Managing Container Storage With Kubernetes Volumes

Your company, BeeBox, is developing some applications that have storage needs beyond the short-lived storage of the container file system. One component, a simple maintenance script, needs to be able to interact with a directory on the host file system. Another needs to be able to share data between two containers in the same Pod.

Your task is to build a simple architecture to demonstrate how these tasks can be accomplished with Kubernetes volumes. Create Pods that meet the specified criteria.

We are going to need to use a volume in order to output some data to the actual host disc. Then we will setup a multi-container pod that shares data between containers using a shared volume.

Create a Pod That Outputs Data to the Host Using a Volume

1. Create a maintenance pod.yml

vi maintenance-pod.yml

Figure 1-1

```
apiVersion: v1
kind: Pod
metadata:
    name: maintenance-pod
spec:
    containers:
    - name: busybox
      image: busybox
      command: ['sh', '-c', 'while true; do echo Success! >> /output/output.txt; sleep 5; done']
      volumeMounts:
      - name: output-vol
        mountPath: /output
    volumes:
     - name: output-vol
       hostPath:
        path: /var/data
~
```

2. Create the pod for the maint. Pod
3. Check the pods status

Now it should be running on the worker node

4. Check the /var/data file

cat /var/data/output.txt

This will show a few to many success! So it will grow larger and larger because it is writing to that file every 5 seconds. So it looks like we completed the first container.

Create a Multi-Container Pod That Shares Data Between Containers Using a Volume

So now we go back on the control plane node after verifying the container on the worker node

5. So now we create another yml file so we can share data between 2 containers

vi shared-data-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
    name: shared-data-pod
spec:
    containers:
    - name: busybox1
      image: busybox
      command: ['sh', '-c', 'while true; do echo Success! >> /output/output.txt; sleep 5; done']
      volumeMounts:
    - name: shared-vol
       mountPath: /output
    - name: busybox2
      image: busybox
      command: ['sh', '-c', 'while true; do cat /input/output.txt; sleep 5; done']
      volumeMounts:
      - name: shared-vol
        mountPath: /input
    volumes:
    - name: shared-vol
      emptyDir: {}
~
```

6. Once that is typed in save the file and then finish creating the multi-container Pod
7. Make sure the pods are up and running and check the status
8. To make sure the pod is working check the logs for the shared-data-pod.yml and check the status

kubectl logs shared-data-pod -c busybox2

There will be a lot of “Success!” again which means we have created both containers. One of which is using a host path volume to write some data to the host disk and the other of which is using an emptyDir volume to share a volume between two containers in the same Pod.