Selecting a Storage Class for an Application

Objectives

- Match applications with storage classes that provide storage services to satisfy application requirements.

Storage Class Selection

Storage classes describe types of storage for the cluster and provision dynamic storage on demand. The cluster administrator determines the meaning of a storage class, which is also named a *profile* in other storage systems. For example, an administrator can create one storage class for development use and another for production use.

Kubernetes supports multiple storage back ends. The storage options differ in cost, performance, reliability, and function. An administrator can create different storage classes for these options. As a result, developers can select the storage solution that fits the needs of the application. Developers do not need to know the storage infrastructure details.

An administrator selects the default storage class for dynamic provisioning. A default storage class enables Kubernetes to automatically provision a PVC that does not specify a storage class. Because an administrator can change the default storage class, a developer should explicitly set the storage class for an application.

Reclaim Policy

Outside the application function, the developer must also consider the impact of the reclaim policy on storage requirements.

A reclaim policy determines what happens to the data on a PVC after the PVC is deleted. When you are finished with a volume, you can delete the PVC object from the API, which enables reclamation of the resource. Kubernetes releases the volume when the PVC is deleted, but the volume is not yet available for another claim. The previous claimant's data remains on the volume and must be handled according to the policy. To keep your data, choose a storage class with the `retain` reclaim policy.

By using the `retain` reclaim policy, when you delete a PVC, only the PVC object is deleted from the cluster. The PV that backed the PVC, the physical storage device that the PV used, and your data still exist. To reclaim the storage and to use it in your cluster again, the cluster administrator must take manual steps.

To manually reclaim a PV as a cluster administrator, follow these steps:

1. Delete the PV.

```
[user@host ~]$ oc delete pv pv-name
```

The associated asset in the external storage infrastructure, such as an Amazon Web Services (AWS) Elastic Block Store (EBS), Google Compute Engine (GCE) Persistent Disk, Azure Disk, or Cinder volume, still exists after the PV is deleted.

2. At this point, the cluster administrator can create another PV by using the same storage and data from the previous PV. A developer could then mount the new PV and access the data from the previous PV.

3. Alternatively, the cluster administrator can remove the data on the storage asset and then delete the storage asset.

To automatically delete the PV, the data, and the physical storage for a deleted PVC, you must choose a storage class that uses the `delete` reclaim policy. This reclaim policy automatically reclaims your storage volume when the PVC is deleted. The `delete` reclaim policy is the default setting for all storage provisioners that adhere to the Kubernetes Container Storage Interface (CSI) standards. If you use a storage class that does not specify a reclaim policy, then the `delete` reclaim policy is used.

For more information about the Kubernetes Container Storage Interface standards, refer to the *Kubernetes CSI Developer Documentation* website at https://kubernetes-csi.github.io/docs/

Kubernetes and Application Responsibilities

Kubernetes does not change how an application relates to storage. The application is responsible for working with its storage devices and for ensuring data integrity and consistency.

Kubernetes storage does not prevent an application from inadvisable actions, such as sharing a data volume between two databases that require exclusive access to data.

Because a PVC is a storage device that your Linux host mounts, an improperly configured application could behave unexpectedly. For example, you could have an iSCSI Logical Unit Number (LUN), which is expressed as a `ReadWriteOnce` (RWO) PVC that is not supposed to be shared, and then mount that same PVC on two pods of the same host. Whether this situation is problematic depends on the applications.

Usually, it is fine for two processes on the same host to share a disk. After all, many applications on your personal machine share a local disk. However, nothing prevents one text editor from overwriting and losing all edits from another text editor. The use of Kubernetes storage must come with the same caution.

Single-node access (RWO) and shared access (RWX) do not ensure that files can be shared safely and reliably. RWO means that only one cluster node can read and write to the PVC. Alternatively, with RWX, Kubernetes provides a storage volume that any pod can access for reading or writing.

Use Cases for Storage Classes

The administrator creates storage classes that serve the needs of the developers.
A `StorageClass` object defines each storage class, and the object contains information about the storage provider and the capabilities of the storage medium. The provider creates PVs to

match the specifications of the storage class. Administrators can create storage classes with various functional levels, based on many factors.

Storage volume modes
A storage class with `block` volume mode support can increase performance for applications that can use raw block devices. Consider using a storage class with `Filesystem` volume mode support for applications that share files or that provide file access.

Quality of Service (QoS) levels
A Solid State Drive (SSD) provides high speed and support for frequently accessed files. Use a lower cost and a slower hard drive (HDD) for files that are accessed less often.

Administrative support tier
A production-tier storage class can include volumes that are backed up often. In contrast, a development-tier storage class might include volumes that are not configured with a backup schedule.

Storage classes can use a combination of these factors and others to best fit the needs of the developers.

Kubernetes matches PVCs with the best available PV that is not bound to another PVC. The PV must provide the access mode that is specified in the PVC, and the volume must be at least as large as the requested size in the PVC. The supported access modes depend on the capabilities of the storage provider. A PVC can specify additional criteria, such as the name of a storage class. If a PVC cannot find a PV that matches all criteria, then the PVC enters a `pending` state and waits until an appropriate PV becomes available.

PVCs can request a specific storage class by specifying the `storageClassName` attribute. This method of selecting a specific storage class ensures that the storage medium is a good fit for the application requirements. Only PVs of the requested storage class can be bound to the PVC. The cluster administrator can configure dynamic provisioners to service storage classes. The cluster administrator can also create a PV on demand that matches the specifications in the PVC.

Create a Storage Class

The following YAML excerpt describes the definition for a `StorageClass` object. A cluster administrator or a `storage-admin` user creates globally scoped `StorageClass` objects. The following resource shows the parameters for configuring a storage class. This example uses the AWS EBS object definition.

```
apiVersion: storage.k8s.io/v1 (1)
kind: StorageClass (2)
metadata:
  name: io1-gold-storage (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'false'
    kubernetes.io/description: Provides RWO and RWOP volumes
...output omitted...
parameters: (5)
  type: io1
  iopsPerGB: "10"
...output omitted...
provisioner: ebs.csi.aws.com (6)
reclaimPolicy: Delete (7)
volumeBindingMode: WaitForFirstConsumer (8)
allowVolumeExpansion: true (9)
```

1. A required item that specifies the current API version.
2. A required item that specifies the API object type.
3. A required item that specifies the name of the storage class.
4. An optional item that specifies annotations for the storage class.
5. An optional item that specifies the required parameters for the specific provisioner; this object differs between plug-ins.
6. A required item that specifies the type of provisioner that is associated with this storage class.
7. An optional item that specifies the selected reclaim policy for the storage class.
8. An optional item that specifies the selected volume binding mode for the storage class.
9. An optional item that specifies the volume expansion setting.

Several attributes, such as the API version, API object type, and annotations, are common for Kubernetes objects, whereas other attributes are specific to storage class objects.

Parameters

Parameters can configure file types, change storage types, enable encryption, enable replication, and so on. Each provisioner has different parameter options. Accepted parameters depend on the storage provisioner. For example, the `io1` value for the `type` parameter, and the `iopsPerGB` parameter, are specific to EBS. When a parameter is omitted, the storage provisioner uses the default value.

Provisioners

The `provisioner` attribute identifies the source of the storage medium plug-in. Provisioners with names that begin with a `kubernetes.io` value are available by default in a Kubernetes cluster.

ReclaimPolicy

The default reclaim policy, `Delete`, automatically reclaims the storage volume when the PVC is deleted. Reclaiming storage in this way can reduce the storage costs. The `Retain` reclaim policy does not delete the storage volume, so that data is not lost if the wrong PVC is deleted. This reclaim policy can result in higher storage costs if space is not manually reclaimed.

VolumeBindingMode
The `volumeBindingMode` attribute determines how volume attachments are handled for a requesting PVC. Using the default `Immediate` volume binding mode creates a PV to match the PVC when the PVC is created. This setting does not wait for the pod to use the PVC, and thus can be inefficient. The `Immediate` binding mode can also cause problems for storage back ends that are topology-constrained or are not globally accessible from nodes in the cluster. PVs are also bound without the knowledge of a pod's scheduling requirements, which might result in unschedulable pods.

By using the `WaitForFirstConsumer` mode, the volume is created after the pod that uses the PVC is in use. With this mode, Kubernetes creates PVs that conform to the pod's scheduling constraints, such as resource requirements and selectors.

AllowVolumeExpansion
When set to the `true` value, the storage class specifies that the underlying storage volume can be expanded if more storage is required. Users can resize the volume by editing the corresponding PVC object. This feature can be used only to grow a volume, not to shrink it. The cluster administrator can use the `create` command to create a storage class from a YAML manifest file. The resulting storage class is not namespaced, and thus is available to all projects in the cluster.

```
[user@host ~]$ oc create -f storage-class-filename.yaml
```

To create a storage class from the web console, log in as an administrator, go to **Storage → StorageClasses**, and click **Create StorageClass**. Complete the form or the YAML manifest.

Cluster Storage Classes

Use the `oc get storageclass` command to view the available storage class options in a cluster.

```
[user@host ~]$ oc get storageclass
```

A regular cluster user can view the attributes of a storage class by using the `describe` command. The following example queries the attributes of the storage class with the `lvms-vg1` name.

```
[user@host ~]$ oc describe storageclass lvms-vg1
IsDefaultClass:        No
Annotations:           description=Provides RWO and RWOP 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 `describe` command can help a developer to decide whether the storage class is a good fit for an application. If none of the storage classes in the cluster are appropriate for the application, then the developer can request the cluster administrator to create a PV with the required features.

Storage Class Usage

Recall that the `oc set volume` command can add a PVC and an associated PV to a deployment. A YAML manifest file can declare the parameters of a PVC independently from the deployment. This method is the preferred option to support repeatability, configuration management, and version control. Use the `storageClassName` attribute to specify the storage class for the PVC.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-block-pvc
spec:
  accessModes:
    - RWO
  volumeMode: Block
  storageClassName: storage-class-name
  resources:
    requests:
      storage: 10Gi
```

Use the `create` command to create the resource from the YAML manifest file.

```
[user@host ~]$ oc create -f my-pvc-filename.yaml
```

Use the `--claim-name` option with the `set volume` command to add the existing PVC to a deployment.

```
[user@host ~]$ oc set volume deployment/deployment-name \
  --add --name my-volume-name \
  --claim-name my-block-pvc \
  --mount-path /var/tmp
```

REFERENCES

For more information, refer to the *Understanding Persistent Storage* section in the Red Hat OpenShift Container Platform 4.18 *Storage* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/storage/index#understanding-persistent-storage

Kubernetes Storage

For more information about the Kubernetes Container Storage Interface standards, refer to the *Kubernetes CSI Developer Documentation* website at https://kubernetes-csi.github.io/docs/