Kubernetes Hands-On Labs — Replication Controller, ReplicaSet, and Deployment

Author : Sandeep Kumar Sharma

Lab 1: Replication Controller

Introduction:

A **ReplicationController (RC)** ensures that a specified number of pod replicas are running at any given time. If a Pod goes down, RC automatically creates another one. However, it's now mostly replaced by ReplicaSet and Deployment in modern Kubernetes setups.

Objective:

Learn how to create, verify, scale, and delete a ReplicationController.

Step 1 — Create ReplicationController YAML

Create a file named `replication-controller.yaml`:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step 2 — Apply the YAML File

```
kubectl apply -f replication-controller.yaml
```

Step 3 — Verify Pods Created by RC

```
kubectl get pods -l app=nginx
kubectl get rc
```

You should see three Pods created and managed by the ReplicationController.

Step 4 — Test Self-Healing

Delete one Pod and see how the RC recreates it automatically:

```
kubectl delete pod <pod-name>
```

Now verify again:

```
kubectl get pods
```

A new Pod should appear.

Step 5 — Scale the RC

To increase replicas from 3 to 5:

```
kubectl scale rc nginx-rc --replicas=5
kubectl get pods
```

Step 6 — Delete RC and Its Pods

```
kubectl delete rc nginx-rc
```

Verify:

```
kubectl get pods
```

✅ **Outcome:** You have learned how to create, scale, and test the self-healing property of a ReplicationController.

Lab 2: ReplicaSet

Introduction:

A **ReplicaSet (RS)** is the modern replacement for ReplicationController. It performs the same function — maintaining a stable set of replicas — but provides more powerful label-based selection and integration with Deployments.

Objective:

Learn how to create, verify, scale, and delete a ReplicaSet.

Step 1 — Create ReplicaSet YAML

Create a file named `replicaset.yaml`:

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

Step 2 — Apply the ReplicaSet

```
kubectl apply -f replicaset.yaml
```

Step 3 — Verify ReplicaSet and Pods

```
kubectl get rs
kubectl get pods -l tier=frontend
```

Step 4 — Delete a Pod and Test Self-Healing

```
kubectl delete pod <pod-name>
```

The ReplicaSet will automatically create a new Pod to maintain 3 replicas.

Step 5 — Scale the ReplicaSet

```
kubectl scale rs nginx-rs --replicas=5
kubectl get pods
```

Now there should be 5 Pods.

Step 6 — Delete the ReplicaSet and Its Pods

```
kubectl delete rs nginx-rs
```

Verify:

```
kubectl get pods
```

✅**Outcome:** You have learned how ReplicaSet ensures desired Pod count and maintains consistency using label-based selection.

Lab 3: Deployment

Introduction:

A **Deployment** is a higher-level abstraction that manages ReplicaSets and Pods. It adds rolling updates, rollbacks, and versioning capabilities, making it the preferred controller in production.

Objective:

Learn how to create, update, scale, and roll back a Deployment.

Step 1 — Create Deployment YAML

Create a file named `deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

Step 2 — Apply the Deployment

```
kubectl apply -f deployment.yaml
```

Step 3 — Verify Deployment and Pods

```
kubectl get deployments
kubectl get pods -l app=nginx
kubectl describe deployment nginx-deployment
```

Step 4 — Test Rolling Update

Edit the deployment to update the image version:

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.25 --record
```

Check rollout status:

```
kubectl rollout status deployment/nginx-deployment
```

Step 5 — Verify Update

```
kubectl get pods -o wide
```

You'll see new Pods created with the updated image version.

Step 6 — Rollback to Previous Version

```
kubectl rollout undo deployment/nginx-deployment
```

Verify rollback:

```
kubectl get pods
```

Step 7 — Scale the Deployment

```
kubectl scale deployment nginx-deployment --replicas=5
kubectl get pods
```

Step 8 — Delete the Deployment

```
kubectl delete deployment nginx-deployment
```

Verify:

```
kubectl get pods
```

✅ **Outcome:** You have learned how to create, update, rollback, and scale a Deployment. This is the recommended controller for managing stateless applications in Kubernetes.

Verification Summary

Controller Type	Key Command	Description
Replication Controller	`kubectl apply -f replication-controller.yaml`	Ensures desired Pod count
ReplicaSet	`kubectl apply -f replicaset.yaml`	Modern version of RC with better selectors
Deployment	`kubectl apply -f deployment.yaml`	Manages ReplicaSets with rolling updates

✅**Final Outcome:** After completing these three labs, participants will clearly understand the differences and evolution from ReplicationController → ReplicaSet → Deployment, and how each contributes to reliable application management in Kubernetes.