OpenShift Dedicated 4

Storage

Configuring and managing storage in OpenShift Dedicated 4

Last Updated: 2020-06-18

OpenShift Dedicated 4 Storage

Configuring and managing storage in OpenShift Dedicated 4

Legal Notice

Abstract

This document provides instructions for configuring persistent volumes from various storage back ends and managing dynamic allocation from Pods.

Table of Contents

CHAPTER 1. UNDERSTANDING PERSISTENT STORAGE

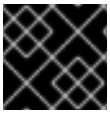
1.1. PERSISTENT STORAGE OVERVIEW

Managing storage is a distinct problem from managing compute resources. OpenShift Dedicated uses the Kubernetes persistent volume (PV) framework to allow cluster administrators to provision persistent storage for a cluster. Developers can use persistent volume claims (PVCs) to request PV resources without having specific knowledge of the underlying storage infrastructure.

PVCs are specific to a project, and are created and used by developers as a means to use a PV. PV resources on their own are not scoped to any single project; they can be shared across the entire OpenShift Dedicated cluster and claimed from any project. After a PV is bound to a PVC, that PV can not then be bound to additional PVCs. This has the effect of scoping a bound PV to a single namespace, that of the binding project.

PVs are defined by a **PersistentVolume** API object, which represents a piece of existing storage in the cluster that was either statically provisioned by the cluster administrator or dynamically provisioned using a StorageClass object. It is a resource in the cluster just like a node is a cluster resource.

PVs are volume plug-ins like **Volumes** but have a lifecycle that is independent of any individual Pod that uses the PV. PV objects capture the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.



IMPORTANT

High availability of storage in the infrastructure is left to the underlying storage provider.

PVCs are defined by a **PersistentVolumeClaim** API object, which represents a request for storage by a developer. It is similar to a Pod in that Pods consume node resources and PVCs consume PV resources. For example, Pods can request specific levels of resources, such as CPU and memory, while PVCs can request specific storage capacity and access modes. For example, they can be mounted once read-write or many times read-only.

1.2. LIFECYCLE OF A VOLUME AND CLAIM

PVs are resources in the cluster. PVCs are requests for those resources and also act as claim checks to the resource. The interaction between PVs and PVCs have the following lifecycle.

1.2.1. Provision storage

In response to requests from a developer defined in a PVC, a cluster administrator configures one or more dynamic provisioners that provision storage and a matching PV.

1.2.2. Bind claims

When you create a PVC, you request a specific amount of storage, specify the required access mode, and create a storage class to describe and classify the storage. The control loop in the master watches for new PVCs and binds the new PVC to an appropriate PV. If an appropriate PV does not exist, a provisioner for the storage class creates one.

The size of all PVs might exceed your PVC size. This is especially true with manually provisioned PVs. To minimize the excess, OpenShift Dedicated binds to the smallest PV that matches all other criteria.

Claims remain unbound indefinitely if a matching volume does not exist or can not be created with any available provisioner servicing a storage class. Claims are bound as matching volumes become available. For example, a cluster with many manually provisioned 50Gi volumes would not match a PVC requesting 100Gi. The PVC can be bound when a 100Gi PV is added to the cluster.

1.2.3. Use Pods and claimed PVs

Pods use claims as volumes. The cluster inspects the claim to find the bound volume and mounts that volume for a Pod. For those volumes that support multiple access modes, you must specify which mode applies when you use the claim as a volume in a Pod.

Once you have a claim and that claim is bound, the bound PV belongs to you for as long as you need it. You can schedule Pods and access claimed PVs by including **persistentVolumeClaim** in the Pod's volumes block.

1.2.4. Release volumes

When you are finished with a volume, you can delete the PVC object from the API, which allows reclamation of the resource. The volume is considered released when the claim is deleted, but it is not yet available for another claim. The previous claimant's data remains on the volume and must be handled according to policy.

1.2.5. Reclaim volumes

The reclaim policy of a **PersistentVolume** tells the cluster what to do with the volume after it is released. Volumes reclaim policy can either be **Retain**, **Recycle**, or **Delete**.

- **Retain** reclaim policy allows manual reclamation of the resource for those volume plug-ins that support it.
- **Recycle** reclaim policy recycles the volume back into the pool of unbound persistent volumes once it is released from its claim.

IMPORTANT

The **Recycle** reclaim policy is deprecated in OpenShift Dedicated 4. Dynamic provisioning is recommended for equivalent and better functionality.

- **Delete** reclaim policy deletes both the **PersistentVolume** object from OpenShift Dedicated and the associated storage asset in external infrastructure, such as AWS EBS or VMware vSphere.

NOTE

Dynamically provisioned volumes are always deleted.

1.3. PERSISTENT VOLUMES

Each PV contains a **spec** and **status**, which is the specification and status of the volume, for example:

PV object definition example

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 5Gi (2)
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain (4)
  ...
status:
  ...
```

1. Name of the persistent volume.
2. The amount of storage available to the volume.
3. The access mode, defining the read-write and mount permissions.
4. The reclaim policy, indicating how the resource should be handled once it is released.

1.3.1. Types of PVs

OpenShift Dedicated supports the following **PersistentVolume** plug-ins:

- AWS Elastic Block Store (EBS)
- GCE Persistent Disk

1.3.2. Capacity

Generally, a PV has a specific storage capacity. This is set by using the PV's **capacity** attribute.

Currently, storage capacity is the only resource that can be set or requested. Future attributes may include IOPS, throughput, and so on.

1.3.3. Access modes

A **PersistentVolume** can be mounted on a host in any way supported by the resource provider. Providers have different capabilities and each PV's access modes are set to the specific modes supported by that particular volume. For example, NFS can support multiple read-write clients, but a specific NFS PV might be exported on the server as read-only. Each PV gets its own set of access modes describing that specific PV's capabilities.

Claims are matched to volumes with similar access modes. The only two matching criteria are access modes and size. A claim's access modes represent a request. Therefore, you might be granted more, but never less. For example, if a claim requests RWO, but the only volume available is an NFS PV (RWO+ROX+RWX), the claim would then match NFS because it supports RWO.

Direct matches are always attempted first. The volume's modes must match or contain more modes than you requested. The size must be greater than or equal to what is expected. If two types of volumes, such as NFS and iSCSI, have the same set of access modes, either of them can match a claim with those modes. There is no ordering between types of volumes and no way to choose one type over another.

All volumes with the same modes are grouped, and then sorted by size, smallest to largest. The binder gets the group with matching modes and iterates over each, in size order, until one size matches.

The following table lists the access modes:

Table 1.1. Access modes

Access Mode	CLI abbreviation	Description
ReadWriteOnce	**RWO**	The volume can be mounted as read-write by a single node.

IMPORTANT

A volume's **AccessModes** are descriptors of the volume's capabilities. They are not enforced constraints. The storage provider is responsible for runtime errors resulting from invalid use of the resource.

For example, NFS offers **ReadWriteOnce** access mode. You must mark the claims as **read-only** if you want to use the volume's ROX capability. Errors in the provider show up at runtime as mount errors.

iSCSI and Fibre Channel volumes do not currently have any fencing mechanisms. You must ensure the volumes are only used by one node at a time. In certain situations, such as draining a node, the volumes can be used simultaneously by two nodes. Before draining the node, first ensure the Pods that use these volumes are deleted.

Table 1.2. Supported access modes for PVs

Volume Plug-in	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWS EBS	▮	-	-

NOTE

Use a recreate deployment strategy for Pods that rely on AWS EBS.

1.3.4. Restrictions

The following restrictions apply when using PVs with OpenShift Dedicated:

- PVs are provisioned with either EBS volumes (AWS) or GCP storage (GCP), depending on where the cluster is provisioned.
- Only RWO access mode is applicable, as EBS volumes and GCE Persistent Disks can not be mounted to multiple nodes.
- **emptyDir** has the same lifecycle as the Pod:
  - **emptyDir** volumes survive container crashes/restarts.
  - **emptyDir** volumes are deleted when the Pod is deleted.

1.3.5. Phase

Volumes can be found in one of the following phases:

Table 1.3. Volume phases

Phase	Description
Available	A free resource not yet bound to a claim.
Bound	The volume is bound to a claim.
Released	The claim was deleted, but the resource is not yet reclaimed by the cluster.
Failed	The volume has failed its automatic reclamation.

You can view the name of the PVC bound to the PV by running:

```
$ oc get pv <pv-claim>
```

1.4. PERSISTENT VOLUME CLAIMS

Each persistent volume claim (PVC) contains a **spec** and **status**, which is the specification and status of the claim, for example:

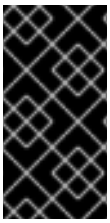
PVC object definition example

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim 1
spec:
  accessModes:
    - ReadWriteOnce 2
  resources:
    requests:
      storage: 8Gi 3
  storageClassName: gold 4
status:
  ...
```

1. Name of the PVC
2. The access mode, defining the read-write and mount permissions
3. The amount of storage available to the PVC
4. Name of the **StorageClass** required by the claim

1.4.1. Storage classes

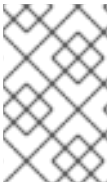
Claims can optionally request a specific storage class by specifying the storage class's name in the **storageClassName** attribute. Only PVs of the requested class, ones with the same **storageClassName** as the PVC, can be bound to the PVC. The cluster administrator can configure dynamic provisioners to service one or more storage classes. The cluster administrator can create a PV on demand that matches the specifications in the PVC.



IMPORTANT

The ClusterStorageOperator may install a default StorageClass depending on the platform in use. This StorageClass is owned and controlled by the operator. It cannot be deleted or modified beyond defining annotations and labels. If different behavior is desired, you must define a custom StorageClass.

The cluster administrator can also set a default storage class for all PVCs. When a default storage class is configured, the PVC must explicitly ask for **StorageClass** or **storageClassName** annotations set to **""** to be bound to a PV without a storage class.



NOTE

If more than one StorageClass is marked as default, a PVC can only be created if the **storageClassName** is explicitly specified. Therefore, only one StorageClass should be set as the default.

1.4.2. Access modes

Claims use the same conventions as volumes when requesting storage with specific access modes.

1.4.3. Resources

Claims, such as Pods, can request specific quantities of a resource. In this case, the request is for storage. The same resource model applies to volumes and claims.

1.4.4. Claims as volumes

Pods access storage by using the claim as a volume. Claims must exist in the same namespace as the Pod by using the claim. The cluster finds the claim in the Pod's namespace and uses it to get the **PersistentVolume** backing the claim. The volume is mounted to the host and into the Pod, for example:

Mount volume to the host and into the Pod example

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html" 1
        name: mypd 2
```

```
volumes:
  - name: mypd
    persistentVolumeClaim:
      claimName: myclaim 3
```

1 Path to mount the volume inside the Pod

2 Name of the volume to mount

3 Name of the PVC, that exists in the same namespace, to use

CHAPTER 2. EXPANDING PERSISTENT VOLUMES

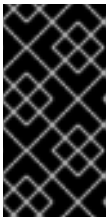
2.1. EXPANDING OPENSHIFT DEDICATED PERSISTENT VOLUME CLAIMS (PVCS)

Expanding PVCs based on volume types that need file system re-sizing, such as AWS EBS, is a two-step process. This process involves expanding volume objects in the cloud provider and then expanding the file system on the actual node. These steps occur automatically after the PVC object is edited and might require a Pod restart to take effect.

Expanding the file system on the node only happens when a new Pod is started with the volume.

Prerequisites

- The controlling StorageClass must have **allowVolumeExpansion** set to **true**. This is the default configuration in OpenShift Dedicated.



IMPORTANT

Decreasing the size of an Amazon Elastic Block Store (EBS) volume is not supported. However, you can create a smaller volume and then migrate your data to it by using a tool such as **oc rsync**. After modifying a volume, you must wait at least six hours before making additional modifications to the same volume.

Procedure

1. Edit the PVC and request a new size by editing the **spec.resources.requests.storage** value. The following **oc patch** command will change the PVC's size:

```
$ oc patch pvc <pvc_name> -p '{"spec": {"resources": {"requests": {"storage": "8Gi"}}}}'
```

2. Once the cloud provider object has finished re-sizing, the PVC might be set to **FileSystemResizePending**. The following command is used to check the condition:

```
$ oc describe pvc mysql

Name:          mysql
Namespace:     my-project
StorageClass:  gp2
Status:        Bound
Volume:        pvc-5fa3feb4-7115-4735-8652-8ebcfec91bb9
Labels:        app=cakephp-mysql-persistent
               template=cakephp-mysql-persistent
               template.openshift.io/template-instance-owner=6c7f7c56-1037-4105-8c08-
55a6261c39ca
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/aws-ebs
               volume.kubernetes.io/selected-node: ip-10-0-128-221.us-east-2.compute.internal
               volume.kubernetes.io/storage-resizer: kubernetes.io/aws-ebs
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi (1)
Access Modes:  RWO
```

```
VolumeMode:    Filesystem
Conditions: (2)
  Type                      Status  LastProbeTime     LastTransitionTime  Reason  Message
  ----                      ------  -----------------  ------------------  ------  -------
  FileSystemResizePending   True    <Timestamp>       <Timestamp>                 Waiting for
user to (re-)start a Pod to
                                                                                  finish file system resize of volume on
node.
Events:
  Type      Reason                Age   From                         Message
  ----      ------                ----  ----                         -------
  Normal    WaitForFirstConsumer  36m   persistentvolume-controller  waiting for first
consumer to be created before binding
  Normal    ProvisioningSucceeded 36m   persistentvolume-controller  Successfully
provisioned volume
                                                                     pvc-5fa3feb4-7115-4735-8652-8ebcfec91bb9
using
                                                                     kubernetes.io/aws-ebs
Mounted By:  mysql-1-q4nz7 (3)
```

1. The current capacity of the PVC.
2. Any relevant conditions are displayed here.
3. The Pod that is currently mounting this volume

3. If the output of the previous command included a message to restart the Pod, delete the mounting Pod that it specified:

```
$ oc delete pod mysql-1-q4nz7
```

4. Once the Pod is running, the newly requested size is available and the **FileSystemResizePending** condition is removed from the PVC.