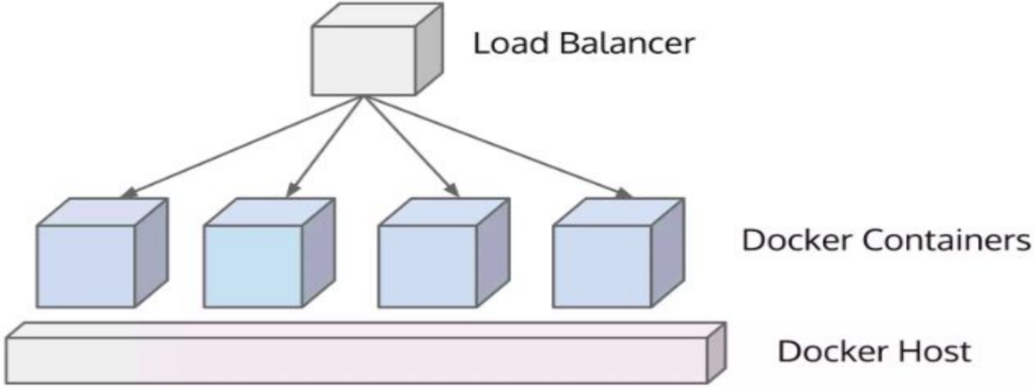
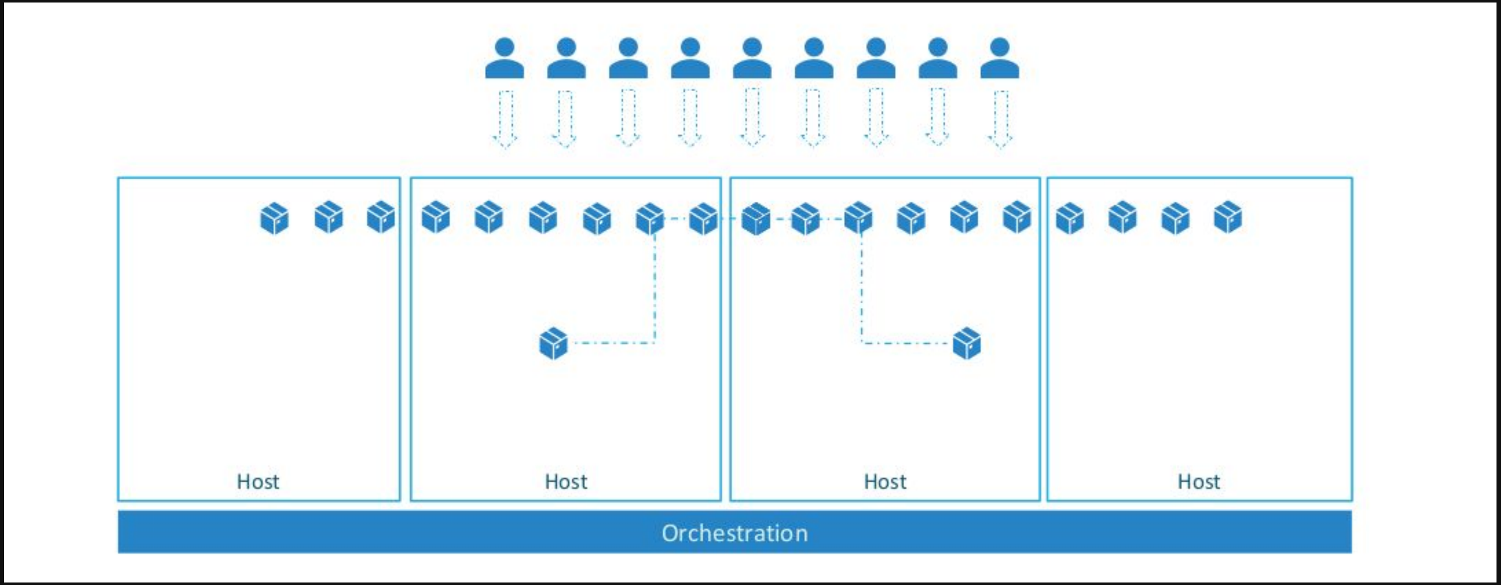
kubernetes

Problems with standalone Docker

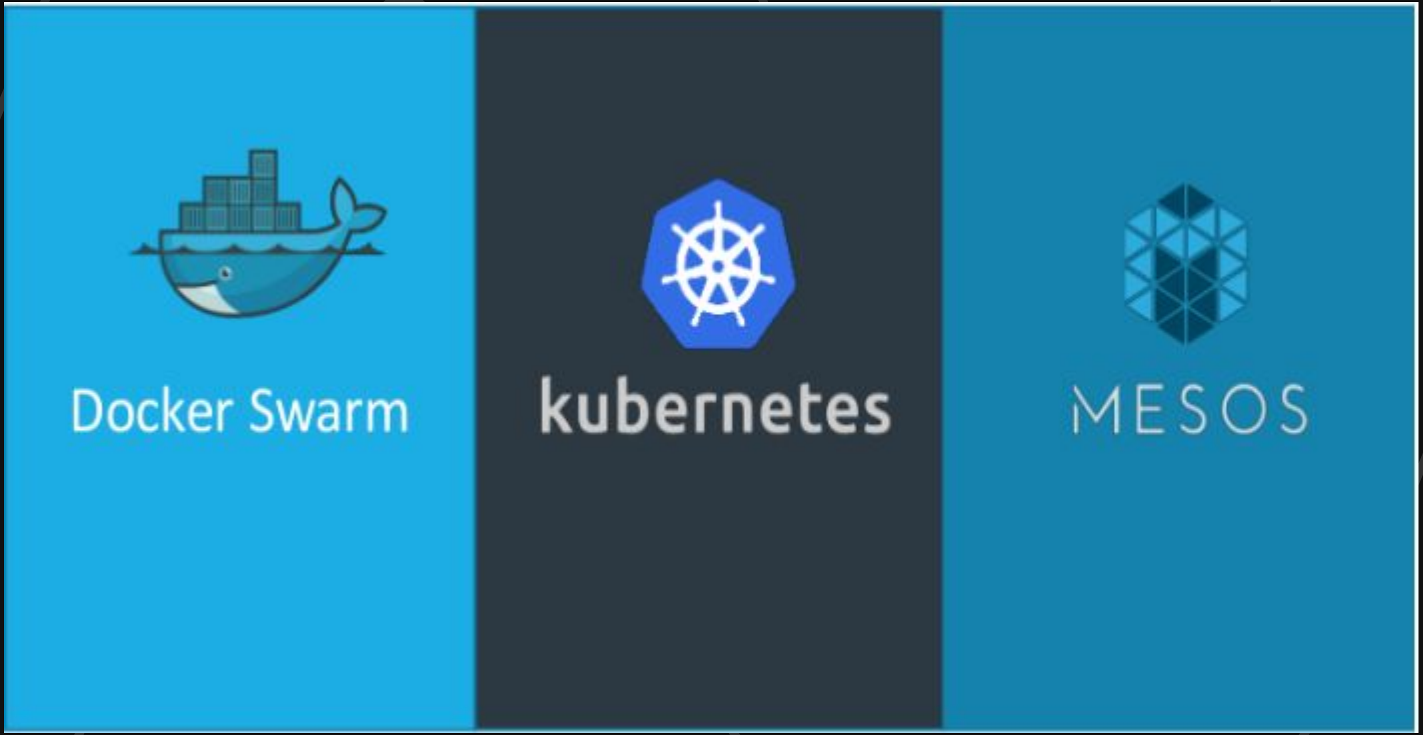


- Running a server cluster on a set of Docker containers, on a single Docker host is vulnerable to single point of failure!

Container Orchestration



Orchestration technology



Advantages of Kubernetes

- Scalability & High Availability
- Automation & Self-Healing
- Efficient Resource Utilization
- Rolling Updates & Rollbacks
- Multi-Cloud & Hybrid Cloud Support
- Microservices & Service Discovery
- Security & Isolation
- CI/CD & DevOps Integration
- Monitoring & Logging

Kubernetes Architecture

POD

- kubernetes does not deploy containers directly on the worker nodes. The containers are encapsulated into a Kubernetes object known as PODs.

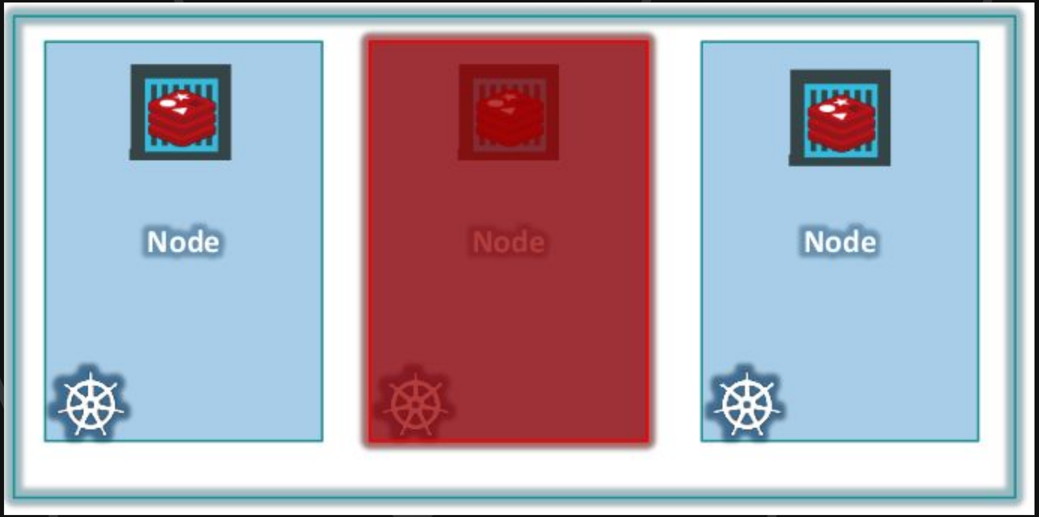
- A POD is a single instance of an application. A POD is the smallest object, that you can create in kubernetes.

Node

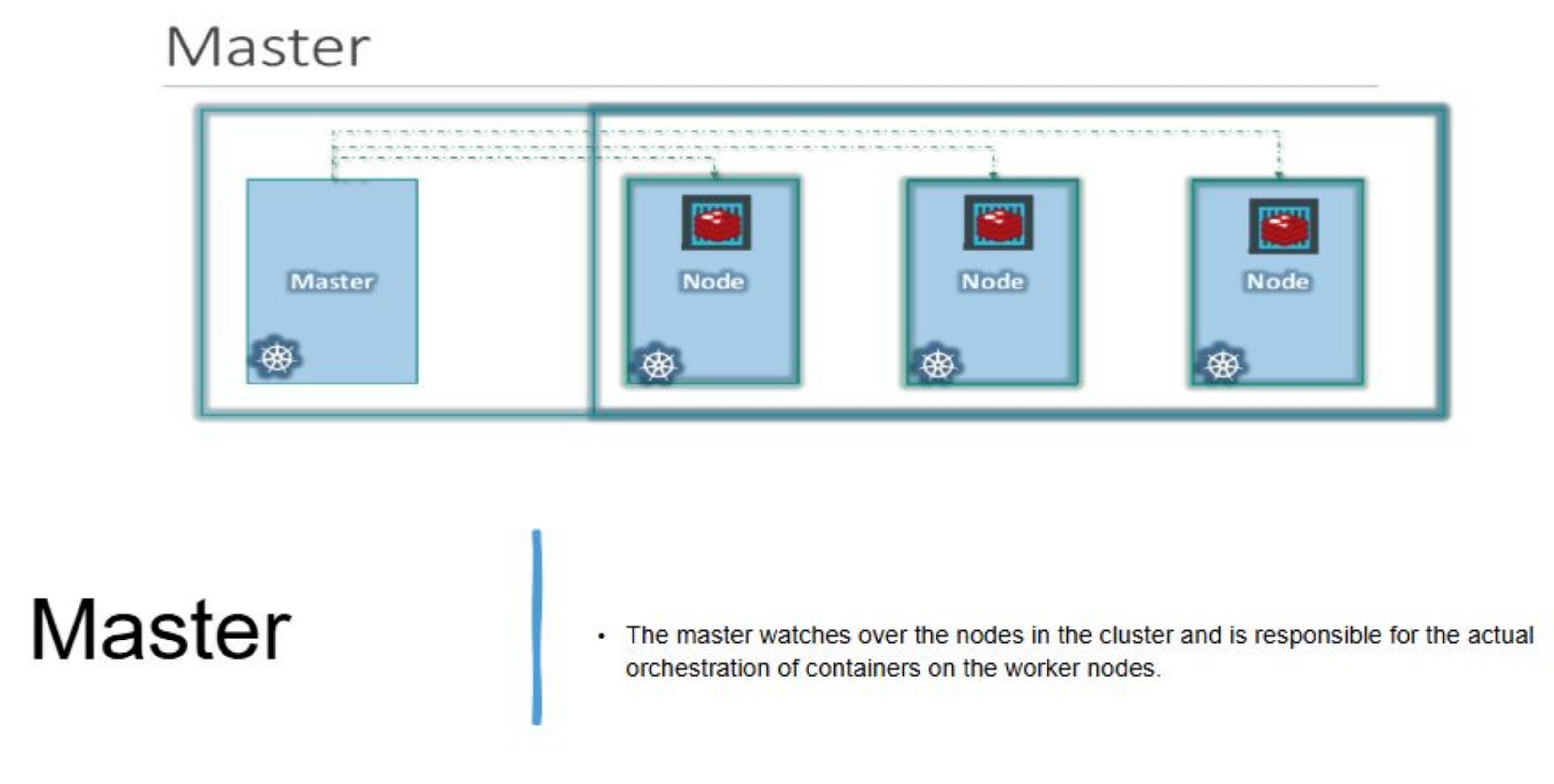
- A cluster is a set of nodes grouped together. This way even if one node fails you have your application still accessible from the other nodes. Moreover having multiple
- It can be VM - or physical Machine

Cluster

- A cluster is a set of nodes grouped together. This way even if one node fails you have your application still accessible from the other nodes. Moreover having multiple
- nodes helps in sharing load as well.



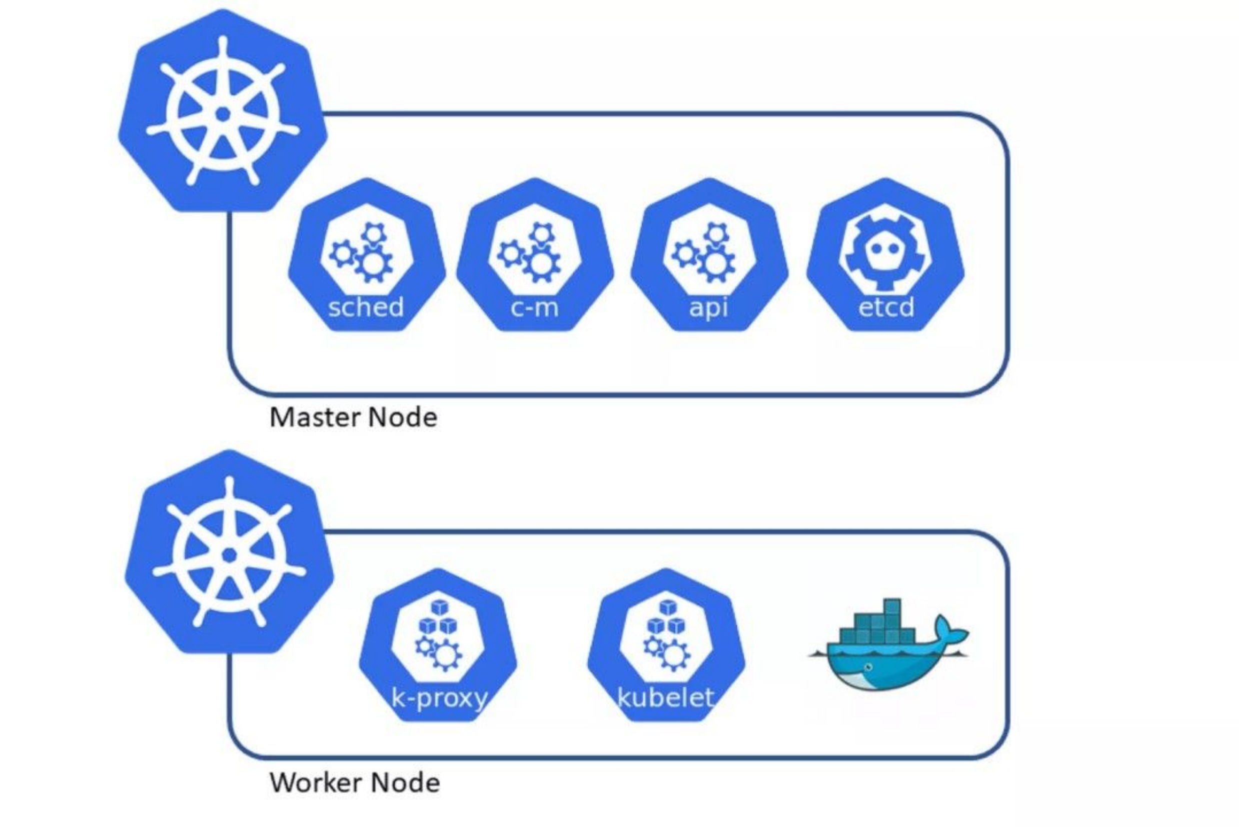
Now we have a cluster, but who is responsible for managing the cluster?



Master

Master

- The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes.


sched
c-m
api
etcd
Master Node
k-proxy
kubelet
Worker Node

Pod

Smallest deployable unit in Kubernetes

A Pod encapsulates one or more containers:

- Usually a single container (but can contain sidecars or helper containers)

All containers in a Pod share:

- The same network namespace (same IP + port space)
- Storage volumes (shared volumes for data exchange)

Pods are ephemeral:

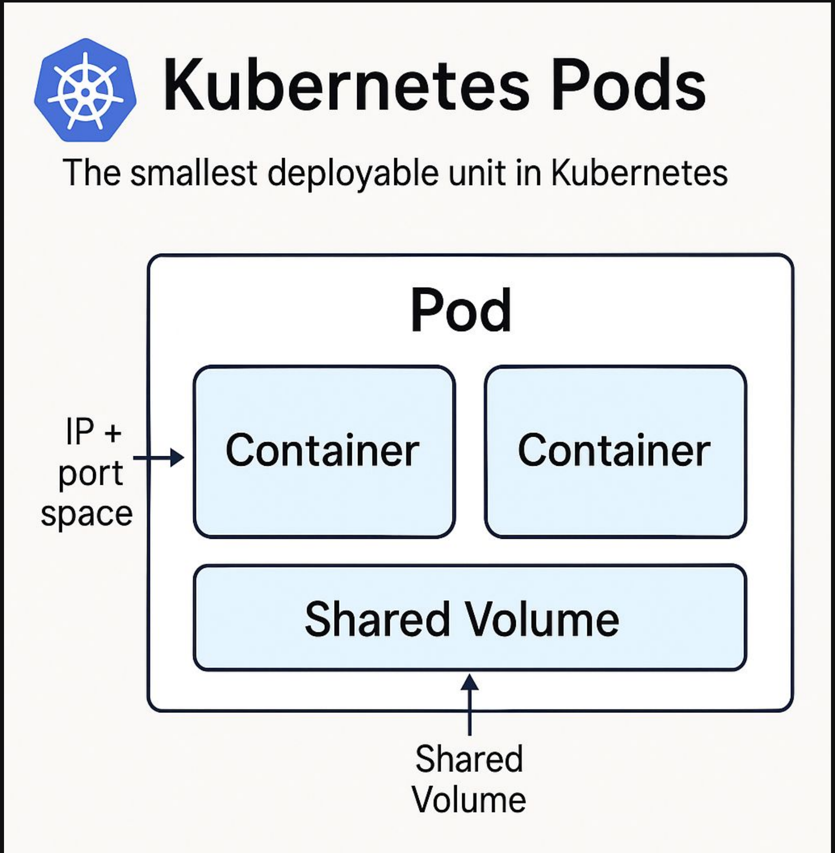
- Pods can be destroyed and recreated by controllers (e.g., Deployments, ReplicaSets)

✅ Pods should not be created directly in production — use Deployments or StatefulSets to manage them.

✅ Pods get a unique IP, but IP changes if the Pod is rescheduled.

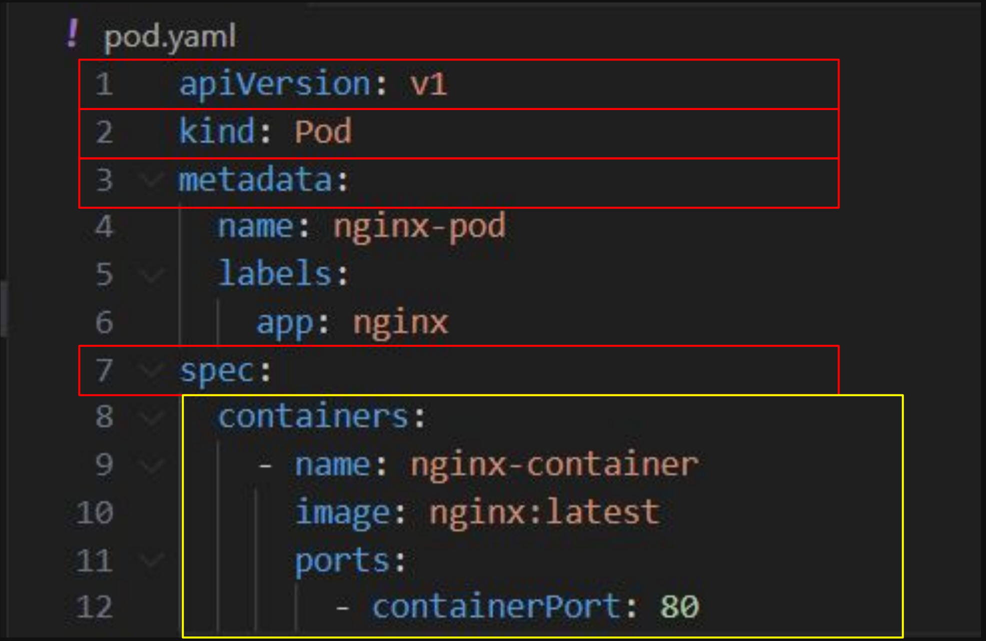
✅ If a Pod dies, it won’t be restarted unless managed by a controller.

✅ Pods are ideal for tightly coupled app components (e.g., app + logging sidecar).



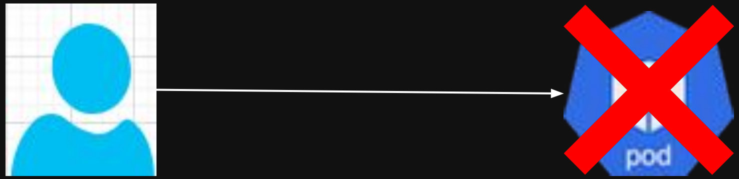
Kubernetes Pods

The smallest deployable unit in Kubernetes



! pod.yaml

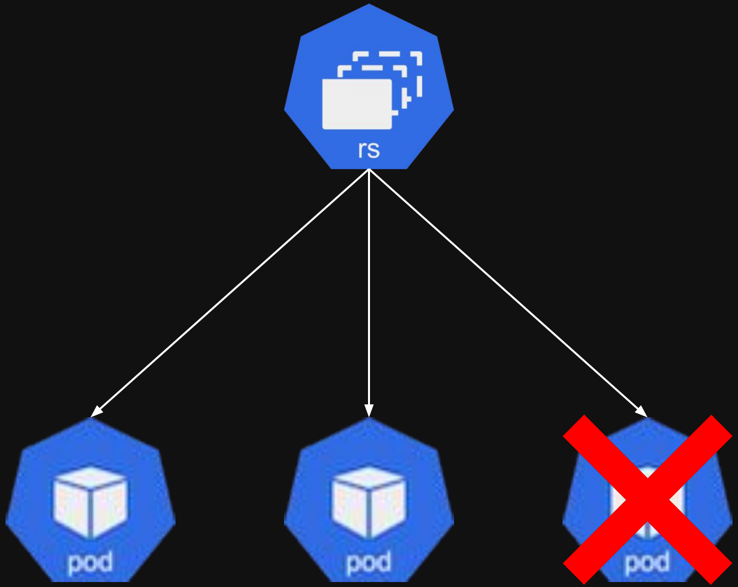
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

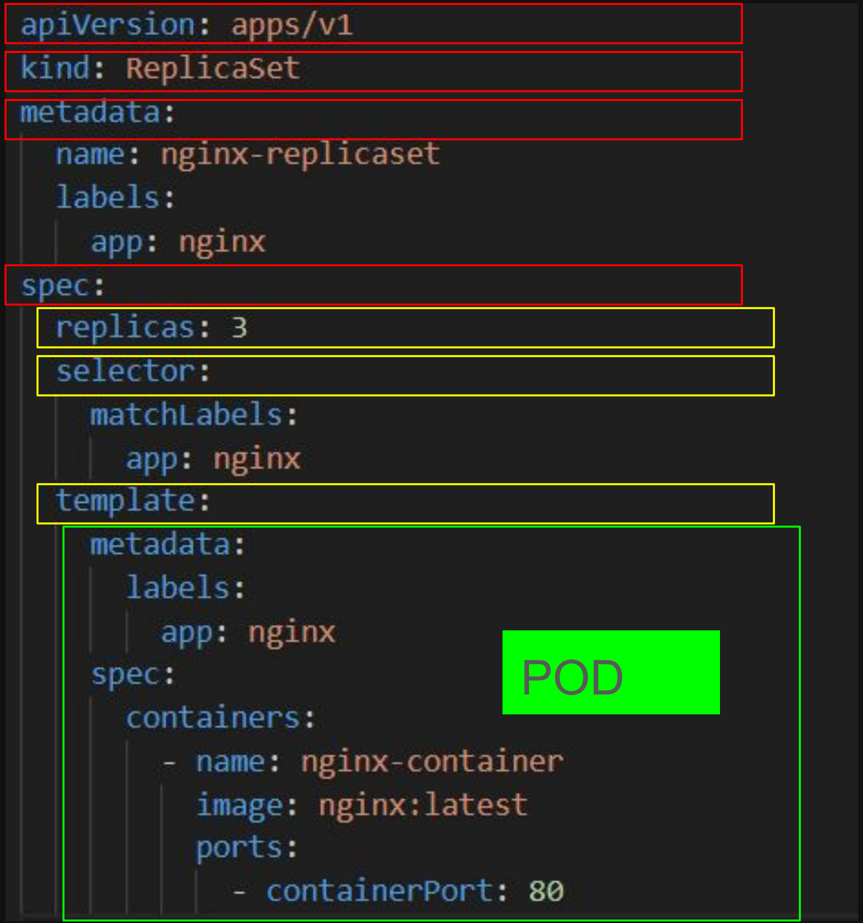

pod

Replicaset

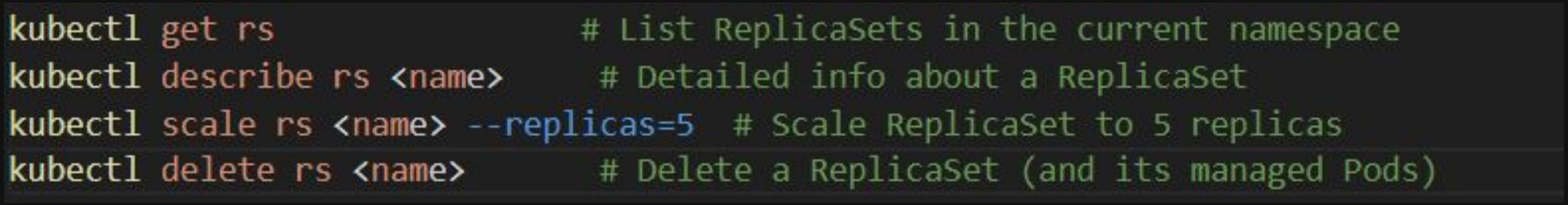
Ensures a specified number of pod replicas are running at any time

- Defines the desired number of identical Pods
- Uses the replicas field to set the target
- Maintains the desired state
- Automatically creates or deletes Pods as needed
- Achieves high availability by Replaces failed Pods - Self-healing
- Pods are selected using labels
- ReplicaSet uses selectors to manage Pods


rs
pod
pod
pod

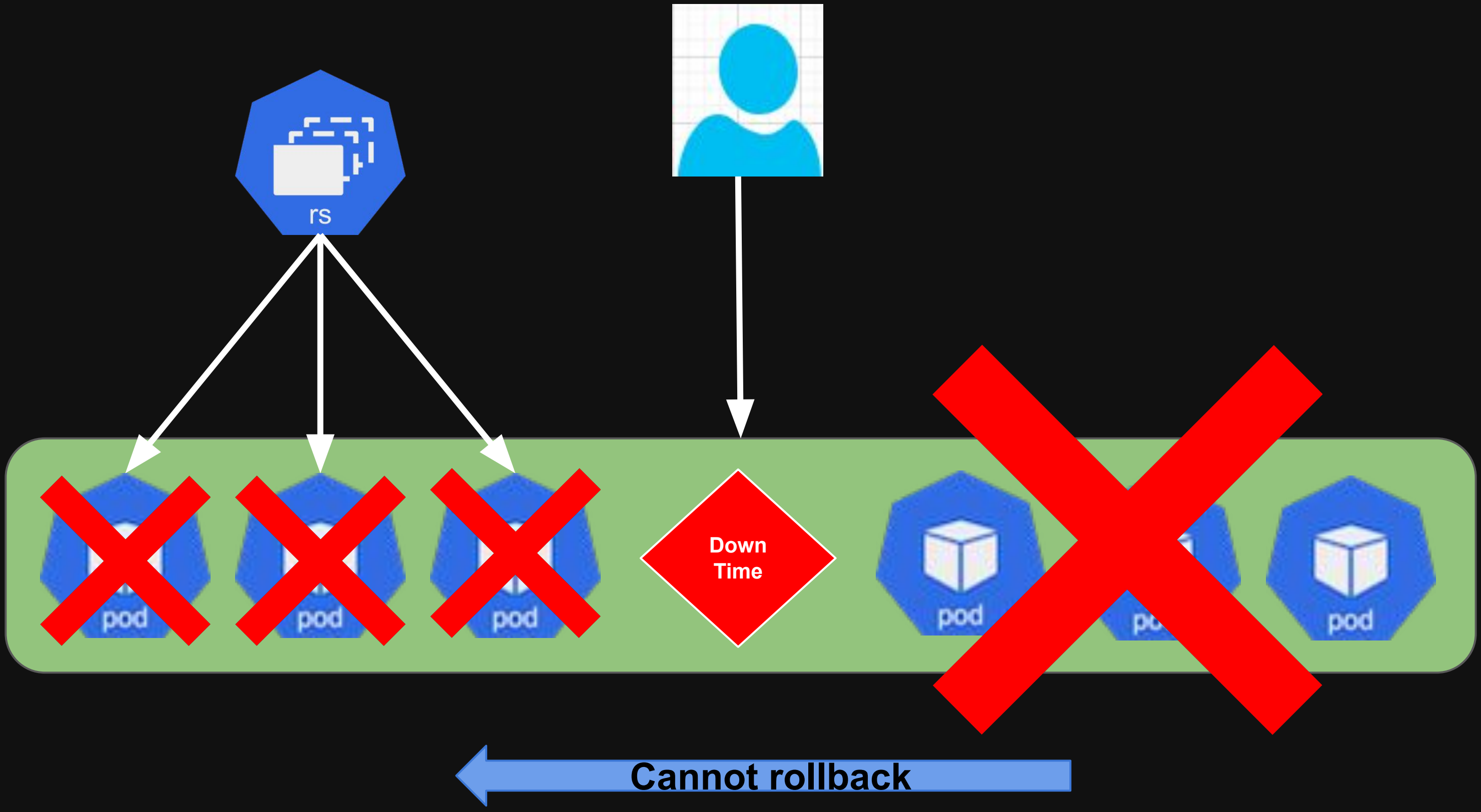


```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx:latest
          ports:
            - containerPort: 80
```



```
kubectl get rs                # List ReplicaSets in the current namespace
kubectl describe rs <name>     # Detailed info about a ReplicaSet
kubectl scale rs <name> --replicas=5  # Scale ReplicaSet to 5 replicas
kubectl delete rs <name>       # Delete a ReplicaSet (and its managed Pods)
```

ReplicaSet Issue

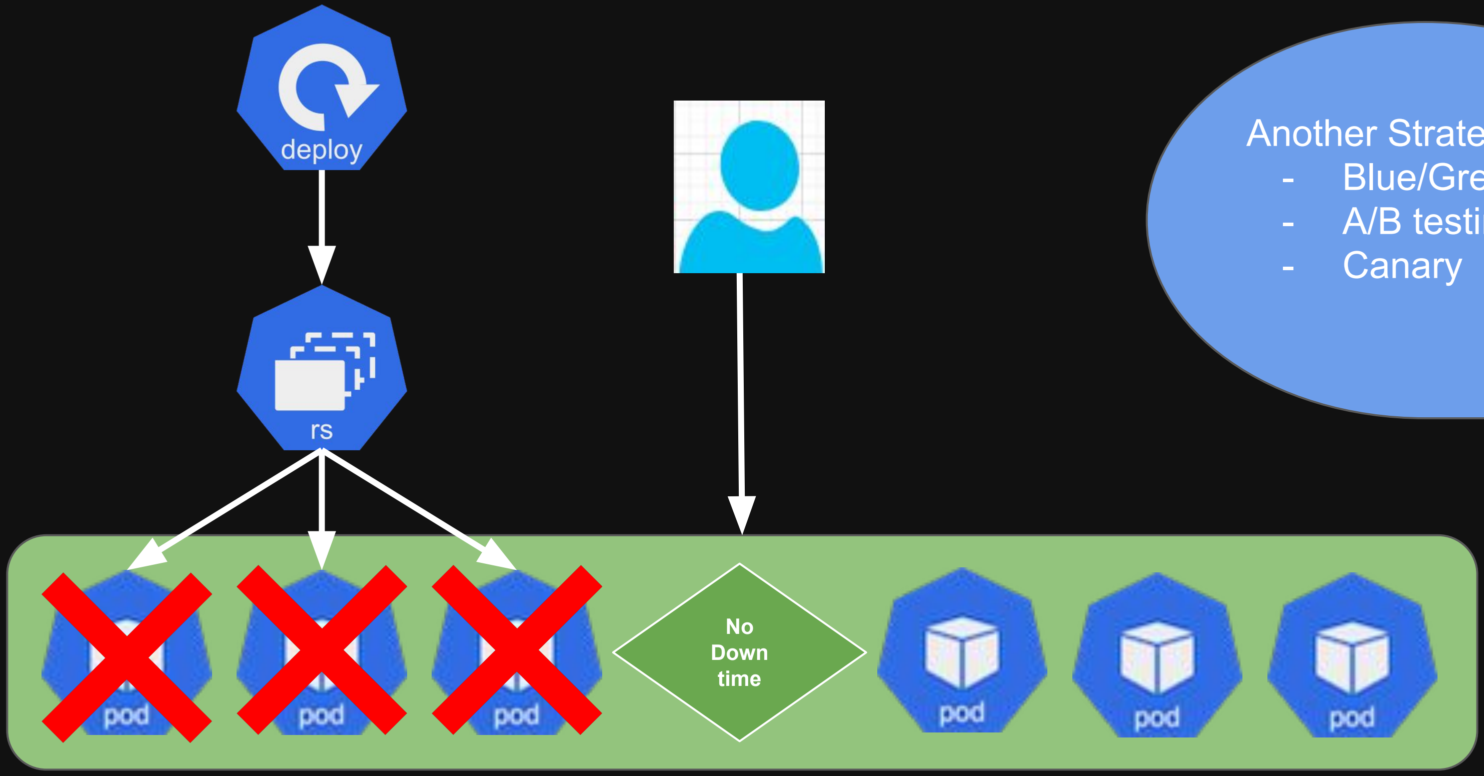


Deployment

Ensures a specified number of pod replicas are running at any time

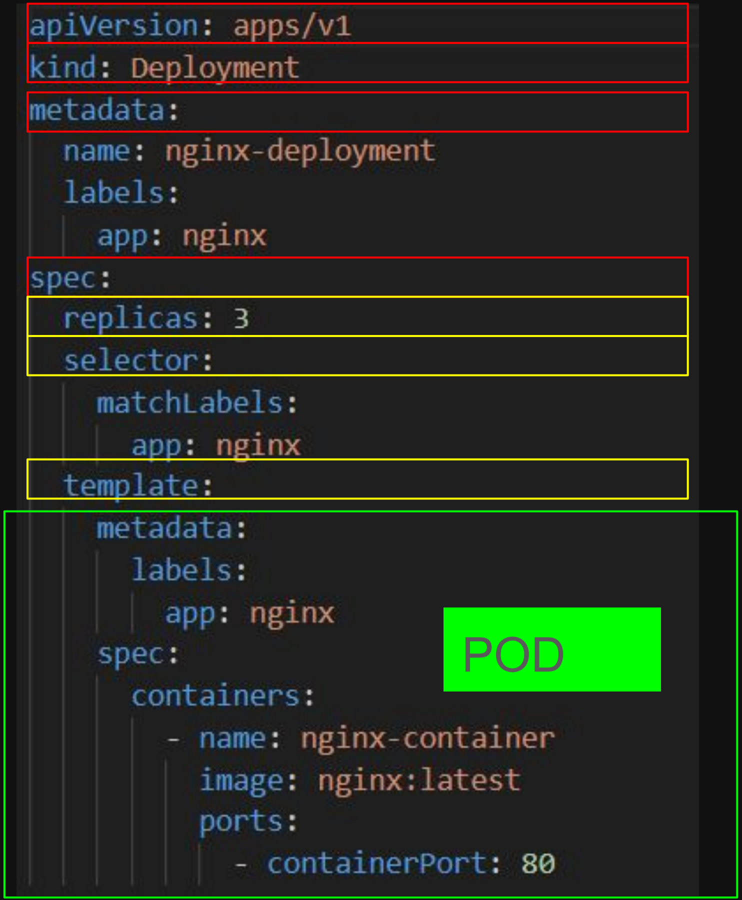
- Defines the desired state for Pods and ReplicaSets
- Automatically manages ReplicaSets to match desired state
- Supports rolling updates and rollbacks
- Ensures high availability
- Provides versioning and history of changes
- Creates, updates, or deletes Pods via ReplicaSets

Rolling-Update Strategy

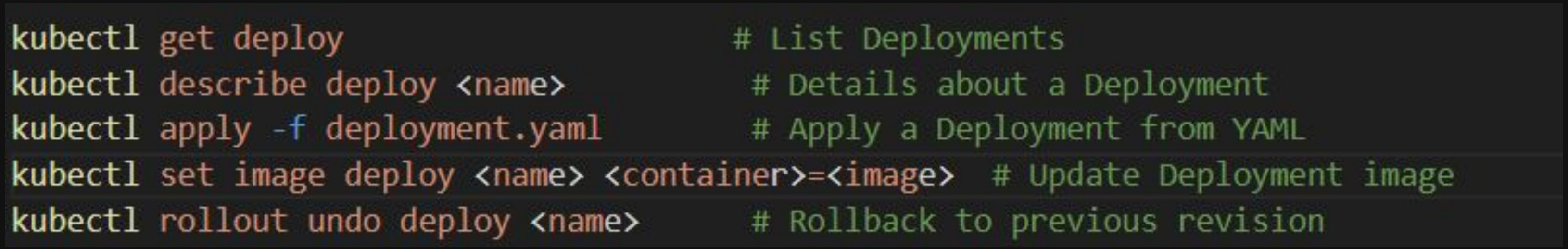


Another Strategies:

- Blue/Green
- A/B testing
- Canary



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx:latest
          ports:
            - containerPort: 80
```



```
kubectl get deploy                     # List Deployments
kubectl describe deploy <name>          # Details about a Deployment
kubectl apply -f deployment.yaml        # Apply a Deployment from YAML
kubectl set image deploy <name> <container>=<image>  # Update Deployment image
kubectl rollout undo deploy <name>      # Rollback to previous revision
```

Demo

Service

Provides stable IP + DNS name

Even if Pods change, the Service endpoint stays the same

Selects Pods via labels

Routes traffic to the right Pods

Enables communication

Exposes Pods internally or externally

Types of Services

ClusterIP → internal only

NodePort → accessible on node IP + port

LoadBalancer → external access via cloud load balancer

pod

master

deploy

rs

node

- Replicaset
- Deploytment
- Deamonset
- cluster IP
- NodePort

Container Orchestration