



Module 3c: Managing Deployments, ReplicaSets and Services

Welcome to this hands-on Kubernetes lab focused on managing Deployments, ReplicaSets, and Services. This comprehensive guide will help you master essential Kubernetes resource management concepts through practical exercises. You'll learn how these core components work together to create robust, scalable applications in your cluster. All exercises should be performed using the student user on the master node.

Understanding Deployments

What is a Deployment?

A Deployment is a high-level Kubernetes resource that manages the lifecycle of your application pods. It provides declarative updates and maintains the desired state of your application, automatically creating and managing ReplicaSets behind the scenes.

Deployments are the recommended way to manage stateless applications in Kubernetes because they handle rolling updates, rollbacks, and scaling automatically.

Key Benefits

- Automated rollout and rollback capabilities
- Self-healing through pod recreation
- Declarative configuration management
- Version history and revision tracking

Creating and Managing Deployments

01

Create the Deployment

Generate a deployment definition file and create the resource:

```
kubectl create deploy nginx --image=nginx -o yaml --dry-run=client > deploy-web.yaml
kubectl create -f deploy-web.yaml
```

02

Display Resources

View all resources created by the deployment:

```
kubectl get deploy,rs,po
```

03

Test Self-Healing

Try deleting pods and ReplicaSets to observe Kubernetes' self-healing behavior:

```
kubectl delete po we
kubectl get po
kubectl delete rs we
kubectl get rs
```

04

Clean Up

Remove the deployment and verify all resources are deleted:

```
kubectl delete deploy web
kubectl get deploy,rs,po
```

Deployments manages ReplicaSets and ReplicaSets manages Pods. Therefore, always deal with the parent resource and not child resource whenever changes need to be done else risk losing your changes.

 **Pro Tip:** To create a resource definition file for an existing deployment, remove the creationTimestamp, resourceVersion and uid lines. Also remove all lines including and after status:

```
kubectl get deploy abc -o yaml > somefile.yaml
```

Working with ReplicaSets

ReplicaSets ensure that a specified number of pod replicas are running at any given time. While Deployments are the preferred method for managing pods, understanding ReplicaSets is crucial since they form the foundation beneath Deployments.



Replication

Maintains desired number of identical pods



Label Selection

Uses labels to identify and manage pods



Self-Healing

Automatically replaces failed pods

ReplicaSet Lab Exercises

1 Create the ReplicaSet

Apply the configuration and wait for pods to start:

```
kubectl apply -f 3s-rs.yaml  
kubectl get rs,po
```

2 Modify Pod Labels

Edit a pod's label to change superlabel value to "dummy":

```
kubectl edit po super-rs-...
```

Then check the pod count:

```
kubectl get rs,po
```

What happened? The ReplicaSet created a new pod because the edited pod no longer matched its selector!

3 Clean Up and Scale

Recreate the ReplicaSet and remove the stray pod:

```
kubectl apply -f 3s-rs.yaml  
kubectl delete po -l superlabel=dummy  
kubectl get rs,po
```

Scale down to 1 replica:

```
kubectl scale rs/super-rs --replicas=1  
kubectl get rs,po
```

4 Orphan Resources

Delete the ReplicaSet while keeping pods:

```
kubectl delete rs super-rs --cascade=orphan  
kubectl get rs,po
```

Final cleanup:

```
kubectl delete po -l superlabel=best
```

Kubernetes Services Overview



What are Services?

Services provide stable networking endpoints for accessing pods. Since pods are ephemeral and their IP addresses change, Services act as an abstraction layer that provides a consistent way to reach your application.

Three Methods to Create Services

1. Expose a deploy/dc, rs/rc, or a pod
2. Using "kubectl create service"
3. Using a service resource definition file

1

ClusterIP

Internal cluster access only
(default)

2

NodePort

Exposes service on each node's
IP at a static port

3

LoadBalancer

Creates external load balancer
(cloud provider required)

Creating and Exposing Services

Recreate Deployment

```
kubectl create -f deploy-web.yaml
```

Attempt to Expose

This will fail because the deployment doesn't define container ports:

```
kubectl expose -h  
kubectl expose deploy/web
```

Add Container Port

Edit deploy-web.yaml to add port specification:

```
vim deploy-web.yaml
```

Add these lines under containers:

```
ports:  
- containerPort: 80
```

Apply Changes

```
kubectl apply -f deploy-web.yaml  
kubectl get deploy,rs,po
```

📖 **Understanding kubectl explain:** Use this command to explore resource structure:

```
kubectl explain deploy.spec.containers  
kubectl explain deploy.spec.containers.ports
```

Now expose the deployment successfully:

```
kubectl expose deploy web  
kubectl get svc  
kubectl get ep nginx
```

Advanced Service Operations

Scaling and Updating Deployments

Modify the deployment to scale replicas, change images, and update ports. Pay attention to whether new ReplicaSets are created with each change:

```
kubectl get deploy,rs,po
kubectl scale deploy web --replicas 2
kubectl get deploy,rs,po,svc,ep -o wide
kubectl set image deploy/web nginx=quay.io/kelvinlai/myphp:port8080
kubectl edit deploy/web # change the port to 8080
```

Testing Service Access

Access the application using both Pod IP and Service IP:

```
curl <pod-ip>:8080
curl <service-ip>:80
```

Important: These commands work from inside the cluster. This demonstrates why services are preferred over direct pod IP references.

Resource Management

Scale back to conserve resources:

```
kubectl scale deploy/web --replicas 1
```

Applications should reference other services using DNS names or service IPs, not pod IPs, since pod IPs can change.

Service Types: NodePort and LoadBalancer

Exposing Services Externally

Create NodePort and LoadBalancer services to make your application accessible from outside the cluster:

```
kubectl expose deploy/web --name web-np --type NodePort
kubectl expose deploy/web --name web-lb --type LoadBalancer
kubectl get svc -o wide
kubectl get ep -o wide
```

 **Note:** The LoadBalancer will remain in "Pending" state unless you have a solution like MetalLB, kube-vip or Cloud provider integration.

Browser Access

Browse to your load balancer IP using the assigned port number. Check your public IP and configure firewall/routing as needed:

```
curl ifconfig.io
```

Example: master:54321

Port Investigation

Check which services are listening on the exposed port:

```
sudo ss -nltp
```

The instructor will explain the networking behavior you observe.

Cleanup

Remove the external services:

```
kubectl delete svc web-lb
web-np
```

Managing Deployment Strategy

Rollout History

View and manage deployment revisions:

```
kubectl rollout history deploy/web  
kubectl rollout history deploy/web --  
revision=2  
kubectl rollout history deploy/web --  
revision=3
```

Making Changes

```
kubectl set image deploy/web nginx=nginx --  
record
```

☐ **Warning:** The --record option is deprecated. Use `kubectl annotate` instead to set `kubernetes.io/change-cause` annotation manually.

Rollback Operations

Verify changes and revert if needed:

```
kubectl describe po web |grep Image:  
kubectl rollout undo deploy/web
```



Deployment Strategy: Recreate

Change from RollingUpdate to Recreate strategy, scale to 3 replicas, and observe how pods are created:

```
strategy:  
type: Recreate
```

With Recreate strategy, all pods are terminated before new ones are created, causing temporary downtime but ensuring no version mixing.

1

RollingUpdate

Gradual replacement, zero downtime

2

Recreate

Complete termination then creation