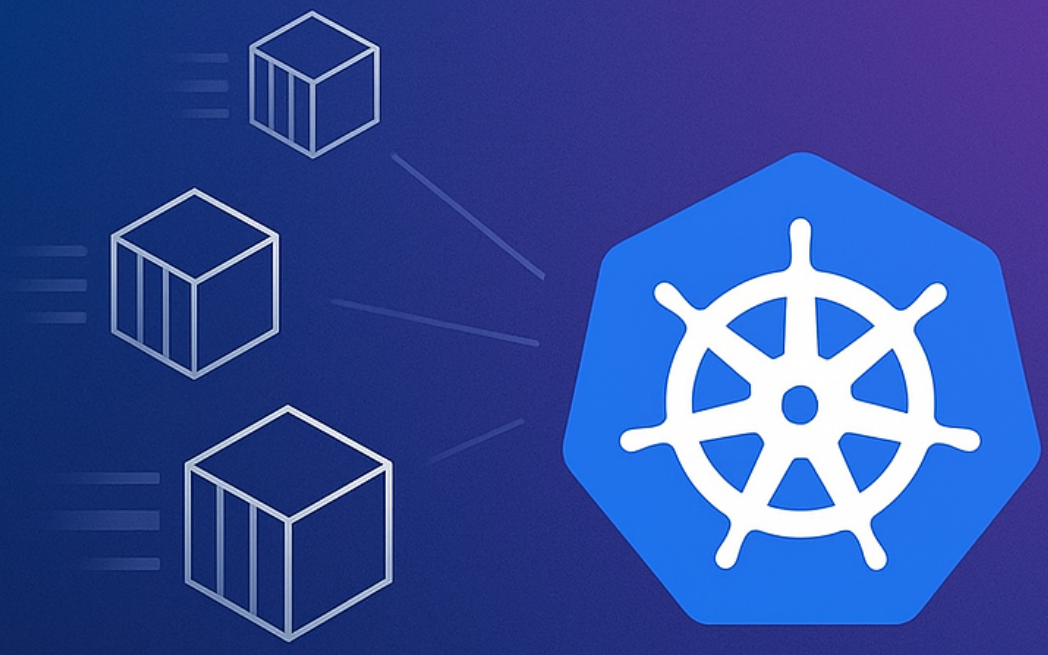


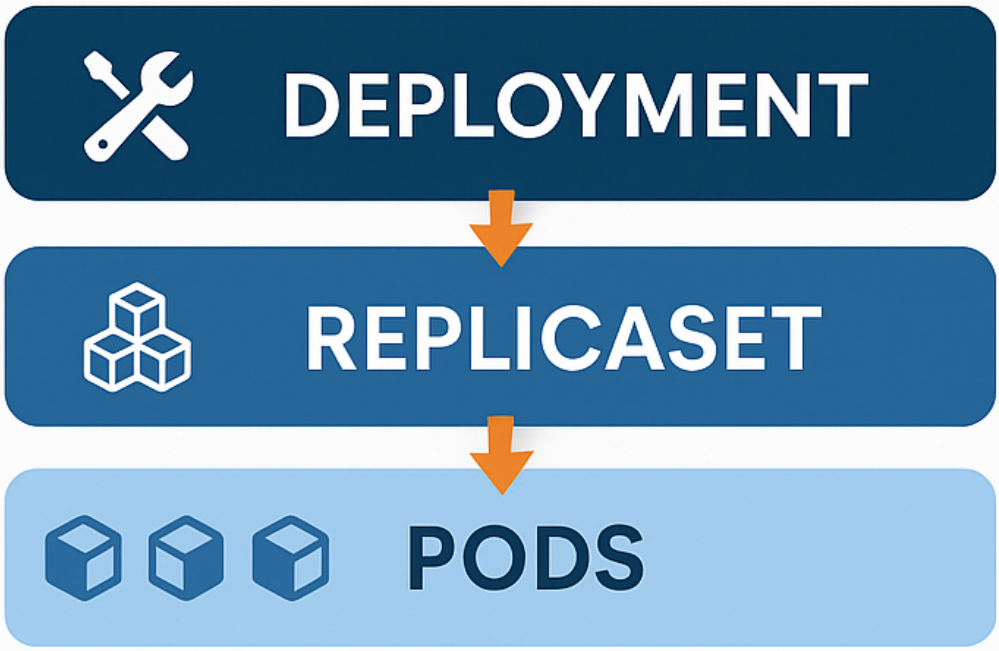
KUBERNETES DEPLOYMENTS

Automate, Scale, Update

WHAT ARE DEPLOYMENTS?

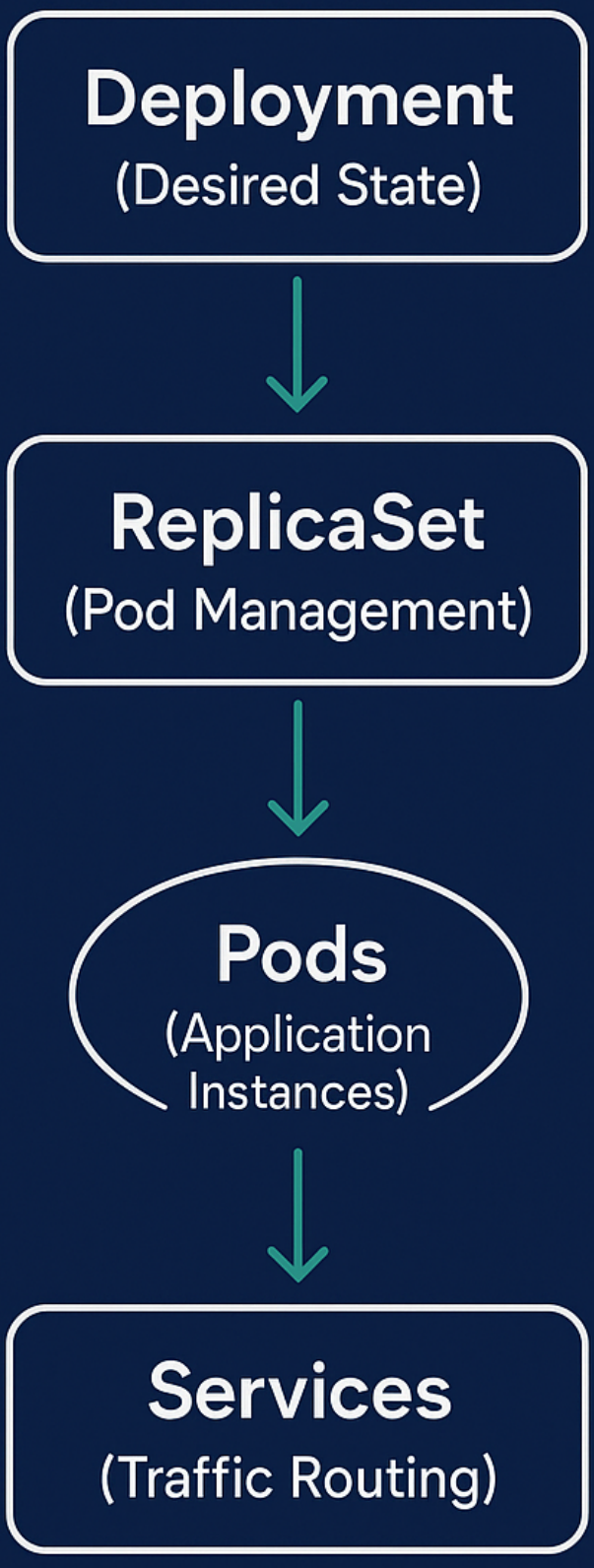
THE ORCHESTRATION POWERHOUSE

- **Declarative Management:** Describe desired state, K8s handles the rest
- **Abstraction Layer:** Sits above ReplicaSets and Pods
- **Automation Hub:** Self-healing, scaling, and updating capabilities
- **Stateless Focus:** Perfect for horizontally scalable applications



CORE ARCHITECTURE

How Deployments Work




```
$ kubectl apply
```

Key Components:

- Controller Pattern: Continuous reconciliation
- Label Selectors Pod identification mechanism
- Pod Templates Blueprint for new instances

```
$ kubectl get pods
$ kubectl get services
```

Essential Spec Fields

Configuration Deep Dive

```
replicas 3

selector:
  matchlàols:
    app: myapp

template
  metadata:
    labels
     app: myapp

strategy
  RollingUpdpdate

strategy
```

replicas
Define your scale (default: 1)

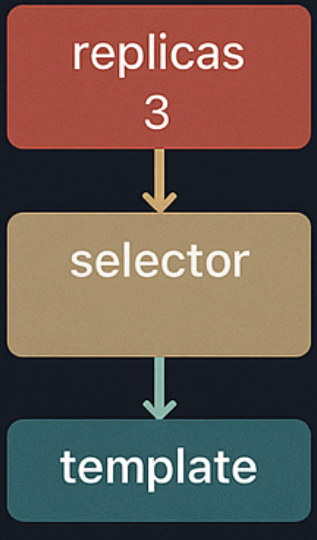
selector
Pod targeting mechanism

template
Pod blueprint definition

strategy
Update methodology (RollingUpdate/ Recreate)

Critical Configuration Options:

```
replicas
se lector
  name: app
  image: myapp:v1.0
```



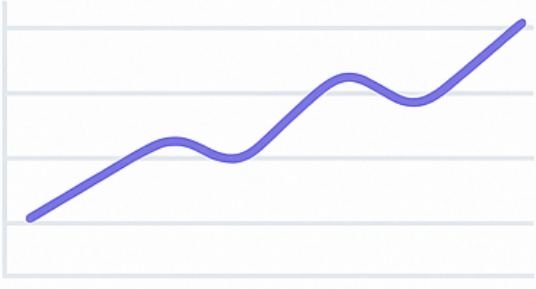
SCALING STRATEGIES

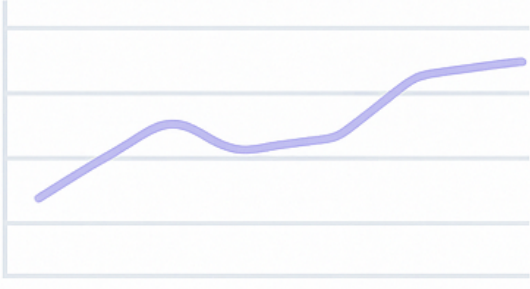
Horizontal Scaling Made Simple

```
Scaling Operations:
kubectl scale deployment myapp ---repliicas=5
# Auto-scalimg (HPA)
kubectl autoscale deployment myapp \
--cpu-percent=70 --min=2 --max=10
```

Best Practices:

- Remove replicas from manifests to prevent overrides
- Use Horizontal Pod Autoscaler (HPA) for dynamic scaling
- Monitor resource utilization patterns
- Consider cluster capacity constraints


CPU

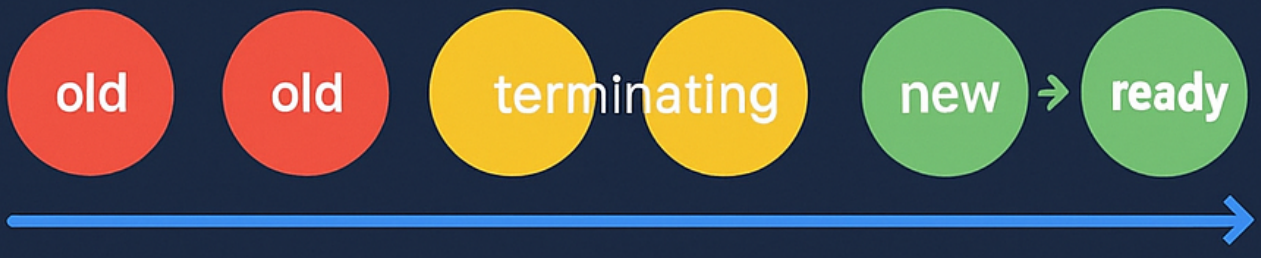

Memory

UPDATE STRATEGIES

Zero-Downtime Deployments

Rolling Update (Default)

- Gradual replacement of old pods
- Configurable surge and unavailability
- Maintains service availability



Update sequence

Recreate Strategy

- Simultaneous termination and creation
- Brief downtime but faster updates

Advanced Parameters:

```yaml
strategy:
  type: RollingUpdate
  rollingUpdate:
   maxSurge:25%
   maxUnavailable:25
   minReadySeconds:30
```

Health Monitoring

Probes and Self-Healing

/health

```
livenesssProbe:
  httpGet
   path: /health
   port: 8080
  initialDelaySeconds:30
  periodSeconds:10
```

/ready

```
readinessProbe:
  httpGet
   path: /ready
   port: 8080
  initialDelaySeconds:5
  periodSeconds:5
```

- Probe Types: HTTP, TCP, Command execution
- Self-Healing: Automatic restart of failed containers

Rollback Capabilities

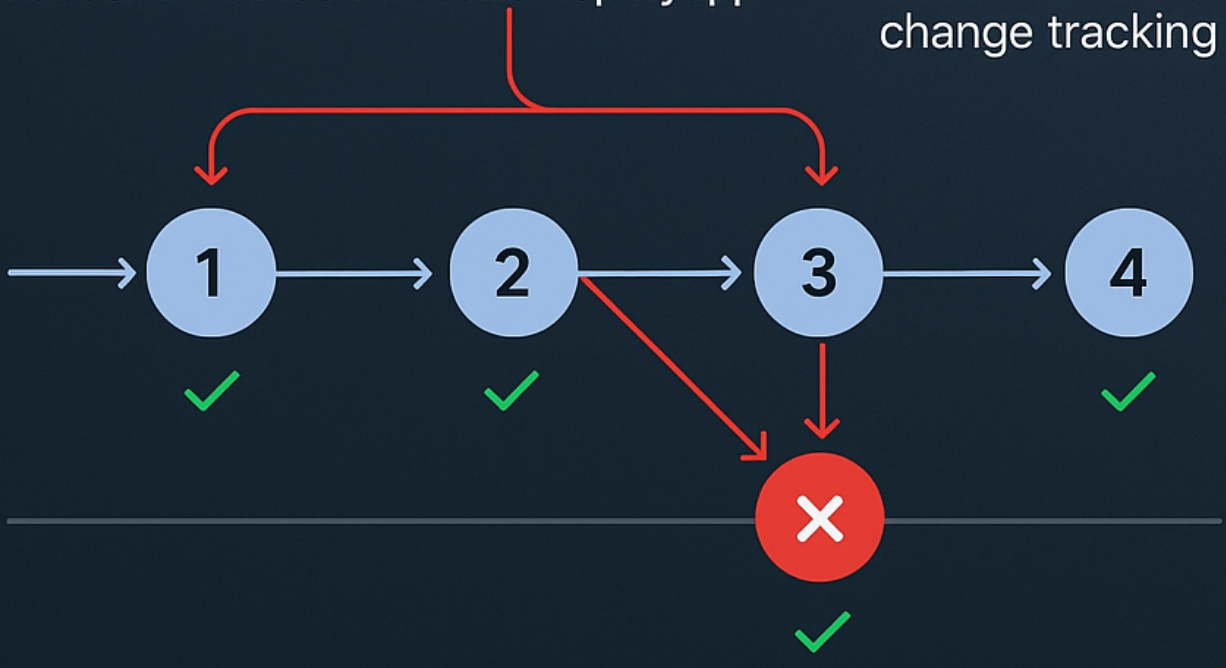
Version Control for Applications

Rollback Commands:

```
kubectl rollout history dep myapp
# Rollback to prevlous version
kubectl rollout undo deplovment
myapp --to-revision=2
kubectl to specific revision
nubectirollout pause deploym yapp
kubectl rollout resume dep myapp
```

Revision Management:

- Automatic revision tracking
- Configurable history limits
- Deployment annotations for change tracking



Essential kubectl Commands

Command Mastery Cheat Sheet

Creation & Management:

```
kubectl create deployment myapp -image=nginx:1.20
kubectl apply -f deployment.yaml
kubectl get deployments
 NAME                  READY  UP-TO-DATE  AVAILABLE
 myapp                 1   1  1           1  2 min
kubectl describe deployment myapp
```

Scaling & Updates:

```
kubectl scale deployment myapp --replicas=3
kubectl set image deployment/myapp container=nginx:1.21
kubectl rollout status deployment myapp
```

Troubleshooting

```
kubectl logs deployment/myapp
kubectl exec -it deployment/myapp -- /bin/bash
```

_

OpenShift Differences

Kubernetes

Bue-Int

```
yamltriggeres:
  type: ConfigChange
  type: ImageChange
  -from
    kind: ImagpStreamTag
    name: myappilatest
```

Migration Path:
DeploymentConfigs → Standard Deployments

Red Hat OpenShift Enhancements

OpenShift-Specific Features:

- **DeploymenConfgs** Legacy):
  - More advanced trigger mechanisms
  - Buiit-in build integgration
- **Enhanced Rolling Updates**
  - Additional hooks (pre/mid/post)
  - Custom deploy ment strategies

Security Enhancements

- Security Context Constraints (SCCs)
- Automatic security policy enforcement
- Non-root containers by default

Sécurity Enhancements

- Security Context Constraints (SCCIs)

BEST PRACTICES 🏆

Production-Ready Guidelines

Resource Management

```
resources
    requests:   cpu: 100m
    memory      128Mi
    limits      cpu: 500m
    memory      512Mi
```

Security Hardening

- Run as non-root user
- Use read-only root filesystem
- Implement network policies
- Regular image vulnerability scanning

Operational Excellence

- Implement comprehensive monitoring
- Use structured logging
- Define clear SLAs/SLOs
- Automate deployment pipelines

🚀 Level Up Your Kubernetes Skills

Next Steps:

- ✓ Practice with hands-on labs
- ✓ Implement CI/CD with Deployments
- ✓ Explore advanced patterns (Blue/Green, Canary)
- ✓ Master Helm charts for complex applications

Connect & Learn:

- Follow for more DevOps insights
- Share your Deployment experiences
- Comment with your biggest K8s challenges
- Save this carousel for quick reference

Resources

- Official Kubernetes Documentation
- CNCF Training Programs
- Platform Engineering Communities

1.234 56

Follow