Kubernetes Secrets Hands-On Lab

Author : Sandeep Kumar Sharma

Introduction: What is a Secret in Kubernetes?

A **Secret** in Kubernetes is used to store **sensitive information** such as passwords, tokens, SSH keys, or any confidential configuration data. Storing this data as a Secret ensures that it's not exposed directly in your Pod definition or container image.

While a ConfigMap holds non-sensitive data, a Secret is specifically designed for sensitive content and can be mounted into Pods securely.

Why Use a Secret?

- To store sensitive data like database passwords, tokens, or certificates.
- To avoid embedding confidential information directly in application code or Pod specs.
- To control access using Kubernetes RBAC (Role-Based Access Control).

Common Use Cases:

1. Storing application credentials (e.g., DB_USER, DB_PASSWORD).
2. Managing SSH keys or TLS certificates.
3. Passing secrets as environment variables or mounted files to containers.

Lab Objective:

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

Part 1: Create a Secret using Command Line

Step 1 — Create the Secret

```
kubectl create secret generic db-secret --from-literal=DB_USER=admin --from-
literal=DB_PASSWORD=MyP@ssw0rd123
```

Explanation: - `db-secret` is the Secret name. - `--from-literal` defines key-value pairs for the Secret.

Step 2 — Verify the Secret

```
kubectl get secret db-secret
```

You'll see:

```
NAME        TYPE      DATA   AGE
db-secret   Opaque    2      10s
```

Step 3 — View Secret Data (Base64 Encoded)

```
kubectl get secret db-secret -o yaml
```

You'll notice that the values are **base64 encoded**:

```
data:
  DB_USER: YWRtaW4=
  DB_PASSWORD: TXlQQHNzdzByZDEyMw==
```

To decode manually:

```
echo 'YWRtaW4=' | base64 --decode
```

Part 2: Create a Secret using YAML File

Step 1 — Write Secret Definition

Create a file named `secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
 type: Opaque
 data:
  username: YWRtaW4=
  password: TXlQYXNzd29yZDEyMw==
```

Note: You must encode the values in base64 format before adding them to the YAML.

To encode your own values:

```
echo -n 'admin' | base64
# Output: YWRtaW4=
echo -n 'MyPassword123' | base64
# Output: TXlQYXNzd29yZDEyMw==
```

Step 2 — Apply the YAML File

```
kubectl apply -f secret.yaml
```

Step 3 — Verify Secret Creation

```
kubectl get secret app-secret -o yaml
```

Part 3: Use Secret as Environment Variables in a Pod

Step 1 — Create Pod Definition

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

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: demo-container
    image: busybox
    command: ['sh', '-c', 'echo User: $DB_USER; echo Password: $DB_PASSWORD;
sleep 3600']
    env:
    - name: DB_USER
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: DB_USER
    - name: DB_PASSWORD
      valueFrom:
```

```
        secretKeyRef:
          name: db-secret
          key: DB_PASSWORD
```

Step 2 — Apply the YAML File

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

Step 3 — Check the Pod Logs

```
kubectl logs secret-env-pod
```

Expected output:

```
User: admin
Password: MyP@sswOrd123
```

Step 4 — Delete the Pod

```
kubectl delete pod secret-env-pod
```

Part 4: Use Secret as a Mounted Volume in Pod

Step 1 — Create YAML Definition

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

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-volume-pod
spec:
  containers:
  - name: demo-container
    image: busybox
    command: ['sh', '-c', 'ls /etc/secret-data; cat /etc/secret-data/DB_USER;
cat /etc/secret-data/DB_PASSWORD; sleep 3600']
    volumeMounts:
    - name: secret-volume
```

```
      mountPath: /etc/secret-data
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: db-secret
```

Step 2 — Deploy the Pod

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

Step 3 — Exec into the Pod

```
kubectl exec -it secret-volume-pod -- /bin/sh
```

Inside the pod, run:

```
ls /etc/secret-data
cat /etc/secret-data/DB_USER
cat /etc/secret-data/DB_PASSWORD
```

You'll see your decoded secret values.

Step 4 — Delete the Pod and Secrets

```
kubectl delete pod secret-volume-pod
kubectl delete secret db-secret app-secret
```

Verification Summary

Task	Command	Description
Create Secret (CLI)	`kubectl create secret generic db-secret --from-literal=...`	Create a Secret quickly
Encode to base64	`echo -n 'value' | base64`	Encode sensitive data
Use in Pod (env)	`secretKeyRef`	Inject secrets as environment variables

Task	Command	Description
Use in Pod (volume)	`volumes.secret.secretName`	Mount secrets as files
Decode value	`echo 'value' | base64 --decode`	Decode for verification
Delete	`kubectl delete secret <name>`	Cleanup

✅**Outcome:** After completing this lab, participants will understand how to securely manage and consume sensitive information using Kubernetes Secrets. They will know how to use Secrets both as environment variables and as mounted files for applications.

Next Step: Move on to **Kubernetes ConfigMap + Secret Integration Lab** to understand how to use both together for dynamic configuration management.