**1. ConfigMaps in Kubernetes**

ConfigMaps store non-sensitive configuration data as key-value pairs. They are mainly used for passing environment variables, command-line arguments, or configuration files to your application.

**Example 1: Creating a ConfigMap from a YAML File**

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

database\_url: "mysql://db-user:password@mysql-service:3306/mydb"

log\_level: "debug"

max\_connections: "100"

Apply the ConfigMap:

kubectl apply -f configmap.yaml

Check if it was created:

kubectl get configmap

kubectl describe configmap app-config

kubectl get configmap app-config -o jsonpath='{.data}'

**Using ConfigMap in a Pod**

You can use a ConfigMap in three ways:

1. **As Environment Variables**
2. **As Command-Line Arguments**
3. **As a Volume Mount (Files)**

**Example 2: Using ConfigMap as Environment Variables**

apiVersion: v1

kind: Pod

metadata:

name:

spec:

containers:

- name: myapp

command: ["/bin/sh", "-c", "printenv"]

image: busybox

env:

- name: DATABASE\_URL

valueFrom:

configMapKeyRef:

name: app-config

key: database\_url

- name: LOG\_LEVEL

valueFrom:

configMapKeyRef:

name: app-config

key: log\_level

**Example 3: Mounting ConfigMap as a Volume**

apiVersion: v1

kind: Pod

metadata:

name: configmap-volume-demo

spec:

volumes:

- name: config-volume

configMap:

name: app-config

containers:

- name: myapp

image: busybox

volumeMounts:

- name: config-volume

mountPath: /etc/config

After applying, you can check:

kubectl exec -it configmap-volume-demo -- ls /etc/config

**2. Secrets in Kubernetes**

Secrets store **sensitive** data like passwords, API keys, and certificates.

**Example 1: Creating a Secret from a YAML File**

apiVersion: v1

kind: Secret

metadata:

name: db-secret

type: Opaque

data:

username: dXNlcm5hbWU= # Base64 encoded "username"

password: cGFzc3dvcmQ= # Base64 encoded "password"

Apply the Secret:

kubectl apply -f secret.yaml

Check the created secret:

kubectl get secret

kubectl describe secret db-secret

**Note:** To manually encode data in base64:

echo -n 'username' | base64

echo -n 'password' | base64

**Using Secrets in a Pod**

Just like ConfigMaps, Secrets can be used in three ways.

**Example 2: Using Secrets as Environment Variables**

apiVersion: v1

kind: Pod

metadata:

name: secret-env-demo

spec:

containers:

- name: myapp

image: busybox

env:

- name: DB\_USER

valueFrom:

secretKeyRef:

name: db-secret

key: username

- name: DB\_PASS

valueFrom:

secretKeyRef:

name: db-secret

key: password

**Example 3: Mounting Secrets as a Volume**

apiVersion: v1

kind: Pod

metadata:

name: secret-volume-demo

spec:

volumes:

- name: secret-volume

secret:

secretName: db-secret

containers:

- name: myapp

image: busybox

volumeMounts:

- name: secret-volume

mountPath: "/etc/secret"

After applying, verify:

kubectl exec -it secret-volume-demo -- ls /etc/secret

**3. Creating ConfigMaps and Secrets from CLI**

**ConfigMap from CLI**

kubectl create configmap app-config --from-literal=database\_url="mysql://db-user@mysql-service:3306/mydb" --from-literal=log\_level="debug"

kubectl create configmap app-config-file --from-file=app.properties

**Secret from CLI**

kubectl create secret generic db-secret --from-literal=username='admin' --from-literal=password='securepass'

kubectl create secret generic tls-secret --from-file=tls.crt=/path/to/cert.crt --from-file=tls.key=/path/to/cert.key

**4. Viewing and Editing ConfigMaps and Secrets**

kubectl get configmap app-config -o yaml

kubectl edit configmap app-config

kubectl get secret db-secret -o yaml

kubectl edit secret db-secret

**5. Deleting ConfigMaps and Secrets**

kubectl delete configmap app-config

kubectl delete secret db-secret

**Key Differences Between ConfigMaps and Secrets**

| **Feature** | **ConfigMap** | **Secret** |
| --- | --- | --- |
| Stores sensitive data? | ❌ No | ✅ Yes |
| Base64 encoding required? | ❌ No | ✅ Yes |
| Access via environment variables? | ✅ Yes | ✅ Yes |
| Access via volume mounts? | ✅ Yes | ✅ Yes |
| Used for? | Configuration data | Sensitive data |