Kubernetes

Secrets, ConfigMaps, and Downward API Challenge Solutions

Try the following exercises on your own, using the class slides and the knowledge from this lab to guide you.

First, recreate the `special-config` ConfigMap if you deleted it:

```
~/dapi$ cd ~/appconfig

~/appconfig$ kubectl delete configmap special-config

~/appconfig$ kubectl create configmap special-config \
--from-literal=special.type=charm --from-literal=special.how=very

configmap/special-config created

~/appconfig$
```

CHALLENGE 1

Mount the values of the `special-config` configmap as environment variables `SPECIAL_LEVEL_KEY` and `SPECIAL_TYPE_KEY`, and modify the `challenge-pod` to output the value of those variables to stdout

- Configure the pod with the command `/bin/sh -c "echo $SPECIAL_LEVEL_KEY $SPECIAL_TYPE_KEY"`

Using our existing ConfigMap called `special-config`.

```
~/appconfig$ kubectl get configmaps

NAME               DATA   AGE
kube-root-ca.crt   1      2d12h
special-config     2      50s

~/appconfig$
```

We are now going to use our ConfigMap as part of the container command. The first imperative command creates the base manifest file:

```
~/appconfig$ kubectl run challenge-pod --dry-run=client -o yaml  \
--image busybox --restart Never \
--env SPECIAL_LEVEL_KEY=changeMe --env SPECIAL_TYPE_KEY=changeMe \
--command /bin/sh -- -c "echo \$SPECIAL_LEVEL_KEY \$SPECIAL_TYPE_KEY" >
cli-pod.yaml

~/appconfig$
```

Make the necessary edits to the environment variables to use the ConfigMap as environment variables as follows.

```
~/appconfig$ nano cli-pod.yaml && cat cli-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: challenge-pod
  name: challenge-pod
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - echo $SPECIAL_LEVEL_KEY $SPECIAL_TYPE_KEY
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:                # Change this
        configMapKeyRef:        # Add this
          name: special-config  # Add this
          key: special.how      # Add this
    - name: SPECIAL_TYPE_KEY
      valueFrom:                # Change this
        configMapKeyRef:        # Add this
          name: special-config  # Add this
          key: special.type     # Add this
    image: busybox
    name: challenge-pod
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

```
~/appconfig$
```

Like the challenge-pod from the previous step, the container will pull the values of SPECIAL_LEVEL_KEY and SPECIAL_TYPE_KEY from the configmap. This time, however, it will use the container's shell to dump the values of those environment variables.

Create the cli-pod:

```
~/appconfig$ kubectl apply -f cli-pod.yaml

pod/challenge-pod created

~/appconfig$
```

```
~/appconfig$ kubectl get pods

NAME            READY   STATUS      RESTARTS   AGE
challenge-pod   0/1     Completed   0          3s

~/appconfig$
```

With the pod created (and completed), check its log to see if the cli command was run and that the environment variables were dumped to its STDOUT:

```
~/appconfig$ kubectl logs challenge-pod

very charm

~/appconfig$
```

Once configMap values are declared as variables, you will be able to consume them as you would any other environment variable inside any pod's container(s).

Remove the challenge-pod again:

```
~/appconfig$ kubectl delete pod challenge-pod

pod "challenge-pod" deleted

~/appconfig$
```

CHALLENGE 2

Modify the challenge-pod manifest to mount the `special-config` configmap as a volume

- Configure the pod with the command `/bin/sh -c "cat /etc/config/special.how"`

- You should be able to use `kubectl logs challenge-pod` to verify that the contents of special.how are printed

The following imperative command creates the base manifest file in yaml:

```
~/appconfig$ kubectl run --dry-run=client -o yaml challenge-pod \
--image busybox --restart Never \
--command /bin/sh -- -c "cat /etc/config/special.how" > vol-cm.yaml

~/appconfig$
```

To declare ConfigMaps as volumes, make sure you use the `.spec.volumes.name` and `.spec.containers.volumeMounts.name` keys to refer to the `special-config` ConfigMap. Edit the manifest to add the `volume` and `volumeMounts` configurations to mount the configMap to the container:

```
~/appconfig$ nano vol-cm.yaml && cat vol-cm.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: challenge-pod
  name: challenge-pod
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - cat /etc/config/special.how
    image: busybox
    name: challenge-pod
    resources: {}
    volumeMounts:                 # Add this
    - name: config-volume         # Add this
      mountPath: /etc/config      # Add this
  volumes:                        # Add this
  - name: config-volume           # Add this
    configMap:                    # Add this
      name: special-config        # Add this
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

```
~/appconfig$
```

In this spec, the special-config ConfigMap is mounted as a volume on the pod. The volume is then mounted in the pod's container at /etc/config. The container's shell will then read the file special.how that should be mounted there:

```
~/appconfig$ kubectl apply -f vol-cm.yaml

pod/challenge-pod created

~/appconfig$ kubectl get pods

NAME            READY   STATUS      RESTARTS   AGE
challenge-pod   0/1     Completed   0          4s

~/appconfig$
```

Check the logs to see if the command succeeded:

```
~/appconfig$ kubectl logs challenge-pod

very

~/appconfig$
```

That worked! When a ConfigMap is mounted in a volume, each key in the volume is treated as a new file that can be found where the ConfigMap was mounted in the pod's container filesystems.

```
~/appconfig$ kubectl delete pod challenge-pod

pod "challenge-pod" deleted

~/appconfig$ kubectl delete cm special-config

configmap "special-config" deleted

~/appconfig$
```

CHALLENGE 3

Deploy a pod running the `httpd:2.4` image that uses the following `index.html` document:

```
~/appconfig$ nano index.html && cat $_
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello!</title>
    </head>
    <body>
    <p>The page has loaded successfully!</p>
    </body>
</html>
```

```
~/appconfig$
```

- The index.html should be stored in the API and supplied to the pod's container when the pod is created.

Create the ConfigMap for `index.html`:

```
~/appconfig$ kubectl create configmap my-webpage --from-file index.html

configmap/my-webpage created

~/appconfig$
```

- Webpages for Apache are placed in the `/usr/local/apache2/htdocs/` directory

Prepare the pod spec that mounts the configMap at the specified directory:

```
~/appconfig$ nano my-websvr.yaml && cat $_
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-websvr
  name: my-websvr
spec:
  containers:
  - image: httpd:2.4
    name: my-websvr
    volumeMounts:
    - name: webpage
      mountPath: /usr/local/apache2/htdocs/
  volumes:
  - name: webpage
```

```
    configMap:
      name: my-webpage
```

```
~/appconfig$
```

Apply and test the page:

```
~/appconfig$ kubectl apply -f my-websvr.yaml

pod/my-websvr created

~/appconfig$ kubectl get pods my-websvr -o wide

NAME        READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
my-websvr   1/1     Running   0          4s    10.32.0.4   ip-172-31-6-204
<none>           <none>

~/appconfig$ curl 10.32.0.4

<!DOCTYPE html>
<html>
    <head>
        <title>Hello!</title>
    </head>
    <body>
    <p>The page has loaded successfully!</p>
    </body>
</html>

~/appconfig$
```