.

Lesson 03 Demo 11

Configuring ConfigMaps

Objective: To configure ConfigMaps to enhance the flexibility, security, and manageability of your applications, making them adaptable to different environments

Tools required: kubeadm, kubectl, kubelet, and containerd

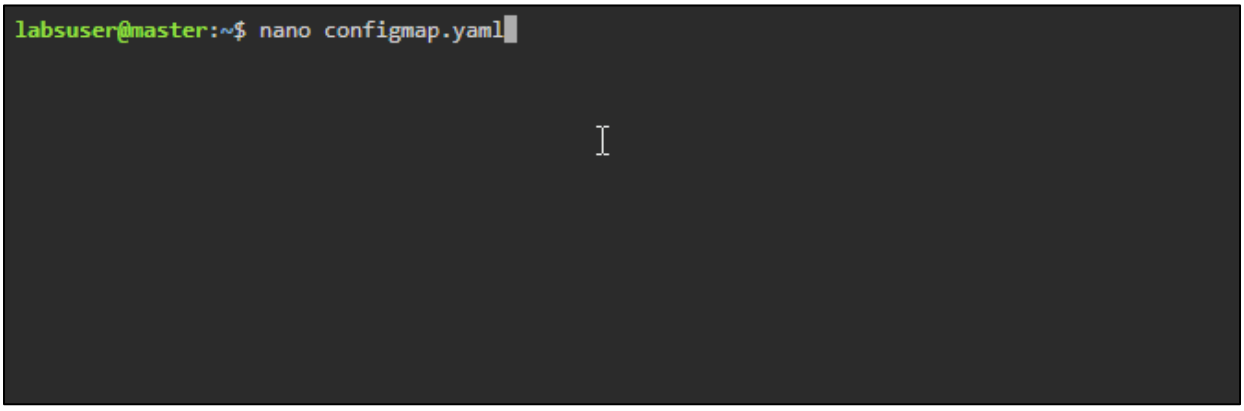
Prerequisites: A Kubernetes cluster (refer to Demo 01 from Lesson 01 for setting up a cluster)

Steps to be followed:

1. Add a ConfigMap entry to the pod
2. Create pods with services

Step 1: Add a ConfigMap entry to the pod

1.1 On the master node, enter the **nano configmap.yaml** command to create a YAML file



.

1.2 Enter the following code in the YAML file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-configmap
data:
  database: mongodb
  database_uri: mongodb://localhost:27017
```

```
  GNU nano 6.2                                                    configmap.yaml *
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-configmap
data:
  database: mongodb
  database_uri: mongodb://localhost:27017
```

1.3 Create a ConfigMap using the command below:
kubectl create -f configmap.yaml

```
labsuser@master:~$ kubectl create -f configmap.yaml
configmap/example-configmap created
labsuser@master:~$
```

.

1.4 Verify the ConfigMap state using the following command:
kubectl get configmap

```
labsuser@master:~$ kubectl get configmap
NAME                DATA   AGE
example-configmap   2      26s
kube-root-ca.crt    1      85m
labsuser@master:~$
```

1.5 Run the **nano configpod.yaml** command to create a YAML file

```
labsuser@master:~$ nano configpod.yaml
```

.

1.6 Enter the following code in the **configpod.yaml** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:1.7.9
      envFrom:
        - configMapRef:
            name: example-configmap
```

```
  GNU nano 6.2                                                    configpod.yaml *
apiVersion: v1
kind: Pod
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:1.7.9
      envFrom:
        - configMapRef:
            name: example-configmap
```

1.7 Create a pod using the following command:
kubectl create -f configpod.yaml

```
labsuser@master:~$ kubectl create -f configpod.yaml
pod/pod-env-var created
labsuser@master:~$
```

.

1.8 Verify the pod state by running the following command:
kubectl get pods

```
labsuser@master:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
pod-env-var   1/1     Running   0          45s
labsuser@master:~$
```

Step 2: Create pods with services

2.1 Run the **nano config-svc.yaml** command to create a YAML file

```
labsuser@master:~$ nano config-svc.yaml
```

.

2.2 Enter the following code in the YAML file:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-env12
spec:
  containers:
    - name: env-var-configmap
      image: nginx:1.7.9
      env:
        - name: testenv
          valueFrom:
            configMapKeyRef:
              name: example-configmap
              key: database
```

```
  GNU nano 6.2                                                    config-svc.yaml *
apiVersion: v1
kind: Pod
metadata:
  name: pod-env12
spec:
  containers:
    - name: env-var-configmap
      image: nginx:1.7.9
      env:
        - name: testenv
          valueFrom:
            configMapKeyRef:
              name: example-configmap
              key: database
```

2.3 Run the following command to create a pod with service:
kubectl create -f config-svc.yaml

```
labsuser@master:~$ kubectl create -f config-svc.yaml
pod/pod-env12 created
labsuser@master:~$
```

.

2.4 Verify the pod state by running the following command:
kubectl get pods

```
labsuser@master:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
pod-env-var   1/1     Running   0          5m45s
pod-env12     1/1     Running   0          20s
labsuser@master:~$
```

2.5 To access the container and verify the database, run the following commands:
kubectl exec -it pod-env12 bash
env
env | grep database

```
labsuser@master:~$ kubectl  exec -it pod-env12 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@pod-env12:/# env
HOSTNAME=pod-env12
TERM=xterm
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.96.0.1
testenv=mongodb
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
NGINX_VERSION=1.7.9-1~wheezy
SHLVL=1
HOME=/root
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
_=/usr/bin/env
root@pod-env12:/# env | grep database
```

2.6 Run the **nano configfile.yaml** command to create a YAML file

```
labsuser@master:~$ nano configfile.yaml
```

.

2.7 Enter the following code in the YAML file:

```
apiVersion: v1
kind: Pod
metadata:
  name: testconfig
spec:
  containers:
    - name: test
      image: docker.io/httpd
      volumeMounts:
      - name: config-volume
        mountPath: /tmp/myenvs/
  volumes:
    - name: config-volume
      configMap:
        name: example-configmap
  restartPolicy: Never
```

.

2.8 Run the following commands to create a pod and verify its state:

kubectl create -f configfile.yaml

kubectl get pods

```
labsuser@master:~$ kubectl create -f configfile.yaml
pod/testconfig created
labsuser@master:~$
```

```
labsuser@master:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
pod-env-var   1/1     Running   0          11m
pod-env12     1/1     Running   0          5m45s
testconfig    1/1     Running   0          27s
labsuser@master:~$
```

2.9 Access the pod by running the following command:

kubectl exec -it testconfig bash

```
labsuser@master:~$ kubectl exec -it testconfig bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@testconfig:/usr/local/apache2#
```

By following the steps above, you have successfully created and applied ConfigMaps to configure environment variables and volumes for your pods in Kubernetes. This approach allows you to decouple configuration data from your application, making your deployments more flexible and scalable.