Exercise 5.2: Configure the Deployment: Attaching Storage

There are several types of storage which can be accessed with Kubernetes, with flexibility of storage being essential to scalability. In this exercise we will configure an NFS server. With the NFS server we will create a new **persistent volume (pv)** and a **persistent volume claim (pvc)** to use it.

1. Search for `pv` and `pvc` YAML example files on http://kubernetes.io/docs and http://kubernetes.io/blog.

2. Use the `CreateNFS.sh` script from the tarball to set up NFS on your cp node. This script will configure the server, export `/opt/sfw` and create a file `/opt/sfw/hello.txt`. Use the **find** command to locate the file if you don't remember where you extracted the tar file. This example narrows the search to your `$HOME` directory. Change for your environment. directory. You may find the same file in more than one sub-directory of the tarfile.

   ```
   student@cp:~$ find $HOME -name CreateNFS.sh
   ```

   ```
   <some_path>/CreateNFS.sh
   ```

   ```
   student@cp:~$ cp <path_from_output_above>/CreateNFS.sh $HOME
   ```

   ```
   student@cp:~$ bash $HOME/CreateNFS.sh
   ```

   ```
   Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial InRelease
   Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]

   <output_omitted>

   Should be ready. Test here and second node

   Export list for localhost:
   /opt/sfw *
   ```

3. Test by mounting the resource from your **second node**. Begin by installing the client software.

   ```
   student@worker:~$ sudo apt-get -y install nfs-common nfs-kernel-server
   ```

   ```
   <output_omitted>
   ```

4. Test you can see the exported directory using **showmount** from you second node.

   ```
   student@worker:~$ showmount -e cp #<-- Edit to be first node's name or IP
   ```

   ```
   Export list for cp:
   /opt/sfw *
   ```

5. Mount the directory. Be aware that unless you edit `/etc/fstab` this is not a persistent mount. Change out the node name for that of your cp node.

   ```
   student@worker:~$ sudo mount cp:/opt/sfw /mnt
   ```

6. Verify the `hello.txt` file created by the script can be viewed.

   ```
   student@worker:~$ ls -l /mnt
   ```

   ```
   total 4
   -rw-r--r-- 1 root root 9 Sep 28 17:55 hello.txt
   ```

7. Return to the cp node and create a YAML file for an object with kind **PersistentVolume**. The included example file needs an edit to the `server:` parameter. Use the hostname of the cp server and the directory you created in the previous step. Only syntax is checked, an incorrect name or directory will not generate an error, but a Pod using the incorrect resource will not start. Note that the `accessModes` do not currently affect actual access and are typically used as labels instead.

```
student@cp:~$ find $HOME -name PVol.yaml
```

```
<some_long_path>/PVol.yaml
```

```
student@cp:~$ cp <path_output_from_above>/PVol.yaml $HOME
```

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

PVol.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvvol-1
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /opt/sfw
    server: cp                        #<-- Edit to match cp node name or IP
    readOnly: false
```

8. Create and verify you have a new 1Gi volume named **pvvol-1**. Note the status shows as `Available`. Remember we made two persistent volumes for the image registry earlier.

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

```
persistentvolume/pvvol-1 created
```

```
student@cp:~$ kubectl get pv
```

```
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM    STORAGECLASS
↪   REASON   AGE
pvvol-1          1Gi        RWX            Retain           Available
↪   4s
registryvm       200Mi      RWO            Retain           Bound       default/nginx-claim0
↪   4d
task-pv-volume   200Mi      RWO            Retain           Bound       default/registry-claim0
↪   4d
```

9. Now that we have a new volume we will use a **persistent volume claim (pvc)** to use it in a Pod. We should have two existing claims from our local registry.

```
student@cp:~/$ kubectl get pvc
```

```
NAME              STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nginx-claim0      Bound    registryvm       200Mi      RWO                           4d
registry-claim0   Bound    task-pv-volume   200Mi      RWO                           4d
```

10. Create or copy a yaml file with the kind **PersistentVolumeClaim**.

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

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-one
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 200Mi
```

11. Create and verify the new pvc status is `bound`. Note the size is `1Gi`, even though `200Mi` was suggested. Only a volume of at least that size could be used, the first volume with found with at least that much space was chosen.

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

```
persistentvolumeclaim/pvc-one created
```

```
student@cp:~$ kubectl get pvc
```

```
NAME          STATUS    VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nginx-claim0      Bound     registryvm       200Mi      RWO                           4d
pvc-one           Bound     pvvol-1          1Gi        RWX                           4s
registry-claim0   Bound     task-pv-volume   200Mi      RWO                           4d
```

12. Now look at the status of the physical volume. It should also show as bound.

```
student@cp:~$ kubectl get pv
```

```
NAME           CAPACITY ACCESS MODES RECLAIM POLICY STATUS
 CLAIM      STORAGECLASS   REASON    AGE
pvvol-1        1Gi      RWX          Retain         Bound
 default/pvc-one                     14m
registryvm     200Mi    RWO          Retain         Bound
 default/nginx-claim0                4d
task-pv-volume 200Mi    RWO          Retain         Bound
 default/registry-claim0             4d
```

13. Edit the `simpleapp.yaml` file to include two new sections. One section for the container while will use the volume mount point, you should have an existing entry for `car-vol`. The other section adds a volume to the deployment in general, which you can put after the configMap volume section.

```
student@cp:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```yaml
....
        volumeMounts:
        - name: car-vol
          mountPath: /etc/cars
        - name: nfs-vol                   #<-- Add this and following line
          mountPath: /opt
....
```

```
     volumes:
      - name: car-vol
        configMap:
          defaultMode: 420
          name: fast-car
      - name: nfs-vol                     #<-- Add this and following two lines
        persistentVolumeClaim:
          claimName: pvc-one
status:
....
```

14. Delete and re-create the deployment.

```
student@cp:~$ kubectl delete deployment try1 ; kubectl create -f $HOME/app1/simpleapp.yaml
```

```
deployment.apps "try1" deleted
deployment.apps/try1 created
```

15. View the details any of the pods in the deployment, you should see `nfs-vol` mounted under `/opt`. The use to command line completion with the **tab** key can be helpful for using a pod name.

```
student@cp:~$ kubectl describe pod try1-594fbb5fc7-5k7sj
```

```
<output_omitted>
    Mounts:
      /etc/cars from car-vol (rw)
      /opt from nfs-vol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-j7cqd (ro)
<output_omitted>
```