🧪 Lab – Core Workloads & Configuration Management (Module 4)

🎯 Objectives

- Create and scale applications using **Deployments**
- Simulate **rolling updates and rollbacks**
- Deploy a **StatefulSet** and observe stable identities and storage
- Implement **liveness** and **readiness probes**
- Use **ConfigMaps** and **Secrets** to inject configuration into pods

---

🔧 Prerequisites

- A running **Kind cluster** with access to `kubectl`

🛠️ Step 1 – Create a Deployment

- Create a deployment named `webapp` with 3 replicas of `nginx`
- Expose port 80

```
kubectl create deployment webapp \
  --image=nginx --replicas=3 \
  --dry-run=client -o yaml > deployment.yaml
kubectl apply -f deployment.yaml
kubectl expose deployment webapp --port=80 --target-port=80
```

🔁 Step 2 – Simulate a Rolling Update & Rollback

- Update image to a newer version or invalid one
- Observe rollout behavior and rollback

```
kubectl set image deployment/webapp nginx=nginx:1.25
kubectl rollout status deployment/webapp
kubectl rollout undo deployment/webapp
```

📦 Step 3 – Deploy a StatefulSet

- Create a `StatefulSet` with headless service, and PVC
- Observe pod names and volume claims

```
kubectl apply -f statefulset.yaml  # Provided in repo
kubectl get pods -o wide
kubectl get pvc

kubectl run -it dnsutils --image=busybox:1.28 --restart=Neve
r --rm -- nslookup webapp
```

🩺 Step 4 – Add Liveness and Readiness Probes

- Patch the deployment or create a new one with HTTP probes on port 80

```
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10

        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 3
          periodSeconds: 5
```

Apply and observe pod status using `kubectl describe pod`

To see the probe in action you can patch the deployment to add a command that simulates a failure:

```
kubectl patch deployment webapp --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/livenessProbe/httpGet/path", "value": "/invalid"}]'
```

fix the liveness probe by reverting the patch:

```
kubectl patch deployment webapp --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/livenessProbe/httpGet/path", "value": "/"}]'
```

🔐 Step 5 – Inject Configuration with ConfigMap and Secret

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30

```
kubectl create configmap app-config --from-literal=ENV=produ
ction
kubectl create secret generic db-creds \
  --from-literal=username=admin --from-literal=password=pass
w0rd
```

Mount into pod or expose via env vars:

```
        envFrom:
          - configMapRef:
              name: app-config
          - secretRef:
              name: db-creds
```

Challenge - Mount ConfigMap and Secret in a Volume

- Modify the deployment to mount the `ConfigMap` and `Secret` as files in a volume

▶ ▶ **Deployment YAML Snippet**

```
          volumeMounts:
            - name: config-volume
              mountPath: /etc/config
            - name: secret-volume
              mountPath: /etc/secret

        volumes:
          - name: config-volume
            configMap:
              name: app-config
          - name: secret-volume
            secret:
              secretName: db-creds
```

🧼 Cleanup

```
kubectl delete deployment webapp
kubectl delete configmap app-config
kubectl delete secret db-creds
kubectl delete pvc --all
kubectl delete svc webapp
kubectl delete statefulset mysql
kubectl delete svc mysql
```

✅ End of Lab – You’ve practiced core workloads, self-healing, and config injection!

Edixos Labs
hello@edixos.com
+33 6 47 43 99 30