Kubernetes ConfigMap Hands-On Lab

Author : Sandeep Kumar Sharma

Introduction: What is a ConfigMap?

In Kubernetes, a **ConfigMap** is used to store configuration data in key-value pairs. Instead of hardcoding configuration values directly into your Pod or container image, ConfigMaps let you **decouple configuration** from your application code. This makes your deployments more flexible and easier to manage.

Why Use a ConfigMap?

- To separate configuration from the container image.
- To manage environment-specific settings (e.g., database URLs, API keys, file paths).
- To reuse the same image in multiple environments (dev, test, prod) by simply changing configuration.

Common Use Cases:

1. Providing configuration values as **environment variables** to containers.
2. Mounting configuration files directly into containers.
3. Managing application-specific configuration dynamically.

Lab Objective:

By the end of this lab, you'll learn: - How to create a ConfigMap using commands. - How to create a ConfigMap using YAML. - How to consume ConfigMaps in Pods (as environment variables and as mounted files).

Part 1: Create a ConfigMap using Command Line

Step 1 — Create a ConfigMap

```
kubectl create configmap app-config --from-literal=APP_MODE=production --from-literal=APP_VERSION=v1.0
```

Explanation: - `app-config` is the ConfigMap name. - `--from-literal` creates key-value pairs directly from the command line.

Step 2 — View the ConfigMap

```
kubectl get configmap app-config -o yaml
```

This will display:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_MODE: production
  APP_VERSION: v1.0
```

Part 2: Create a ConfigMap using YAML File

Step 1 — Write ConfigMap Definition

Create a file named `configmap.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  labels:
    app: demo
 data:
  APP_NAME: "MyDemoApp"
  APP_ENV: "Development"
  WELCOME_MSG: "Welcome to ConfigMap Lab!"
```

Step 2 — Apply the ConfigMap

```
kubectl apply -f configmap.yaml
```

Step 3 — Verify ConfigMap

```
kubectl get configmap my-config -o yaml
```

Part 3: Use ConfigMap in Pod as Environment Variables

Step 1 — Create Pod Definition

Create a file named `configmap-env-pod.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-env-pod
spec:
  containers:
  - name: demo-container
    image: busybox
    command: ['sh', '-c', 'echo App: $APP_NAME; echo Env: $APP_ENV; echo
Message: $WELCOME_MSG; sleep 3600']
    envFrom:
    - configMapRef:
        name: my-config
```

Step 2 — Apply the YAML File

```
kubectl apply -f configmap-env-pod.yaml
```

Step 3 — Check the Logs

```
kubectl logs configmap-env-pod
```

Expected output:

```
App: MyDemoApp
Env: Development
Message: Welcome to ConfigMap Lab!
```

Step 4 — Delete the Pod

```
kubectl delete pod configmap-env-pod
```

Part 4: Use ConfigMap in Pod as Mounted Volume

Step 1 — Create ConfigMap with File Data

```
kubectl create configmap web-config --from-file=index.html=/usr/share/nginx/
html/index.html --from-literal=FOOTER=Kubernetes-Lab
```

Or using YAML:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: web-config
 data:
  index.html: |
    <html>
    <head><title>ConfigMap Mounted Example</title></head>
    <body>
      <h1>This page is served using a ConfigMap file!</h1>
      <p>Environment: Training</p>
    </body>
    </html>
```

Save this as `web-config.yaml` and apply:

```
kubectl apply -f web-config.yaml
```

Step 2 — Create a Pod that Mounts the ConfigMap

Create a file named `configmap-volume-pod.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-volume-pod
spec:
  containers:
  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: config-volume
      mountPath: /usr/share/nginx/html
  volumes:
```

```
  - name: config-volume
    configMap:
      name: web-config
```

Step 3 — Deploy the Pod

```
kubectl apply -f configmap-volume-pod.yaml
```

Step 4 — Verify ConfigMap Mounted

```
kubectl exec -it configmap-volume-pod -- /bin/bash
cat /usr/share/nginx/html/index.html
```

You'll see the content served from your ConfigMap.

Step 5 — Access via Port Forwarding (Optional)

```
kubectl port-forward pod/configmap-volume-pod 8080:80
```

Now open your browser and navigate to `http://localhost:8080` — you should see your custom HTML page.

Step 6 — Delete the Pod and ConfigMap

```
kubectl delete pod configmap-volume-pod
kubectl delete configmap web-config my-config app-config
```

Verification Summary

Task	Command	Description
Create ConfigMap (CLI)	`kubectl create configmap app-config --from-literal=...`	Create ConfigMap quickly
View ConfigMap	`kubectl get configmap <name> -o yaml`	Check content
Use as env vars	`envFrom.configMapRef`	Inject ConfigMap values

Task	Command	Description
Use as file	`volumes.configMap.name`	Mount as file in container
Delete	`kubectl delete configmap <name>`	Cleanup

✅ **Outcome:** After completing this lab, participants will understand how to create and use ConfigMaps effectively for configuration management in Kubernetes. They'll also be confident in using both CLI and YAML methods for real-world deployments.

Next Step: Move on to **Kubernetes Secrets Lab** to learn how to manage sensitive data securely.