then v1.20**

# Upgrade Kubernetes Cluster

**To check new versions:**

```
#kubeadm upgrade plan
```

**To upgrade:**

```
#kubeadm upgrade apply <version>
```

**Above command will upgrade: apiserver, controller manager, scheduler, kube-proxy, coredns, etcd.**

**Note: We need to upgrade kubelet, kubeadm and kubectl manually.**

# Upgrade Kubernetes Cluster

## Steps:

- Upgrade the Control Plane
- Upgrade the Nodes
- Upgrade Clients such as Kubectl

# Upgrade Kubeadm, Kubectl and Kubelet (RPM Based)

First drain node then upgrade.

```
#yum upgrade -y kubeadm-1.20.x kubelet-1.20.x kubectl-1.20.x  
#systemctl daemon-reload  
#systemctl restart kubelet
```

After upgrade uncordon the node.

# Upgrade Kubeadm, Kubectl and Kubelet (Debian Based)

First drain node then upgrade.

```
#apt-mark unhold kubeadm kubectl kubelet
```

```
#apt-get update && apt-get upgrade -y kubeadm=1.20.x  
kubelet=1.20.x kubectl=1.20.x
```

```
#apt-mark hold kubeadm kubelet kubectl
```

```
#systemctl daemon-reload
```

```
#systemctl restart kubelet
```

After upgrade uncordon the node.

# Static Pod

# Static Pod

Static pods are pods created and managed by kubelet daemon on a specific node without API server observing them. If the static pod crashes, kubelet restarts them.

**Static Pod Default Path:** /etc/kubernetes/manifests

**Static Pod Config File:** /var/lib/kubelet/config.yaml

# Demo of Static Pod

```
[root@k8s-worker1 ~]# cd /etc/kubernetes/manifests/  
[root@k8s-worker1 manifests]# cat pod.yml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: mypod  
spec:  
  containers:  
  - name: abc  
    image: quay.io/gauravkumar9130/mywebapp  
[root@k8s-worker1 manifests]#
```

# Verify

---

```
[root@k8s-master ~]# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE
mypod-k8s-worker1  1/1     Running   0          115s   10.244.194.80  k8s-worker1
[root@k8s-master ~]# █
```

# ETCD Backup and Restore

# ETCD Backup

## Require Packages:

```
#yum install epel-release -y  
#yum install etcd -y
```

```
[root@k8s-master ~]# kubectl get pods -n prod  
NAME      READY     STATUS      RESTARTS   AGE  
abc       1/1      Running     0          6m52s  
[root@k8s-master ~]# █
```

# ETCD Backup

```
[root@k8s-master ~]# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/healthcheck-client.crt --key /etc/kubernetes/pki/etcd/healthcheck-client.key snapshot save mysnapshot.db
Snapshot saved at mysnapshot.db
[root@k8s-master ~]# █
```

# ETCD Backup Status

```
[root@k8s-master ~]# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/healthcheck-client.crt --key /etc/kubernetes/pki/etcd/healthcheck-client.key snapshot status mysnapshot.db  
c531f61d, 120384, 1461, 3.6 MB  
[root@k8s-master ~]#
```

# ETCD Restore

---

```
[root@k8s-master ~]# kubectl delete ns prod
namespace "prod" deleted
[root@k8s-master ~]# kubectl get pods -n prod
No resources found in prod namespace.
[root@k8s-master ~]# █
```

# ETCD Restore

```
[root@k8s-master ~]# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/healthcheck-client.crt --key /etc/kubernetes/pki/etcd/healthcheck-client.key snapshot restore mysnapshot.db
2021-02-17 14:54:45.355597 I | mvcc: restore compact to 119421
2021-02-17 14:54:45.366584 I | etcdserver/membership: added member 8e9e05c52164694d [http://localhost:2380] to cluster cdf818194e3a8c32
[root@k8s-master ~]#
```

```
[root@k8s-master ~]# cd default.etcd/
[root@k8s-master default.etcd]# rm -rf /var/lib/etcd/member/
[root@k8s-master default.etcd]# mv member/ /var/lib/etcd/
[root@k8s-master default.etcd]# █
```

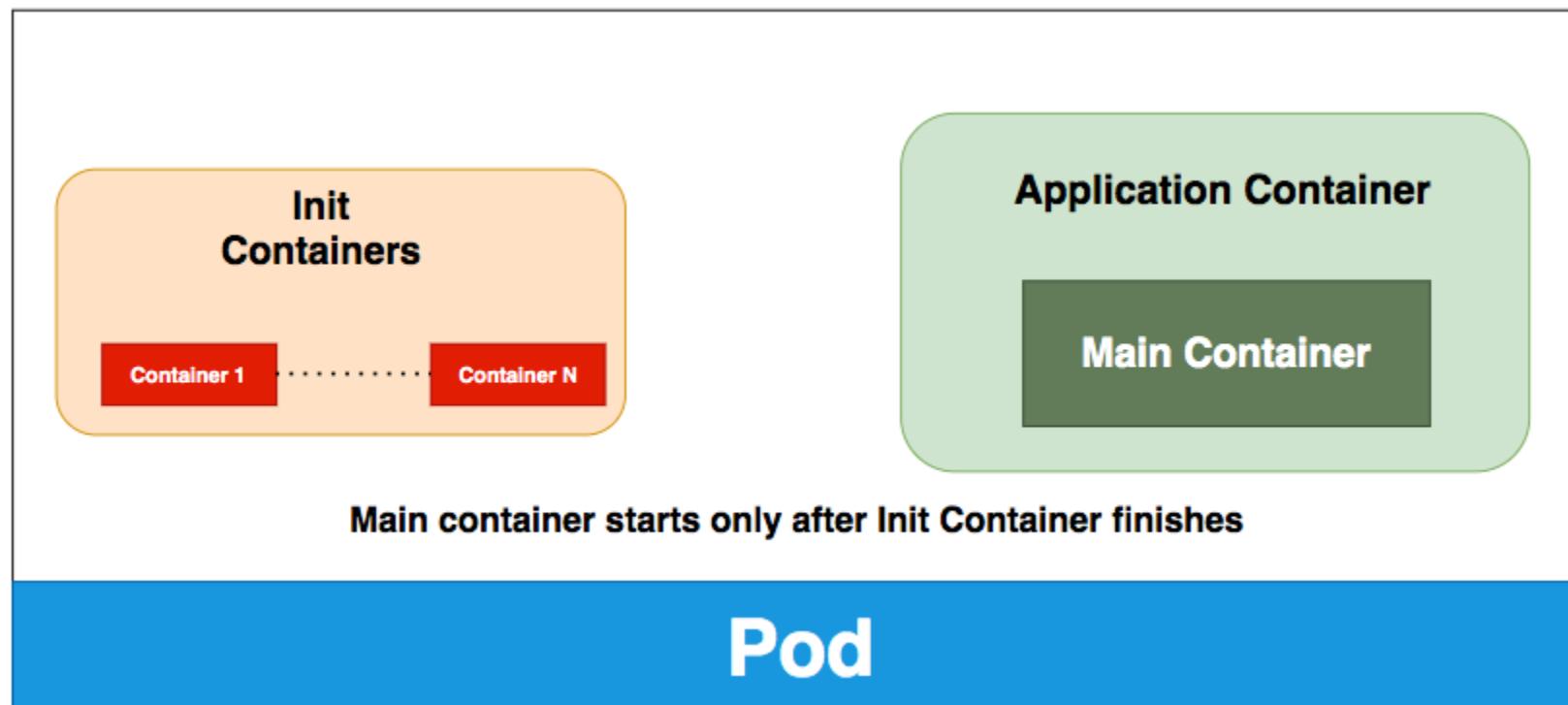
# ETCD Restore

```
[root@k8s-master ~]# kubectl get pods -n prod
NAME      READY     STATUS      RESTARTS      AGE
abc       1/1      Running     0            18m
[root@k8s-master ~]#
```

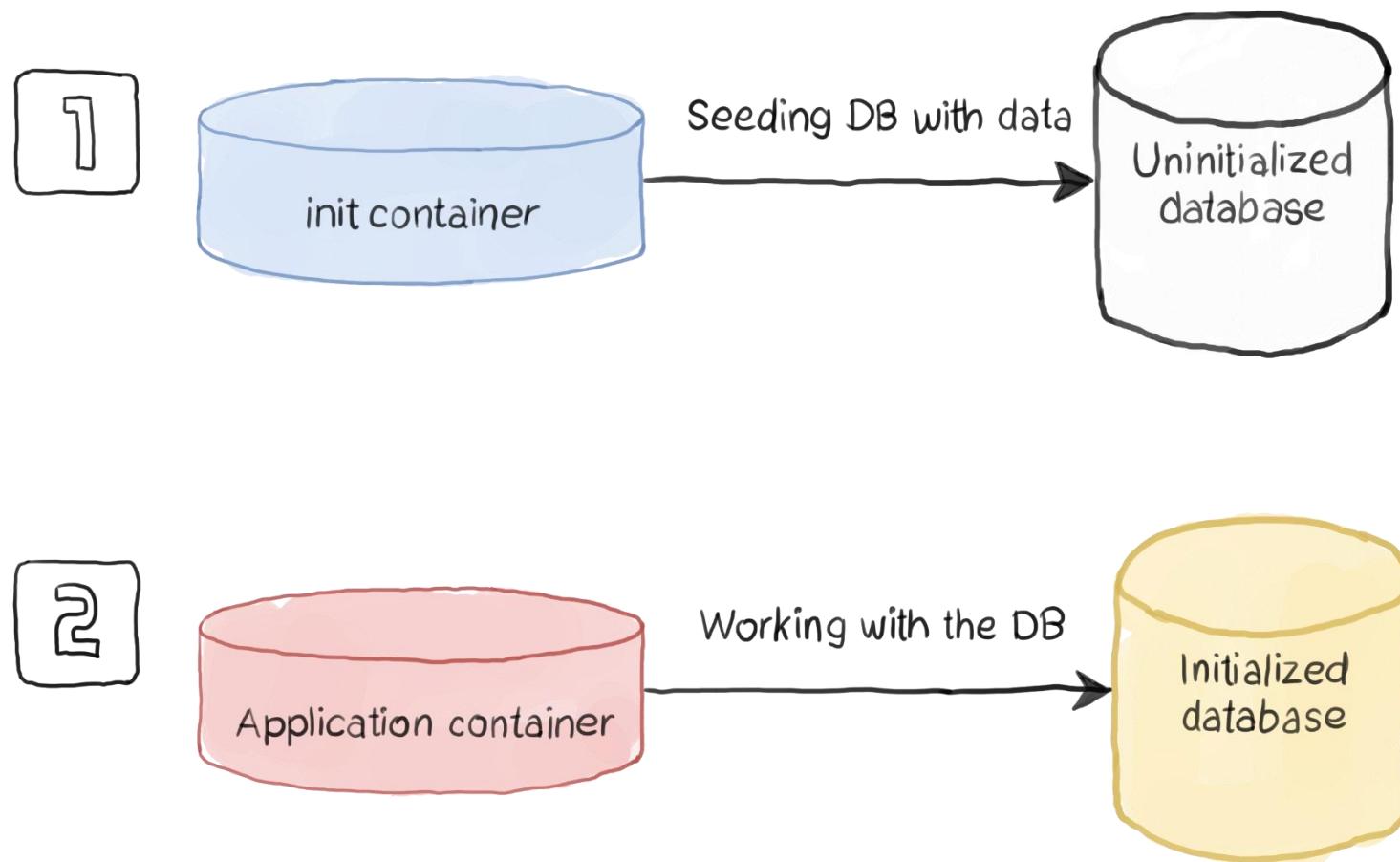
# Init Containers

# Init Container

A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.



# Example of Init Container

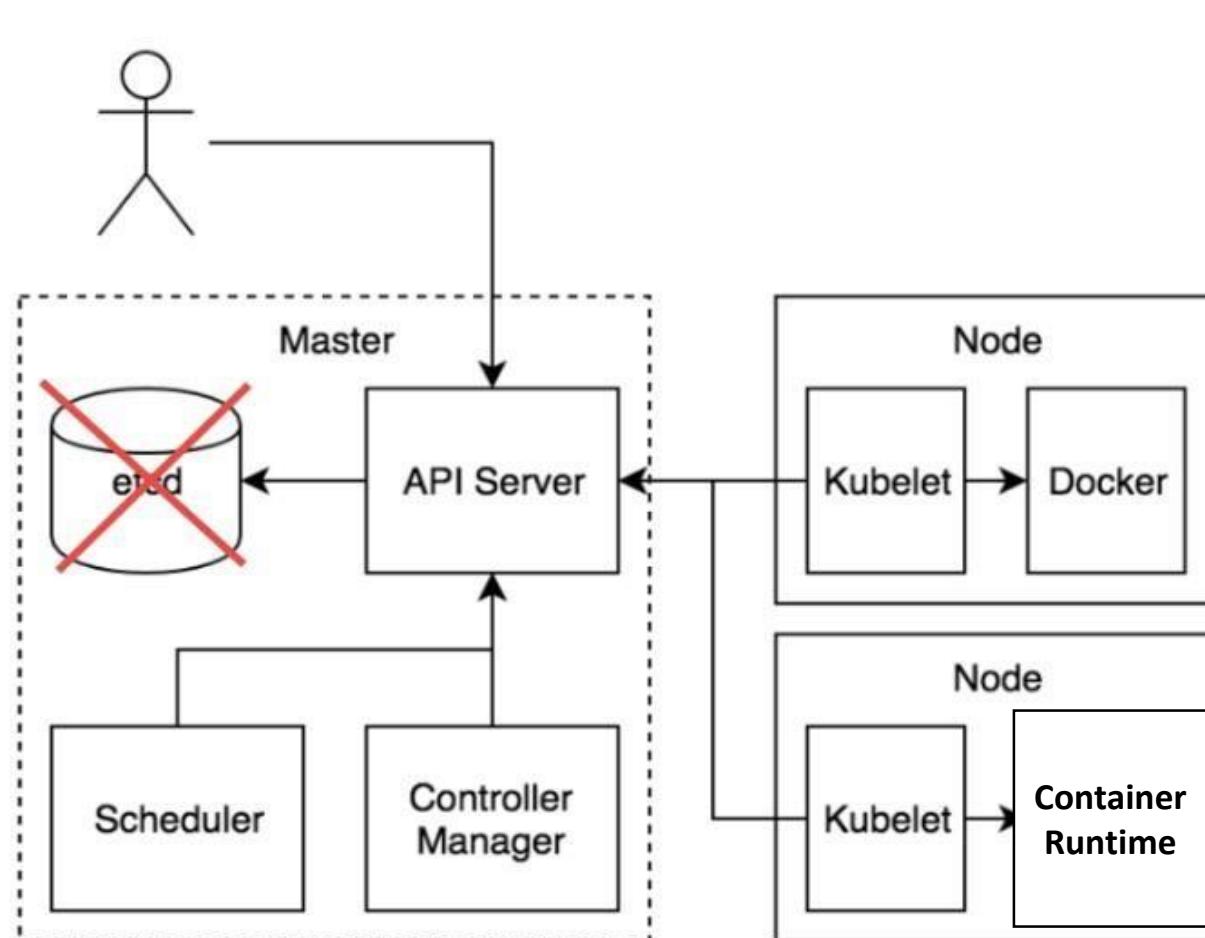


# Demo of Init Container

```
apiVersion: v1
kind: Pod
metadata:
  name: test-sql-pod
spec:
  containers:
    - name: container1
      image: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: mypass
      volumeMounts:
        - name: mysqldb
          mountPath: /mydb
  initContainers:
    - name: initcontainer
      image: busybox:1.28
      command: ["wget","-O","/mydb/docker.repo","https://download.docker.com/linux/centos/docker-ce.repo"]
      volumeMounts:
        - name: mysqldb
          mountPath: /mydb
  volumes:
    - name: mysqldb
      emptyDir: {}
```

# Troubleshooting

# ETCD Lost



# Note

**In a real cluster, this would be caused by etcd crashing or data corruption. In the lab we'll simulate with the command below.**

```
#mv /etc/kubernetes/manifests/etcd.yaml /
```

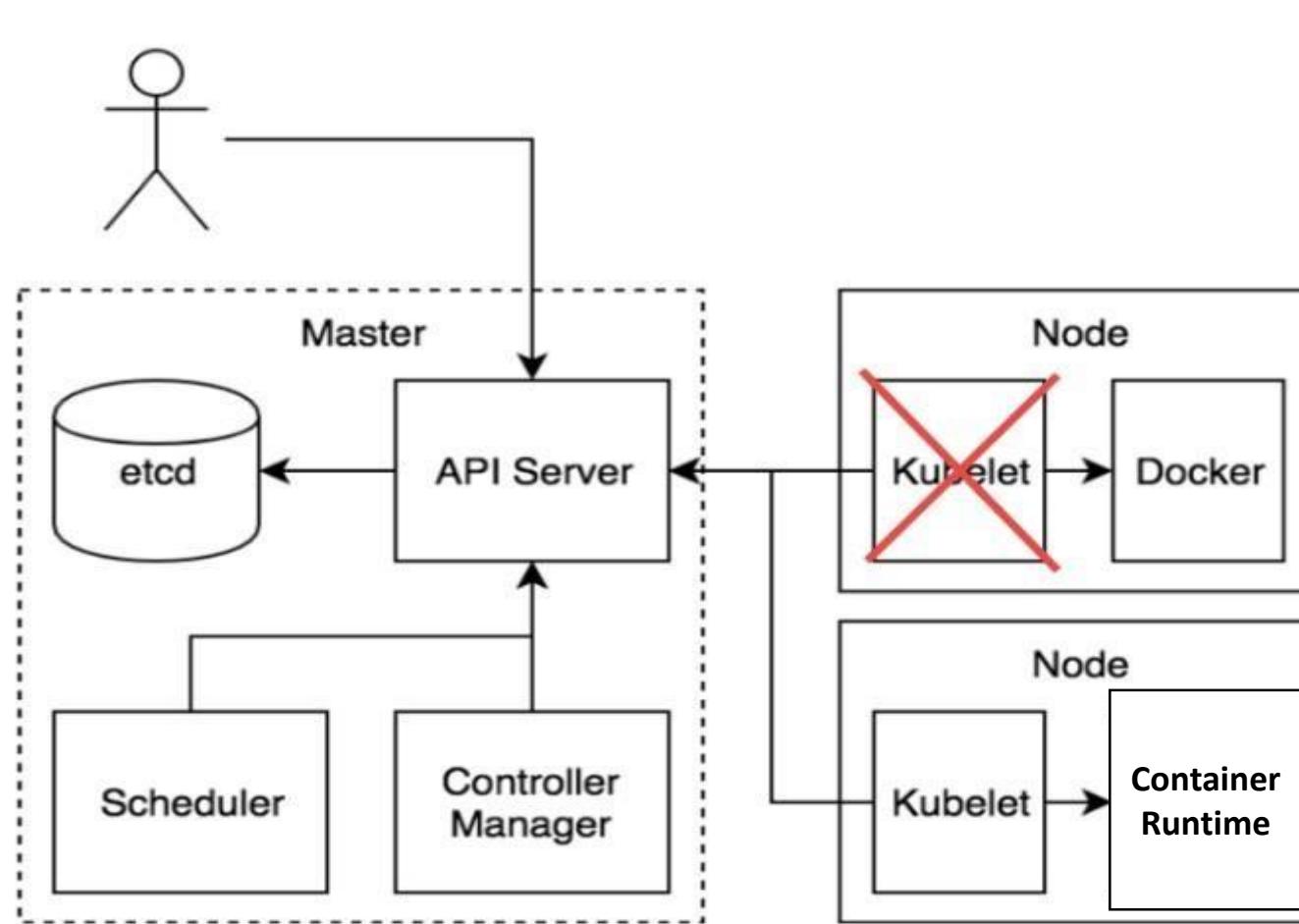
# Symptoms

- apiserver should fail to come up
- kubelet will not be able to reach it but will continue to run existing pods
- manual recovery necessary before apiserver is restarted

# Troubleshooting

```
#journalctl -u kubelet -f  
#systemctl status kubelet  
#crictl ps
```

# Kubelet Issue



## Note

In a real cluster, this would be caused by worker node going offline or kubelet crashing

```
#systemctl stop kubelet (on k8s-worker1)
```

# Symptoms

- unable to stop, update, or start new pods, services, replication controller

# List Nodes

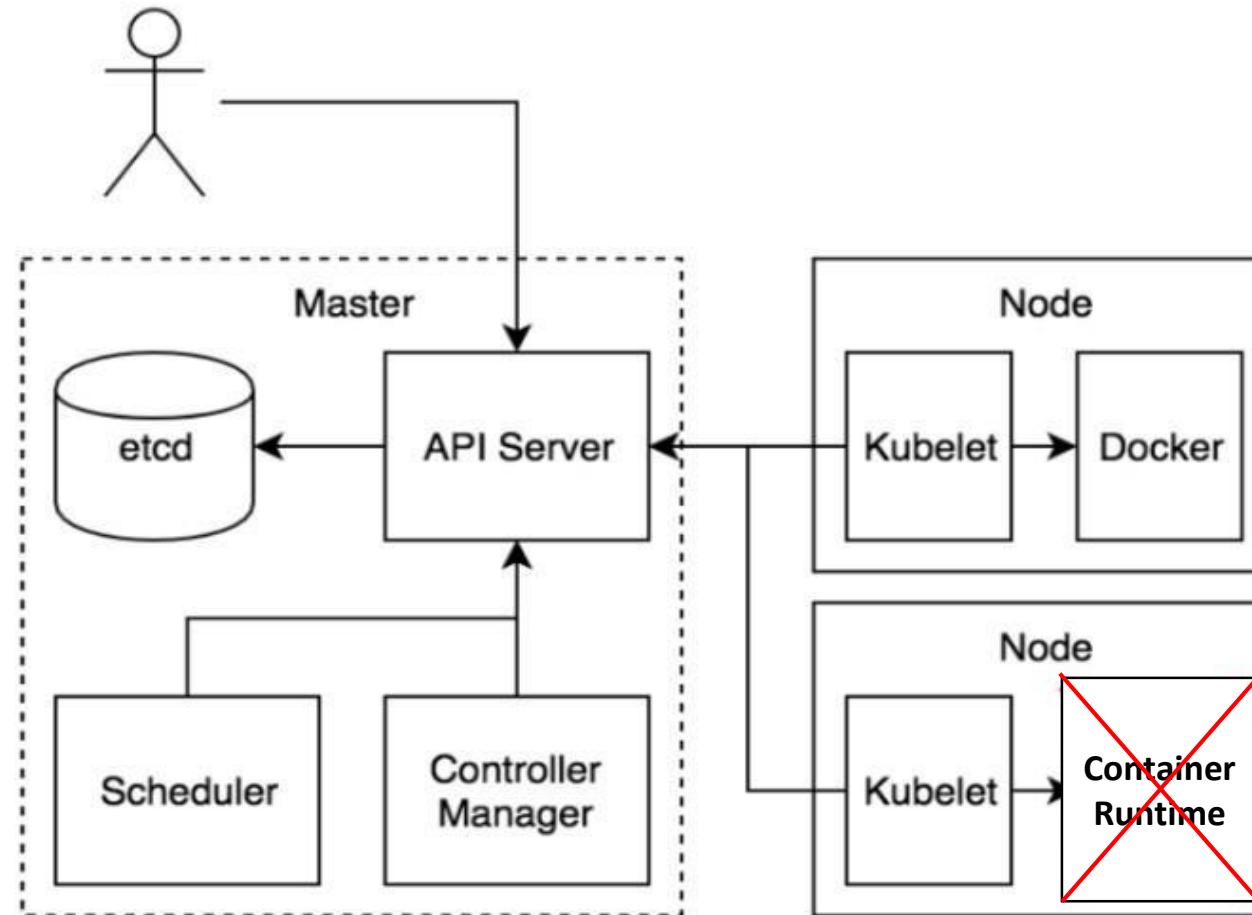
```
[root@k8s-master ~]# kubectl get nodes
NAME        STATUS      ROLES                  AGE   VERSION
k8s-master   Ready       control-plane,master  23h   v1.20.2
k8s-worker1  NotReady   <none>                 23h   v1.20.2
k8s-worker2  Ready       <none>                 23h   v1.20.2
[root@k8s-master ~]# █
```

# Create Application

```
#kubectl run <pod name> --image=<containername>
```

**This will not create on k8s-worker1.**

# Container Runtime Issue



# Note

In a real cluster, this would be caused by worker node going offline or crio crashing.

```
#systemctl stop crio (k8s-worker2)
```

# Symptoms

- Unable to create new workloads. Existing workloads are stopped.

# List Nodes

```
[root@k8s-master ~]# kubectl get nodes
NAME        STATUS      ROLES                  AGE   VERSION
k8s-master   Ready       control-plane,master  23h   v1.20.2
k8s-worker1  NotReady   <none>                 23h   v1.20.2
k8s-worker2  Ready       <none>                 23h   v1.20.2
[root@k8s-master ~]# █
```

# Troubleshooting

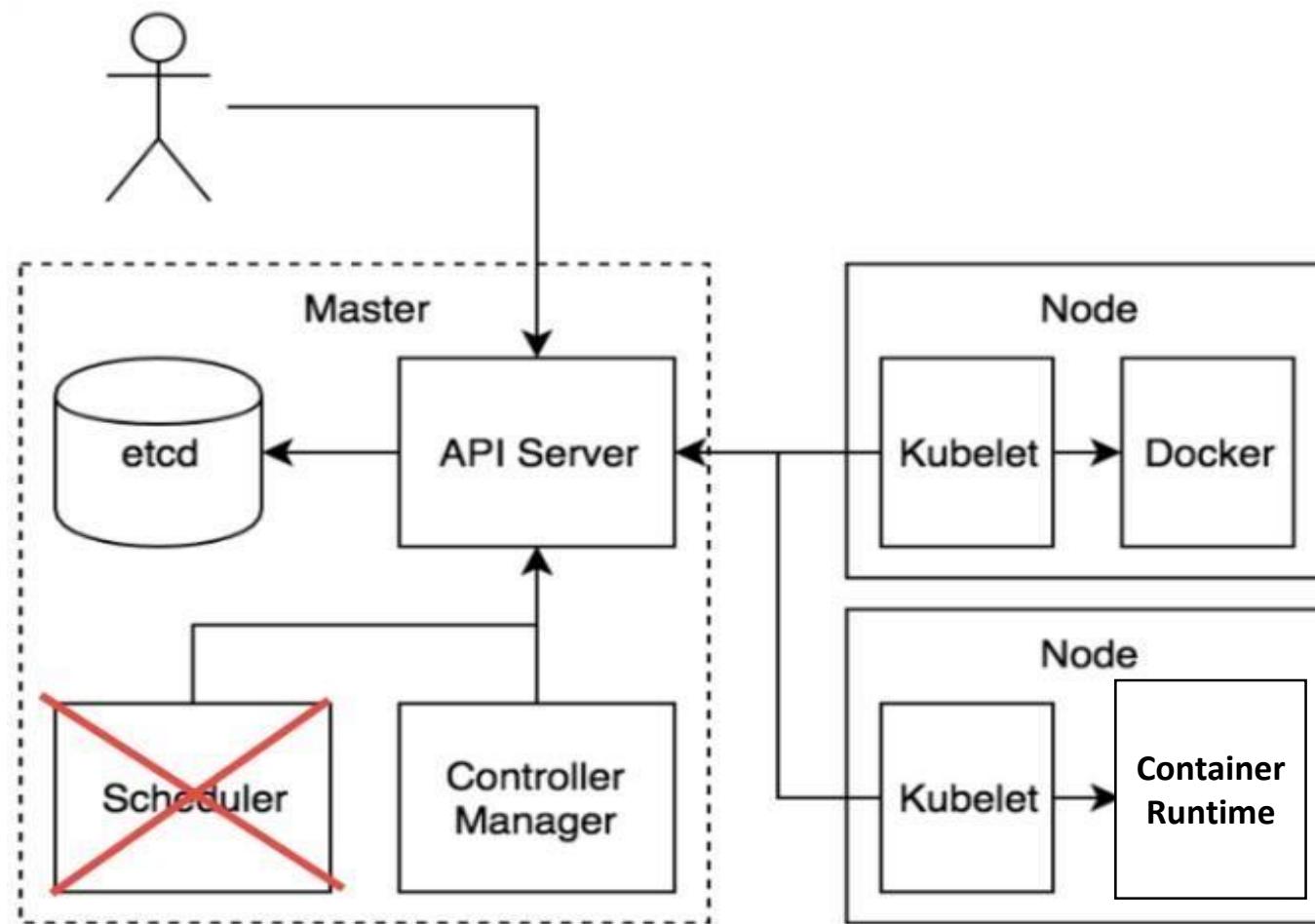
## Master Node:

```
#kubectl get nodes  
#kubectl describe node k8s-worker2
```

## Worker Node:

```
#journalctl -u kubelet -f  
#crictl ps  
#systemctl status crio
```

# Scheduler Issue



# Note

In a real cluster, this would be caused by kube-scheduler shutdown or kube-scheduler crashing.

```
#mv /etc/kubernetes/manifests/kube-scheduler.yaml /home/
```

# Symptoms

- pods get created but will not be scheduled to a node.

```
[root@k8s-master ~]# kubectl run mypod --image=quay.io/gauravkumar9130/mywebapp
pod/mypod created
[root@k8s-master ~]# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
mypod    0/1     Pending   0          3s
[root@k8s-master ~]# █
```

# Restore

```
#mv /home/ /etc/kubernetes/manifests/kube-scheduler.yaml
```

Any  
**Question**



“Thank You”

