Guided Exercise: Selecting a Storage Class for an Application

Deploy a MySQL database with persistent storage based on block storage by selecting block storage instead of default file storage.

Outcomes

- Create volumes from a storage class.
- Deploy applications with persistent storage.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command ensures that the cluster is accessible and that all resources are available for this exercise.

```
[student@workstation ~]$ lab start storage-classes
```

Instructions

1. In a terminal window, log in to the RHOCP cluster as the `developer` user with `developer` as the password, and switch to the `storage-classes` project.

   Log in to the OpenShift cluster.

   ```
   [student@workstation ~]$ oc login -u developer -p developer \
     https://api.ocp4.example.com:6443
   Login successful.
   ...output omitted...
   ```

   Set the `storage-classes` project as the active project.

   ```
   [student@workstation ~]$ oc project storage-classes
   ...output omitted...
   ```

2. Examine the available storage classes on the cluster. Identify an appropriate storage class to use for a database application.

   Use the `oc get` command to retrieve a list of storage classes in the cluster. You can use the `storageclass` short name, `sc`, in the command.

   ```
   [student@workstation ~]$ oc get sc
   NAME                   PROVISIONER   ...
   lvms-vg1               topolvm.io    ...
   nfs-storage (default)  k8s-sigs.io/nfs-subdir-external-provisioner ...
   ```

   Because an administrator can change the default storage class, applications must specify a storage class that meets the application requirements.

   Use the `oc describe` command to view the details of the `lvms-vg1` storage class.

   ```
   [student@workstation ~]$ oc describe sc lvms-vg1
   Name:                  lvms-vg1
   IsDefaultClass:        No
   Annotations:           description=Provides RWO and RWOP Filesystem & Block volumes
   Provisioner:           topolvm.io
   Parameters:            csi.storage.k8s.io/fstype=xfs,topolvm.io/device-class=vg1
   AllowVolumeExpansion:  True
   MountOptions:          <none>
   ReclaimPolicy:         Delete
   VolumeBindingMode:     WaitForFirstConsumer
   Events:                <none>
   ```

   The description annotation states that the storage class provides support for block volumes. For some applications, such as databases, block volumes can provide a performance advantage over file system volumes. In the `lvms-vg1` storage class, the `AllowVolumeExpansion` field is set to `True`. With volume expansion, cluster users can edit their PVC objects and specify a new size for the PVC. Kubernetes then uses the storage back end to automatically expand the volume to the requested size. Kubernetes also expands the file system of pods that use the PVC. Enabling volume expansion can help to protect an application from failing due to the data growing too fast. With these features, the `lvms-vg1` storage class is a good choice for the database application.

3. Use the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image to create a MySQL deployment named `db-pod`. Add the missing environment variables for the pod to run.

   Create the `db-pod` deployment.

```
[student@workstation ~]$ oc create deployment db-pod \
  --port 3306 \
  --image registry.ocp4.example.com:8443/rhel8/mysql-80
deployment.apps/db-pod created
```

Add the environment variables.

```
[student@workstation ~]$ oc set env deployment/db-pod \
  MYSQL_USER=user1 \
  MYSQL_PASSWORD=redhat123 \
  MYSQL_DATABASE=items
deployment.apps/db-pod updated
```

Verify that the pod is running.

```
[student@workstation ~]$ oc get pods
NAME                     READY   STATUS    RESTARTS   AGE
db-pod-b4ccfb74-nn4s5    1/1     Running   0          5s
```

Expose the `db-pod` deployment to create a service.

```
[student@workstation ~]$ oc expose deployment/db-pod
service/db-pod exposed
```

4. Add a 1 Gi, RWO PVC named `db-pod-pvc` to the deployment. Specify the volume name as `lvm-storage`, and set the `/var/lib/mysql` directory as the mount path. Use the `lvms-vg1` storage class.

Use the `oc set volume` command to create a PVC for the deployment.

```
[student@workstation ~]$ oc set volumes deployment/db-pod \
  --add --name lvm-storage --type pvc --claim-mode rwo \
  --claim-size 1Gi --mount-path /var/lib/mysql \
  --claim-class lvms-vg1 --claim-name db-pod-pvc
deployment.apps/db-pod volume updated
```

The `claim-class` option specifies a non-default storage class.

NOTE

The `oc set volumes` command does not have an option to set the volume mode property. Thus, the `oc set volumes` command results in using the default `Filesystem` volume mode. Use a declarative manifest to set the volume mode to `Block`. The use of declarative manifests is out of scope for this course.

Use the `oc get pvc` command to view the status of the PVC. Identify the name of the PV, and confirm that the PVC uses the `lvms-vg1` non-default storage class.

```
[student@workstation ~]$ oc get pvc
NAME       STATUS VOLUME    CAPACITY ACCESS MODES STORAGECLASS
db-pod-pvc Bound  pvc-72... 1Gi      RWO          lvms-vg1  ...
```

Use the `oc describe pvc` command to inspect the details of the `db-pod-pvc` PVC.

```
[student@workstation ~]$ oc describe pvc db-pod-pvc
Name:          db-pod-pvc
Namespace:     storage-classes
StorageClass:  lvms-vg1
Status:        Bound
Volume:        pvc-72084a46-3f32-435e-987b-4ad3b9026021
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: topolvm.io
               volume.kubernetes.io/selected-node: master01
               volume.kubernetes.io/storage-provisioner: topolvm.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Used By:       db-pod-568888457d-qmxn9
Events:
...output omitted...
```

The `Used By` attribute confirms that the PVC is bound to a pod.

5. Connect to the database to verify that it is working.

```
[student@workstation ~]$ oc run query-db -it --rm \
  --image registry.ocp4.example.com:8443/rhel8/mysql-80 \
  --restart Never --command \
  -- /bin/bash -c "mysql -uuser1 -predhat123 --protocol tcp \
  -h db-pod -P3306 items -e 'show databases;'"
mysql: [Warning] Using a password on the command line interface can be insecure.
+--------------------+
| Database           |
+--------------------+
| information_schema |
| items              |
| performance_schema |
+--------------------+
pod "query-db" deleted
```

6. Delete the `db-pod` deployment and the `db-pod-pvc` PVC.

   Delete the `db-pod` deployment and service.

```
[student@workstation ~]$ oc delete deployment db-pod
deployment.apps "db-pod" deleted
[student@workstation ~]$ oc delete service db-pod
service "db-pod" deleted
```

   Verify that the PVC still exists without the deployment.

```
[student@workstation ~]$ oc get pvc
NAME       STATUS VOLUME    CAPACITY ACCESS MODES STORAGECLASS
db-pod-pvc Bound  pvc-72... 1Gi      RWO          lvms-vg1  ...
```

   Delete the `db-pod-pvc` PVC.

```
[student@workstation ~]$ oc delete pvc db-pod-pvc
persistentvolumeclaim "db-pod-pvc" deleted
```

7. Create a PVC for an application that requires a shared storage volume from the `nfs-storage` storage class.

   Create a PVC YAML manifest file named `nfs-pvc.yaml` with the following contents:

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs-storage
```

Create the PVC by using the `oc create -f` command and the YAML manifest file.

```
[student@workstation ~]$ oc create -f nfs-pvc.yaml
persistentvolumeclaim/nfs-pvc created
```

Use the `oc describe pvc` command to view the details of the PVC resource.

```
[student@workstation ~]$ oc describe pvc nfs-pvc
Name:          nfs-pvc
Namespace:     storage-classes
StorageClass:  nfs-storage
Status:        Bound
Volume:        pvc-9f462124-d96e-43e5-96a7-f96dd9375579
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
               volume.kubernetes.io/storage-provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Used By:       <none>
Events:
...output omitted...
```

The `Used By: <none>` attribute shows that no pod is using the PVC. The `Status: Bound` value and the `Volume` attribute assignment confirm that the volume binding mode of the storage class is set to `Immediate`.

8. Use the `registry.ocp4.example.com:8443/ubi9/httpd-24:1-321` container image to create a web application deployment named `web-pod`.

   Create the `web-pod` deployment.

```
[student@workstation ~]$ oc create deployment web-pod \
  --port 8080 \
  --image registry.ocp4.example.com:8443/ubi9/httpd-24:1-321
deployment.apps/web-pod created
```

   Create a service for the `web-pod` application.

```
[student@workstation ~]$ oc expose deployment web-pod
service/web-pod exposed
```

   Expose the service to create a route for the `wep-pod` application. Specify `web-pod.apps.ocp4.example.com` as the hostname.

```
[student@workstation ~]$ oc expose svc web-pod \
  --hostname web-pod.apps.ocp4.example.com
route.route.openshift.io/web-pod exposed
```

   View the route that is assigned to the `web-pod` application.

```
[student@workstation ~]$ oc get routes
NAME     HOST/PORT                     PATH  SERVICES   PORT ...
web-pod  web-pod.apps.ocp4.example.com       web-pod    8080 ...
```

   Use the `curl` command to view the index page of the `web-pod` application.

```
[student@workstation ~]$ curl -s\
  http://web-pod.apps.ocp4.example.com/ | head
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        <head>
                <title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
...output omitted...
```

9. Add the `nfs-pvc` PVC to the `web-pod` deployment.

   Use the `oc set volume` command to add the PVC to the deployment. Specify the volume name as `nfs-volume`, and set the mount path to the `/var/www/html` directory.

   ```
   [student@workstation ~]$ oc set volumes deployment/web-pod \
     --add --name nfs-volume \
     --claim-name nfs-pvc \
     --mount-path /var/www/html
   deployment.apps/web-pod volume updated
   ```

   The volume `mount-path` is set to the `/var/www/html` directory. The server uses this path to serve HTML content.

10. Use the `registry.ocp4.example.com:8443/redhattraining/do180-roster` container image to create a custom web application deployment named `app-pod`.

    Create the `app-pod` deployment and specify port `9090` as the target port.

    ```
    [student@workstation ~]$ oc create deployment app-pod \
      --port 9090 --image \
      registry.ocp4.example.com:8443/redhattraining/do180-roster
    deployment.apps/app-pod created
    ```

    Create a service for the `app-pod` application.

    ```
    [student@workstation ~]$ oc expose deployment app-pod
    service/app-pod exposed
    ```

    Expose the service to create a route for the `app-pod` application. Use `app-pod.apps.ocp4.example.com` for the hostname.

    ```
    [student@workstation ~]$ oc expose svc app-pod \
      --hostname app-pod.apps.ocp4.example.com
    route.route.openshift.io/app-pod exposed
    ```

11. Add the `nfs-pvc` PVC to the `app-pod` application deployment.

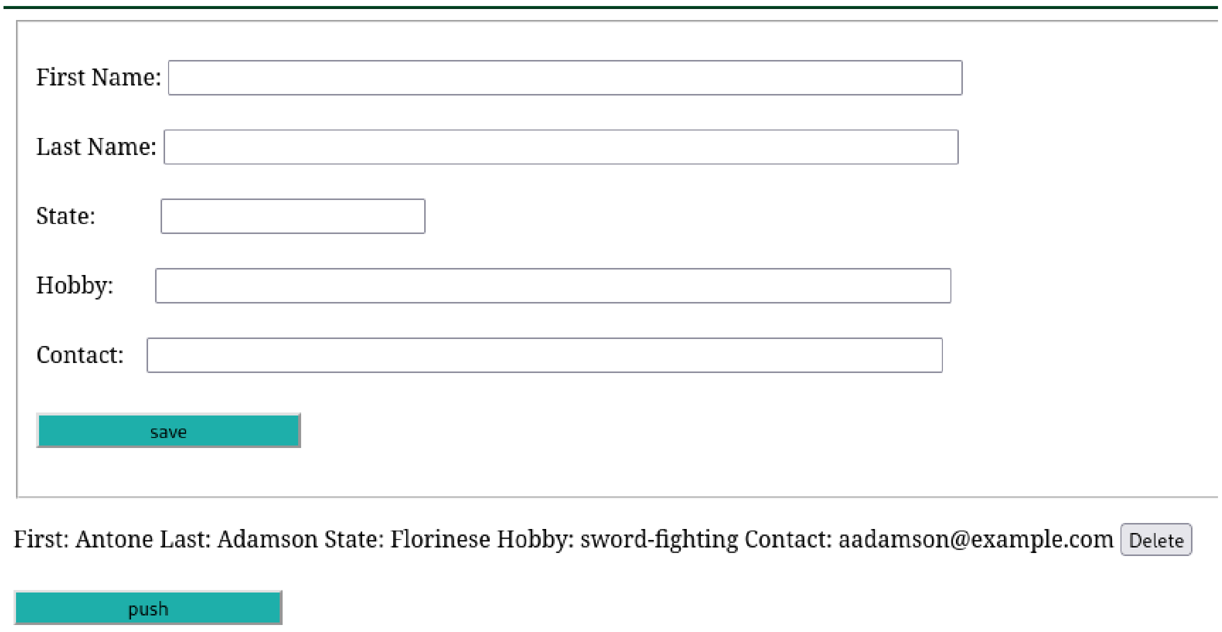
    Use the `oc set volume` command to add the PVC to the deployment. Set the volume name to `nfs-volume` and set the mount path to the `/var/tmp` directory.

    ```
    [student@workstation ~]$ oc set volumes deployment/app-pod \
      --add --name nfs-volume \
      --claim-name nfs-pvc \
      --mount-path /var/tmp
    deployment.apps/app-pod volume updated
    ```

    At this point, the `web-pod` and the `app-pod` applications are sharing a PVC. Kubernetes cannot prevent data conflicts between the two applications. In this case, the `app-pod` application is a writer and the `web-pod` application is a reader, and thus they do not conflict. The application implementation, not Kubernetes, prevents data corruption from the two applications that use the same PVC. The RWO access mode does not protect data integrity; a single node can mount the volume as read/write, and pods that share the volume must exist on the same node.

12. Use the `app-pod` application to add content to the shared volume.

    Open a web browser and go to `http://app-pod.apps.ocp4.example.com`



First: Antone Last: Adamson State: Florinese Hobby: sword-fighting Contact: aadamson@example.com

In the form, enter your information and click **save**. The application adds your information to the list after the form.

Click **push** to create the `/var/tmp/People.html` file on the shared volume.

13. Verify that the `web-pod` application displays the `People.html` file from the shared PVC. Open a browser tab and go to `http://web-pod.apps.ocp4.example.com/People.html`

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish storage-classes
```