Lab 10.2: Secrets and ConfigMaps
ID720 Virtualisation

Introduction

1 Create a project

Some of the value of tools like containers and Kubernetes is the way they enable rapid and flexible deployment. In order to do that we need ways to inject configuration variables into our containers. Kubernetes gives us two tools to do this: *Secrets* and *ConfigMaps*.

Carry out this lab's tasks on GKE. You will need to edit some Kubernetes manifest files. Google Cloud Shell includes a Cloud Shell Editor that you may find useful for this.

2 Launch a cluster

To start you will need a Kubernetes cluster. Launch a cluster named `configreader-lab` following the procedure from lab 10.1.

3 Create a Secret

Kubernetes *secrets* are collections of key-value pairs that can be used by running pods to obtain sensitive information like database usernames and passwords. Note that secrets are not encrypted and anyone with access to your cluster can get the values of the secrets. However, they do let us extract sensitive information from our applications, making them easier to manage safely. Also, Kubernetes will protect secrets from accidental disclosure. See `https://kubernetes.io/docs/concepts/configuration/secret/` for more information.

In your cloud shell, create a secret manifest file like the one below. Name your manifest file `configreader-secret.yml`

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecrets
type: Opaque
data:
  secret: Ym9ub2JvCg==
```

This will create a secret called `mysecrets` with one key-value pair in it. Note that the value of the secret is base64 encoded.

Create your secret with the command

```
kubectl apply -f configreader-secret.yml
```

4 Create a ConfigMap

ConfigMaps are collections of key-value pairs, just like Secrets. The main difference is that Kubernetes doesn't treat the information as being sensitive. See `https://kubernetes.io/docs/concepts/configuration/configmap/` for more information.

In your cloud shell, create a ConfigMap manifest file like the one below. Name your manifest file `configreader-map.yml`

```
kind: ConfigMap
apiVersion: v1
metadata:
  name:  configs
  namespace: default
data:
  CONFIG1: monkey
  CONFIG2: lemur
```

Create your map with the command

```
kubectl apply -f configreader-map.yml
```

5 Create a deployment using our secrets and maps

We have seen how to create deployments before. The new thing we will add today is some configuration that lets our pod use our ConfigMap and Secret to populate environment variables.

In your cloud shell, create a deployment manifest file like the one below. Name your manifest file `configreader-deployment.y`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  generation: 1
  labels:
    run: configreader
  name: configreader
  namespace: default
spec:
  progressDeadlineSeconds: 420
  minReadySeconds: 10
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      run: configreader
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      labels:
        run: configreader
    spec:
      containers:
        - image: tclark/configreader
          imagePullPolicy: Always
          name: configreader
          resources:
            requests:
              cpu: "10m"
              memory: "64Mi"
          env:
            - name: SECRET
              valueFrom:
                secretKeyRef:
                  name: mysecrets
                  key: secret
          envFrom:
          - configMapRef:
              name: configs
          ports:
            - containerPort: 5000
              name: http
              protocol: TCP
      dnsPolicy: ClusterFirst
      restartPolicy: Always
```

Note the `env` and `envFrom` sections above. Also, be aware that our pod won't be launched if its required Secrets and ConfigMaps are not present.

Launch and expose your deployments with the command

```
kubectl apply -f configreader-deployment.yml
kubectl expose deployment configreader --type LoadBalancer --port 80 --target-port 5000
```

Once everything is running you can get the IP address of your load balancer and view the results in your browser.

Don't forget to delete your load balancer and cluster at the end of the lab!