Create and Use ConfigMaps and Secrets in a Kubernetes Pod

Table of Contents

Introduction

Kubernetes provides two essential resources for managing configuration data and sensitive information: **ConfigMaps** and **Secrets**. These resources allow you to externalize application configuration and securely store sensitive data such as passwords, API keys, and other credentials.

- **ConfigMaps**: Used to store non-sensitive configuration data in key-value pairs, such as environment variables or configuration files.
- **Secrets**: Used to store sensitive information, with additional features like base64 encoding to enhance security.

In this lab, you will learn how to:

- Create a ConfigMap to store configuration data.
- Create a Secret to store sensitive information.
- Use both ConfigMaps and Secrets within a Kubernetes pod to configure and secure your application.

Problem Statement

As your applications grow, it becomes important to separate the application code from its configuration data. You may also need to store sensitive information securely, without hardcoding it in your application.

This lab aims to show you how to manage this external configuration through **ConfigMaps** and **Secrets**, and how to use them inside a pod to ensure that your applications can access configuration data and secrets securely.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-03) is required before proceeding with Lab Guide-04.

- A running Kubernetes cluster on Minikube.

- `kubectl` installed and configured to interact with your Minikube cluster.
- Basic understanding of Kubernetes pods and deployments.

Setup Instructions

Step 1: Create a ConfigMap

A ConfigMap is a way to store key-value pairs for configuration. In this example, we will create a ConfigMap that stores some environment variables for an NGINX pod.

1. **Create a ConfigMap YAML File**

   Create a file named `nginx-configmap.yaml` with the following content:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: nginx-config
   data:
     welcome-message: "Welcome to NGINX running in Kubernetes!"
     log-level: "info"
   ```

   - **welcome-message**: Stores a custom message for the NGINX web server.
   - **log-level**: Specifies the log level for NGINX.

2. **Apply the ConfigMap**

   Run the following command to create the ConfigMap in your cluster:

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

3. **Verify the ConfigMap**

   Check that the ConfigMap was created successfully:

   ```
   kubectl get configmaps
   ```

   ```
   PS C:\Users\Administrator> kubectl apply -f nginx-configmap.yaml
   configmap/nginx-config created
   PS C:\Users\Administrator> kubectl get configmaps
   NAME               DATA   AGE
   kube-root-ca.crt   1      54m
   nginx-config       2      27s
   ```

Step 2: Create a Secret

A Secret is used to store sensitive information like passwords or API keys. We will create a Secret to store a basic authentication password for NGINX.

1. **Create a Secret YAML File**

   Create a file named `nginx-secret.yaml` with the following content. The secret data needs to be base64 encoded.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: nginx-secret
   data:
     username: YWRtaW4=   # base64 for "admin"
     password: cGFzc3dvcmQ=   # base64 for "password"
   ```

   - **username**: Base64 encoded username (`admin`).
   - **password**: Base64 encoded password (`password`).

2. **Apply the Secret**

   Run the following command to create the Secret:

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

3. **Verify the Secret**

   List all the secrets to ensure it was created:

   ```
   kubectl get secrets
   ```

   ```
   PS C:\Users\Administrator> kubectl apply -f nginx-secret.yaml
   secret/nginx-secret created
   PS C:\Users\Administrator> kubectl get secrets
   NAME           TYPE     DATA   AGE
   nginx-secret   Opaque   2      10s
   ```

   You can also decode the secret values if needed:

   ```
   kubectl get secret nginx-secret -o yaml
   ```

```
PS C:\Users\Administrator> kubectl get secret nginx-secret -o yaml
apiVersion: v1
data:
  password: cGFzc3dvcmQ=
  username: YWRtaW4=
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"password":"cGFzc3dvcmQ=","username":"YWRtaW4="},"kind":"Secret","metadata":{"annotations":{},"name":"nginx-secret",
"namespace":"default"}}
  creationTimestamp: "2024-11-20T10:31:12Z"
  name: nginx-secret
  namespace: default
  resourceVersion: "2972"
  uid: b99107bf-56bf-4294-98ac-5b0bd309a33e
type: Opaque
```

Step 3: Use ConfigMaps and Secrets in a Pod

Next, we'll create an NGINX pod that uses the ConfigMap and Secret we just created. The ConfigMap will be used for environment variables, and the Secret will be mounted as a file.

1. **Create the Pod YAML File**

   Create a file named `nginx-pod-config-secret.yaml` with the following content:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    env:
    - name: WELCOME_MESSAGE
      valueFrom:
        configMapKeyRef:
          name: nginx-config
          key: welcome-message
    - name: LOG_LEVEL
      valueFrom:
        configMapKeyRef:
          name: nginx-config
          key: log-level
    volumeMounts:
    - name: secret-volume
      mountPath: "/etc/nginx/secret"
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: nginx-secret
```

   Key points:

   - **env**: Environment variables are set using values from the ConfigMap (`nginx-config`).

- **volumeMounts**: The Secret is mounted into the container as a volume at `/etc/nginx/secret`.

2. **Apply the Pod Configuration**

   Run the following command to create the pod:

```
kubectl apply -f nginx-pod-config-secret.yaml
```

3. **Verify the Pod**

   Ensure that the pod is running:

```
kubectl get pods
```

```
PS C:\Users\Administrator> kubectl apply -f nginx-pod-config-secret.yaml
pod/nginx-pod configured
PS C:\Users\Administrator> kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          10m
```

4. **Inspect the Environment Variables and Secrets**

   Once the pod is running, you can exec into the pod to verify that the environment variables and secrets are being used correctly:

```
kubectl exec -it nginx-pod -- /bin/bash
```

   Check the environment variables:

```
echo $WELCOME_MESSAGE
echo $LOG_LEVEL
```

```
PS C:\Users\Administrator> kubectl exec -it nginx-pod -- /bin/bash
root@nginx-pod:/# echo $WELCOME_MESSAGE
Welcome to NGINX running in Kubernetes!
root@nginx-pod:/# echo $LOG_LEVEL
info
```

   Check the contents of the secret file:

```
cat /etc/nginx/secret/username
cat /etc/nginx/secret/password
```

```
PS C:\Users\Administrator> kubectl exec -it nginx-pod -- /bin/bash
root@nginx-pod:/# cat /etc/nginx/secret/username
adminroot@nginx-pod:/# cat /etc/nginx/secret/password
passwordroot@nginx-pod:/#
```

References

- Kubernetes ConfigMaps Documentation
- Kubernetes Secrets Documentation
- Minikube Documentation
