Exercise 8.1: Create a ConfigMap

Overview

Container files are ephemeral, which can be problematic for some applications. Should a container be restarted the files will be lost. In addition, we need a method to share files between containers inside a Pod.

A `Volume` is a directory accessible to containers in a Pod. Cloud providers offer volumes which persist further than the life of the Pod, such that AWS or GCE volumes could be pre-populated and offered to Pods, or transferred from one Pod to another. **Ceph** is also another popular solution for dynamic, persistent volumes.

Unlike current **Docker** volumes a Kubernetes volume has the lifetime of the Pod, not the containers within. You can also use different types of volumes in the same Pod simultaneously, but Volumes cannot mount in a nested fashion. Each must have their own mount point. Volumes are declared with `spec.volumes` and mount points with `spec.containers.volumeMounts` parameters. Each particular volume type, 24 currently, may have other restrictions. https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

We will also work with a `ConfigMap`, which is basically a set of key-value pairs. This data can be made available so that a Pod can read the data as environment variables or configuration data. A `ConfigMap` is similar to a `Secret`, except they are not base64 byte encoded arrays. They are stored as strings and can be read in serialized form.

There are three different ways a `ConfigMap` can ingest data, from a literal value, from a file or from a directory of files.

1. We will create a `ConfigMap` containing primary colors. We will create a series of files to ingest into the `ConfigMap`. First, we create a directory `primary` and populate it with four files. Then we create a file in our home directory with our favorite color.

```
student@cp:~$ mkdir primary

student@cp:~$ echo c > primary/cyan

student@cp:~$ echo m > primary/magenta

student@cp:~$ echo y > primary/yellow

student@cp:~$ echo k > primary/black

student@cp:~$ echo "known as key" >> primary/black

student@cp:~$ echo blue > favorite
```

2. Now we will create the `ConfigMap` and populate it with the files we created as well as a literal value from the command line.

```
student@cp:~$ kubectl create configmap colors \
     --from-literal=text=black  \
     --from-file=./favorite  \
     --from-file=./primary/
```

```
  configmap/colors created
```

3. View how the data is organized inside the cluster. Use the `yaml` then the `json` output type to see the formatting.

```
student@cp:~$ kubectl get configmap colors
```

```
NAME      DATA      AGE
colors    6         30s
```

```
student@cp:~$ kubectl get configmap colors -o yaml
```

```
apiVersion: v1
data:
  black: |
    k
    known as key
  cyan: |
    c
  favorite: |
    blue
  magenta: |
    m
  text: black
  yellow: |
    y
kind: ConfigMap
<output_omitted>
```

4. Now we can create a Pod to use the `ConfigMap`. In this case a particular parameter is being defined as an environment variable.

```
student@cp:~$ vim simpleshell.yaml
```

simpleshell.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: shell-demo
5  spec:
6    containers:
7    - name: nginx
8      image: nginx
9      env:
10     - name: ilike
11       valueFrom:
12         configMapKeyRef:
13           name: colors
14           key: favorite
```

5. Create the Pod and view the environmental variable. After you view the parameter, exit out and delete the pod.

```
student@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod/shell-demo created
```

```
student@cp:~$ kubectl exec shell-demo -- /bin/bash -c 'echo $ilike'
```

```
blue
```

```
student@cp:~$ kubectl delete pod shell-demo
```

```
pod "shell-demo" deleted
```

6. All variables from a file can be included as environment variables as well. Comment out the previous `env:` stanza and add a slightly different `envFrom` to the file. Having new and old code at the same time can be helpful to see and understand the differences. Recreate the Pod, check all variables and delete the pod again. They can be found spread throughout the environment variable output.

```
student@cp:~$ vim simpleshell.yaml
```

simpleshell.yaml

```
<output_omitted>
    image: nginx
#    env:
#    - name: ilike
#      valueFrom:
#        configMapKeyRef:
#          name: colors
#          key: favorite
    envFrom:                    #<-- Same indent as image: line
    - configMapRef:
        name: colors
```

```
student@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod/shell-demo created
```

```
student@cp:~$ kubectl exec shell-demo -- /bin/bash -c 'env'
```

```
black=k
known as key

KUBERNETES_SERVICE_PORT_HTTPS=443
cyan=c
<output_omitted>
```

```
student@cp:~$ kubectl delete pod shell-demo
```

```
pod "shell-demo" deleted
```

7. A `ConfigMap` can also be created from a YAML file. Create one with a few parameters to describe a car.

```
student@cp:~$ vim car-map.yaml
```

car-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fast-car
  namespace: default
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby
```

8. Create the `ConfigMap` and verify the settings.

```
student@cp:~$ kubectl create -f car-map.yaml
```

```
configmap/fast-car created
```

```
student@cp:~$ kubectl get configmap fast-car -o yaml
```

```yaml
apiVersion: v1
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby
kind: ConfigMap
<output_omitted>
```

9. We will now make the `ConfigMap` available to a Pod as a mounted volume. You can again comment out the previous environmental settings and add the following new stanza. The `containers:` and `volumes:` entries are indented the same number of spaces.

```
student@cp:~$ vim simpleshell.yaml
```

simpleshell.yaml

```yaml
<output_omitted>
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
      - name: car-vol
        mountPath: /etc/cars
  volumes:
    - name: car-vol
      configMap:
        name: fast-car
<comment out rest of file>
```

10. Create the Pod again. Verify the volume exists and the contents of a file within. Due to the lack of a carriage return in the file your next prompt may be on the same line as the output, `Shelby`.

```
student@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod "shell-demo" created
```

```
student@cp:~$  kubectl exec shell-demo -- /bin/bash -c 'df -ha |grep car'
```

```
/dev/root       9.6G  3.2G   6.4G  34% /etc/cars
```

```
student@cp:~$ kubectl exec shell-demo -- /bin/bash -c 'cat /etc/cars/car.trim'
```

```
Shelby  #<-- Then your prompt
```

11. Delete the `Pod` and `ConfigMaps` we were using.

```
student@cp:~$ kubectl delete pods shell-demo
```

```
pod "shell-demo" deleted
```

```
student@cp:~$ kubectl delete configmap fast-car colors
```

```
configmap "fast-car" deleted
configmap "colors" deleted
```