OpenShift Container Platform 4.14

Storage

Configuring and managing storage in OpenShift Container Platform

Last Updated: 2023-11-15

OpenShift Container Platform 4.14 Storage

Configuring and managing storage in OpenShift Container Platform

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. OPENSHIFT CONTAINER PLATFORM STORAGE OVERVIEW

OpenShift Container Platform supports multiple types of storage, both for on-premise and cloud providers. You can manage container storage for persistent and non-persistent data in an OpenShift Container Platform cluster.

1.1. GLOSSARY OF COMMON TERMS FOR OPENSHIFT CONTAINER PLATFORM STORAGE

This glossary defines common terms that are used in the storage content.

Access modes

Volume access modes describe volume capabilities. You can use access modes to match persistent volume claim (PVC) and persistent volume (PV). The following are the examples of access modes:

- ReadWriteOnce (RWO)
- ReadOnlyMany (ROX)
- ReadWriteMany (RWX)
- ReadWriteOncePod (RWOP)

Cinder

The Block Storage service for Red Hat OpenStack Platform (RHOSP) which manages the administration, security, and scheduling of all volumes.

Config map

A config map provides a way to inject configuration data into pods. You can reference the data stored in a config map in a volume of type **ConfigMap**. Applications running in a pod can use this data.

Container Storage Interface (CSI)

An API specification for the management of container storage across different container orchestration (CO) systems.

Dynamic Provisioning

The framework allows you to create storage volumes on-demand, eliminating the need for cluster administrators to pre-provision persistent storage.

Ephemeral storage

Pods and containers can require temporary or transient local storage for their operation. The lifetime of this ephemeral storage does not extend beyond the life of the individual pod, and this ephemeral storage cannot be shared across pods.

Fiber channel

A networking technology that is used to transfer data among data centers, computer servers, switches and storage.

FlexVolume

FlexVolume is an out-of-tree plugin interface that uses an exec-based model to interface with storage drivers. You must install the FlexVolume driver binaries in a pre-defined volume plugin path on each node and in some cases the control plane nodes.

fsGroup

The fsGroup defines a file system group ID of a pod.

iSCSI

Internet Small Computer Systems Interface (iSCSI) is an Internet Protocol-based storage networking standard for linking data storage facilities. An iSCSI volume allows an existing iSCSI (SCSI over IP) volume to be mounted into your Pod.

hostPath

A hostPath volume in an OpenShift Container Platform cluster mounts a file or directory from the host node's filesystem into your pod.

KMS key

The Key Management Service (KMS) helps you achieve the required level of encryption of your data across different services. you can use the KMS key to encrypt, decrypt, and re-encrypt data.

Local volumes

A local volume represents a mounted local storage device such as a disk, partition or directory.

NFS

A Network File System (NFS) that allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables system administrators to consolidate resources onto centralized servers on the network.

OpenShift Data Foundation

A provider of agnostic persistent storage for OpenShift Container Platform supporting file, block, and object storage, either in-house or in hybrid clouds

Persistent storage

Pods and containers can require permanent storage for their operation. OpenShift Container Platform uses the Kubernetes persistent volume (PV) framework to allow cluster administrators to provision persistent storage for a cluster. Developers can use PVC to request PV resources without having specific knowledge of the underlying storage infrastructure.

Persistent volumes (PV)

OpenShift Container Platform uses the Kubernetes persistent volume (PV) framework to allow cluster administrators to provision persistent storage for a cluster. Developers can use PVC to request PV resources without having specific knowledge of the underlying storage infrastructure.

Persistent volume claims (PVCs)

You can use a PVC to mount a PersistentVolume into a Pod. You can access the storage without knowing the details of the cloud environment.

Pod

One or more containers with shared resources, such as volume and IP addresses, running in your OpenShift Container Platform cluster. A pod is the smallest compute unit defined, deployed, and managed.

Reclaim policy

A policy that tells the cluster what to do with the volume after it is released. A volume's reclaim policy can be **Retain**, **Recycle**, or **Delete**.

Role-based access control (RBAC)

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

Stateless applications

A stateless application is an application program that does not save client data generated in one session for use in the next session with that client.

Stateful applications

A stateful application is an application program that saves data to persistent disk storage. A server, client, and applications can use a persistent disk storage. You can use the **Statefulset** object in OpenShift Container Platform to manage the deployment and scaling of a set of Pods, and provides guarantee about the ordering and uniqueness of these Pods.

Static provisioning

A cluster administrator creates a number of PVs. PVs contain the details of storage. PVs exist in the Kubernetes API and are available for consumption.

Storage

OpenShift Container Platform supports many types of storage, both for on-premise and cloud providers. You can manage container storage for persistent and non-persistent data in an OpenShift Container Platform cluster.

Storage class

A storage class provides a way for administrators to describe the classes of storage they offer. Different classes might map to quality of service levels, backup policies, arbitrary policies determined by the cluster administrators.

VMware vSphere’s Virtual Machine Disk (VMDK) volumes

Virtual Machine Disk (VMDK) is a file format that describes containers for virtual hard disk drives that is used in virtual machines.

1.2. STORAGE TYPES

OpenShift Container Platform storage is broadly classified into two categories, namely ephemeral storage and persistent storage.

1.2.1. Ephemeral storage

Pods and containers are ephemeral or transient in nature and designed for stateless applications. Ephemeral storage allows administrators and developers to better manage the local storage for some of their operations. For more information about ephemeral storage overview, types, and management, see Understanding ephemeral storage .

1.2.2. Persistent storage

Stateful applications deployed in containers require persistent storage. OpenShift Container Platform uses a pre-provisioned storage framework called persistent volumes (PV) to allow cluster administrators to provision persistent storage. The data inside these volumes can exist beyond the lifecycle of an individual pod. Developers can use persistent volume claims (PVCs) to request storage requirements. For more information about persistent storage overview, configuration, and lifecycle, see Understanding persistent storage.

1.3. CONTAINER STORAGE INTERFACE (CSI)

CSI is an API specification for the management of container storage across different container orchestration (CO) systems. You can manage the storage volumes within the container native environments, without having specific knowledge of the underlying storage infrastructure. With the CSI, storage works uniformly across different container orchestration systems, regardless of the storage vendors you are using. For more information about CSI, see Using Container Storage Interface (CSI) .

1.4. DYNAMIC PROVISIONING

Dynamic Provisioning allows you to create storage volumes on-demand, eliminating the need for cluster administrators to pre-provision storage. For more information about dynamic provisioning, see Dynamic provisioning.

CHAPTER 2. UNDERSTANDING EPHEMERAL STORAGE

2.1. OVERVIEW

In addition to persistent storage, pods and containers can require ephemeral or transient local storage for their operation. The lifetime of this ephemeral storage does not extend beyond the life of the individual pod, and this ephemeral storage cannot be shared across pods.

Pods use ephemeral local storage for scratch space, caching, and logs. Issues related to the lack of local storage accounting and isolation include the following:

- Pods cannot detect how much local storage is available to them.
- Pods cannot request guaranteed local storage.
- Local storage is a best-effort resource.
- Pods can be evicted due to other pods filling the local storage, after which new pods are not admitted until sufficient storage is reclaimed.

Unlike persistent volumes, ephemeral storage is unstructured and the space is shared between all pods running on a node, in addition to other uses by the system, the container runtime, and OpenShift Container Platform. The ephemeral storage framework allows pods to specify their transient local storage needs. It also allows OpenShift Container Platform to schedule pods where appropriate, and to protect the node against excessive use of local storage.

While the ephemeral storage framework allows administrators and developers to better manage local storage, I/O throughput and latency are not directly effected.

2.2. TYPES OF EPHEMERAL STORAGE

Ephemeral local storage is always made available in the primary partition. There are two basic ways of creating the primary partition: root and runtime.

Root

This partition holds the kubelet root directory, **/var/lib/kubelet/** by default, and **/var/log/** directory. This partition can be shared between user pods, the OS, and Kubernetes system daemons. This partition can be consumed by pods through **EmptyDir** volumes, container logs, image layers, and container-writable layers. Kubelet manages shared access and isolation of this partition. This partition is ephemeral, and applications cannot expect any performance SLAs, such as disk IOPS, from this partition.

Runtime

This is an optional partition that runtimes can use for overlay file systems. OpenShift Container Platform attempts to identify and provide shared access along with isolation to this partition. Container image layers and writable layers are stored here. If the runtime partition exists, the **root** partition does not hold any image layer or other writable storage.

2.3. EPHEMERAL STORAGE MANAGEMENT

Cluster administrators can manage ephemeral storage within a project by setting quotas that define the limit ranges and number of requests for ephemeral storage across all pods in a non-terminal state. Developers can also set requests and limits on this compute resource at the pod and container level.

You can manage local ephemeral storage by specifying requests and limits. Each container in a pod can specify the following:

- **spec.containers[].resources.limits.ephemeral-storage**
- **spec.containers[].resources.requests.ephemeral-storage**

Limits and requests for ephemeral storage are measured in byte quantities. You can express storage as a plain integer or as a fixed-point number using one of these suffixes: E, P, T, G, M, k. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. For example, the following quantities all represent approximately the same value: 128974848, 129e6, 129M, and 123Mi. The case of the suffixes is significant. If you specify 400m of ephemeral storage, this requests 0.4 bytes, rather than 400 mebibytes (400Mi) or 400 megabytes (400M), which was probably what was intended.

The following example shows a pod with two containers. Each container requests 2GiB of local ephemeral storage. Each container has a limit of 4GiB of local ephemeral storage. Therefore, the pod has a request of 4GiB of local ephemeral storage, and a limit of 8GiB of local ephemeral storage.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        ephemeral-storage: "2Gi" ①
      limits:
        ephemeral-storage: "4Gi" ②
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        ephemeral-storage: "2Gi" ③
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

① ③ Request for local ephemeral storage.

② Limit for local ephemeral storage.

This setting in the pod spec affects how the scheduler makes a decision on scheduling pods, and also how kubelet evict pods. First of all, the scheduler ensures that the sum of the resource requests of the scheduled containers is less than the capacity of the node. In this case, the pod can be assigned to a node only if its available ephemeral storage (allocatable resource) is more than 4GiB.

Secondly, at the container level, since the first container sets resource limit, kubelet eviction manager measures the disk usage of this container and evicts the pod if the storage usage of this container exceeds its limit (4GiB). At the pod level, kubelet works out an overall pod storage limit by adding up the limits of all the containers in that pod. In this case, the total storage usage at the pod level is the sum of the disk usage from all containers plus the pod's **emptyDir** volumes. If this total usage exceeds the overall pod storage limit (4GiB), then the kubelet also marks the pod for eviction.

For information about defining quotas for projects, see Quota setting per project.

2.4. MONITORING EPHEMERAL STORAGE

You can use **/bin/df** as a tool to monitor ephemeral storage usage on the volume where ephemeral container data is located, which is **/var/lib/kubelet** and **/var/lib/containers**. The available space for only **/var/lib/kubelet** is shown when you use the **df** command if **/var/lib/containers** is placed on a separate disk by the cluster administrator.

To show the human-readable values of used and available space in **/var/lib**, enter the following command:

```
$ df -h /var/lib
```

The output shows the ephemeral storage usage in **/var/lib**:

Example output

```
Filesystem  Size  Used Avail Use% Mounted on
/dev/disk/by-partuuid/4cd1448a-01    69G   32G   34G  49% /
```

CHAPTER 3. UNDERSTANDING PERSISTENT STORAGE

3.1. PERSISTENT STORAGE OVERVIEW

Managing storage is a distinct problem from managing compute resources. OpenShift Container Platform 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 Container Platform 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 plugins 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.

3.2. 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.

3.2.1. 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.

Alternatively, a cluster administrator can create a number of PVs in advance that carry the details of the real storage that is available for use. PVs exist in the API and are available for use.

3.2.2. 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 Container Platform 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.

3.2.3. 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.

NOTE

If you attach persistent volumes that have high file counts to pods, those pods can fail or can take a long time to start. For more information, see When using Persistent Volumes with high file counts in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state?.

3.2.4. Storage Object in Use Protection

The Storage Object in Use Protection feature ensures that PVCs in active use by a pod and PVs that are bound to PVCs are not removed from the system, as this can result in data loss.

Storage Object in Use Protection is enabled by default.

NOTE

A PVC is in active use by a pod when a **Pod** object exists that uses the PVC.

If a user deletes a PVC that is in active use by a pod, the PVC is not removed immediately. PVC removal is postponed until the PVC is no longer actively used by any pods. Also, if a cluster admin deletes a PV that is bound to a PVC, the PV is not removed immediately. PV removal is postponed until the PV is no longer bound to a PVC.

3.2.5. Release a persistent volume

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.

3.2.6. Reclaim policy for persistent volumes

The reclaim policy of a persistent volume tells the cluster what to do with the volume after it is released. A volume's reclaim policy can be **Retain**, **Recycle**, or **Delete**.

- **Retain** reclaim policy allows manual reclamation of the resource for those volume plugins 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 Container Platform 4. Dynamic provisioning is recommended for equivalent and better functionality.

- **Delete** reclaim policy deletes both the **PersistentVolume** object from OpenShift Container Platform and the associated storage asset in external infrastructure, such as Amazon Elastic Block Store (Amazon EBS) or VMware vSphere.

NOTE

Dynamically provisioned volumes are always deleted.

3.2.7. Reclaiming a persistent volume manually

When a persistent volume claim (PVC) is deleted, the persistent volume (PV) still exists and is considered "released". However, the PV is not yet available for another claim because the data of the previous claimant remains on the volume.

Procedure

To manually reclaim the PV as a cluster administrator:

1. Delete the PV.

   ```
   $ oc delete pv <pv-name>
   ```

   The associated storage asset in the external infrastructure, such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume, still exists after the PV is deleted.

2. Clean up the data on the associated storage asset.

3. Delete the associated storage asset. Alternately, to reuse the same storage asset, create a new PV with the storage asset definition.

The reclaimed PV is now available for use by another PVC.

3.2.8. Changing the reclaim policy of a persistent volume

To change the reclaim policy of a persistent volume:

1. List the persistent volumes in your cluster:

   ```
   $ oc get pv
   ```

   #### Example output

   ```
   NAME                      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
   ```

```
CLAIM           STORAGECLASS    REASON   AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Delete         Bound
default/claim1     manual                   10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Delete         Bound
default/claim2     manual                   6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Delete         Bound
default/claim3     manual                   3s
```

2. Choose one of your persistent volumes and change its reclaim policy:

```
$ oc patch pv <your-pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

3. Verify that your chosen persistent volume has the right policy:

```
$ oc get pv
```

Example output

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM           STORAGECLASS    REASON   AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Delete         Bound
default/claim1     manual                   10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Delete         Bound
default/claim2     manual                   6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi       RWO          Retain         Bound
default/claim3     manual                   3s
```

In the preceding output, the volume bound to claim **default/claim3** now has a **Retain** reclaim policy. The volume will not be automatically deleted when a user deletes claim **default/claim3**.

3.3. PERSISTENT VOLUMES

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

PersistentVolume 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.

3.3.1. Types of PVs

OpenShift Container Platform supports the following persistent volume plugins:

- AliCloud Disk
- AWS Elastic Block Store (EBS)
- AWS Elastic File Store (EFS)
- Azure Disk
- Azure File
- Cinder
- Fibre Channel
- GCP Persistent Disk
- GCP Filestore
- IBM Power Virtual Server Block
- IBM® VPC Block
- HostPath
- iSCSI
- Local volume
- NFS
- OpenStack Manila
- Red Hat OpenShift Data Foundation
- VMware vSphere

3.3.2. Capacity

Generally, a persistent volume (PV) has a specific storage capacity. This is set by using the **capacity** attribute of the PV.

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

3.3.3. Access modes

A persistent volume 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.

IMPORTANT

Volume access modes describe volume capabilities. They are not enforced constraints. The storage provider is responsible for runtime errors resulting from invalid use of the resource. Errors in the provider show up at runtime as mount errors.

For example, NFS offers **ReadWriteOnce** access mode. If you want to use the volume's ROX capability, mark the claims as **ReadOnlyMany**.

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, delete the pods that use the volumes.

The following table lists the access modes:

Table 3.1. Access modes

Access Mode	CLI abbreviation	Description
ReadWriteOnce	**RWO**	The volume can be mounted as read-write by a single node.
ReadWriteOncePod [1]	**RWOP**	The volume can be mounted as read-write by a single pod on a single node.
ReadOnlyMany	**ROX**	The volume can be mounted as read-only by many nodes.
ReadWriteMany	**RWX**	The volume can be mounted as read-write by many nodes.

1. ReadWriteOncePod access mode for persistent volumes is a Technology Preview feature.

IMPORTANT

ReadWriteOncePod access mode for persistent volumes is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Table 3.2. Supported access modes for persistent volumes

Volume plugin	ReadWriteOnce [1]	ReadWriteOncePod [2]	ReadOnlyMany	ReadWriteMany
AliCloud Disk	▮	▮	-	-
AWS EBS [3]	▮	▮	-	-
AWS EFS	▮	▮	▮	▮
Azure File	▮	▮	▮	▮
Azure Disk	▮	▮	-	-
Cinder	▮	▮	-	-
Fibre Channel	▮	▮	▮	-
GCP Persistent Disk	▮	▮	-	-
GCP Filestore	▮	▮	▮	▮
HostPath	▮	▮	-	-
IBM Power Virtual Server Disk	▮	▮	▮	▮
IBM® VPC Disk	▮	▮	-	-

Volume plugin	ReadWriteOnce [1]	ReadWriteOncePod [2]	ReadOnlyMany	ReadWriteMany
iSCSI	▮	▮	▮	-
Local volume	▮	▮	-	-
NFS	▮	▮	▮	▮
OpenStack Manila	-	▮	-	▮
Red Hat OpenShift Data Foundation	▮	▮	-	▮
VMware vSphere	▮	▮	-	▮ [4]

1. ReadWriteOnce (RWO) volumes cannot be mounted on multiple nodes. If a node fails, the system does not allow the attached RWO volume to be mounted on a new node because it is already assigned to the failed node. If you encounter a multi-attach error message as a result, force delete the pod on a shutdown or crashed node to avoid data loss in critical workloads, such as when dynamic persistent volumes are attached.

2. ReadWriteOncePod is a Technology Preview feature.

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

4. If the underlying vSphere environment supports the vSAN file service, then the vSphere Container Storage Interface (CSI) Driver Operator installed by OpenShift Container Platform supports provisioning of ReadWriteMany (RWX) volumes. If you do not have vSAN file service configured, and you request RWX, the volume fails to get created and an error is logged. For more information, see "Using Container Storage Interface" → "VMware vSphere CSI Driver Operator".

3.3.4. Phase

Volumes can be found in one of the following phases:

Table 3.3. Volume phases

Phase	Description
Available	A free resource not yet bound to a claim.
Bound	The volume is bound to a claim.

Phase	Description
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 that is bound to the PV by running the following command:

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

3.3.4.1. Mount options

You can specify mount options while mounting a PV by using the attribute **mountOptions**.

For example:

Mount options example

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  mountOptions: 1
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

1 Specified mount options are used while mounting the PV to the disk.

The following PV types support mount options:

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- GCE Persistent Disk

- iSCSI
- Local volume
- NFS
- Red Hat OpenShift Data Foundation (Ceph RBD only)
- VMware vSphere

NOTE

Fibre Channel and HostPath PVs do not support mount options.

Additional resources

- ReadWriteMany vSphere volume support

3.4. PERSISTENT VOLUME CLAIMS

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

PersistentVolumeClaim 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.

3.4.1. 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 Cluster Storage Operator might install a default storage class depending on the platform in use. This storage class 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 storage class.

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 storage class is marked as default, a PVC can only be created if the **storageClassName** is explicitly specified. Therefore, only one storage class should be set as the default.

3.4.2. Access modes

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

3.4.3. 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.

3.4.4. Claims as volumes

Pods access storage by using the claim as a volume. Claims must exist in the same namespace as the pod 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. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host **/dev/pts** files. It is safe to mount the host by using **/host**.

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

3.5. BLOCK VOLUME SUPPORT

OpenShift Container Platform can statically provision raw block volumes. These volumes do not have a file system, and can provide performance benefits for applications that either write to the disk directly or implement their own storage service.

Raw block volumes are provisioned by specifying **volumeMode: Block** in the PV and PVC specification.

IMPORTANT

Pods using raw block volumes must be configured to allow privileged containers.

The following table displays which volume plugins support block volumes.

Table 3.4. Block volume support

Volume Plugin	Manually provisioned	Dynamically provisioned	Fully supported
AliCloud Disk	▮	▮	▮
Amazon Elastic Block Store (Amazon EBS)	▮	▮	▮
Amazon Elastic File Storage (Amazon EFS)			
Azure Disk	▮	▮	▮
Azure File			
Cinder	▮	▮	▮
Fibre Channel	▮		▮
GCP	▮	▮	▮
HostPath			
IBM VPC Disk	▮	▮	▮

Volume Plugin	Manually provisioned	Dynamically provisioned	Fully supported
iSCSI	▮		▮
Local volume	▮		▮
NFS			
Red Hat OpenShift Data Foundation	▮	▮	▮
VMware vSphere	▮	▮	▮

IMPORTANT

Using any of the block volumes that can be provisioned manually, but are not provided as fully supported, is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

3.5.1. Block volume examples

PV example

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block 1
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

1 **volumeMode** must be set to **Block** to indicate that this PV is a raw block volume.

PVC example

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ①
  resources:
    requests:
      storage: 10Gi
```

① **volumeMode** must be set to **Block** to indicate that a raw block PVC is requested.

Pod specification example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices: ①
        - name: data
          devicePath: /dev/xvda ②
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc ③
```

① **volumeDevices**, instead of **volumeMounts**, is used for block devices. Only **PersistentVolumeClaim** sources can be used with raw block volumes.

② **devicePath**, instead of **mountPath**, represents the path to the physical device where the raw block is mapped to the system.

③ The volume source must be of type **persistentVolumeClaim** and must match the name of the PVC as expected.

Table 3.5. Accepted values for **volumeMode**

Value	Default
Filesystem	Yes
Block	No

Table 3.6. Binding scenarios for block volumes

PV **volumeMode**	PVC **volumeMode**	Binding result
Filesystem	Filesystem	Bind
Unspecified	Unspecified	Bind
Filesystem	Unspecified	Bind
Unspecified	Filesystem	Bind
Block	Block	Bind
Unspecified	Block	No Bind
Block	Unspecified	No Bind
Filesystem	Block	No Bind
Block	Filesystem	No Bind

IMPORTANT

Unspecified values result in the default value of **Filesystem**.

3.6. USING FSGROUP TO REDUCE POD TIMEOUTS

If a storage volume contains many files (~1,000,000 or greater), you may experience pod timeouts.

This can occur because, by default, OpenShift Container Platform recursively changes ownership and permissions for the contents of each volume to match the **fsGroup** specified in a pod's **securityContext** when that volume is mounted. For large volumes, checking and changing ownership and permissions can be time consuming, slowing pod startup. You can use the **fsGroupChangePolicy** field inside a **securityContext** to control the way that OpenShift Container Platform checks and manages ownership and permissions for a volume.

fsGroupChangePolicy defines behavior for changing ownership and permission of the volume before being exposed inside a pod. This field only applies to volume types that support **fsGroup**-controlled ownership and permissions. This field has two possible values:

- **OnRootMismatch**: Only change permissions and ownership if permission and ownership of root directory does not match with expected permissions of the volume. This can help shorten the time it takes to change ownership and permission of a volume to reduce pod timeouts.

- **Always**: Always change permission and ownership of the volume when a volume is mounted.

fsGroupChangePolicy example

```
securityContext:
```

```
runAsUser: 1000
runAsGroup: 3000
fsGroup: 2000
fsGroupChangePolicy: "OnRootMismatch" 1
...
```

1 **OnRootMismatch** specifies skipping recursive permission change, thus helping to avoid pod timeout problems.

NOTE

The fsGroupChangePolicyfield has no effect on ephemeral volume types, such as secret, configMap, and emptydir.

CHAPTER 4. CONFIGURING PERSISTENT STORAGE

4.1. PERSISTENT STORAGE USING AWS ELASTIC BLOCK STORE

OpenShift Container Platform supports Amazon Elastic Block Store (EBS) volumes. You can provision your OpenShift Container Platform cluster with persistent storage by using Amazon EC2.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. You can dynamically provision Amazon EBS volumes. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users. You can define a KMS key to encrypt container-persistent volumes on AWS. By default, newly created clusters using OpenShift Container Platform version 4.10 and later use gp3 storage and the AWS EBS CSI driver .

IMPORTANT

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

For OpenShift Container Platform, automatic migration from AWS EBS in-tree to the Container Storage Interface (CSI) driver is available as a Technology Preview (TP) feature. With migration enabled, volumes provisioned using the existing in-tree driver are automatically migrated to use the AWS EBS CSI driver. For more information, see CSI automatic migration feature .

4.1.1. Creating the EBS storage class

Storage classes are used to differentiate and delineate storage levels and usages. By defining a storage class, users can obtain dynamically provisioned persistent volumes.

4.1.2. Creating the persistent volume claim

Prerequisites

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure

1. In the OpenShift Container Platform console, click **Storage → Persistent Volume Claims**.
2. In the persistent volume claims overview, click **Create Persistent Volume Claim**.
3. Define the desired options on the page that appears.
   a. Select the previously-created storage class from the drop-down menu.
   b. Enter a unique name for the storage claim.
   c. Select the access mode. This selection determines the read and write access for the storage claim.
   d. Define the size of the storage claim.

4. Click **Create** to create the persistent volume claim and generate a persistent volume.

4.1.3. Volume format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that the volume contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This verification enables you to use unformatted AWS volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

4.1.4. Maximum number of EBS volumes on a node

By default, OpenShift Container Platform supports a maximum of 39 EBS volumes attached to one node. This limit is consistent with the AWS volume limits. The volume limit depends on the instance type.

IMPORTANT

As a cluster administrator, you must use either in-tree or Container Storage Interface (CSI) volumes and their respective storage classes, but never both volume types at the same time. The maximum attached EBS volume number is counted separately for in-tree and CSI volumes, which means you could have up to 39 EBS volumes of each type.

For information about accessing additional storage options, such as volume snapshots, that are not possible with in-tree volume plug-ins, see AWS Elastic Block Store CSI Driver Operator.

4.1.5. Encrypting container persistent volumes on AWS with a KMS key

Defining a KMS key to encrypt container-persistent volumes on AWS is useful when you have explicit compliance and security guidelines when deploying to AWS.

Prerequisites

- Underlying infrastructure must contain storage.
- You must create a customer KMS key on AWS.

Procedure

1. Create a storage class:

```
$ cat << EOF | oc create -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
parameters:
  fsType: ext4 2
  encrypted: "true"
  kmsKeyId: keyvalue 3
provisioner: ebs.csi.aws.com
```

```
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
EOF
```

1. Specifies the name of the storage class.

2. File system that is created on provisioned volumes.

3. Specifies the full Amazon Resource Name (ARN) of the key to use when encrypting the container-persistent volume. If you do not provide any key, but the **encrypted** field is set to **true**, then the default KMS key is used. See Finding the key ID and key ARN on AWS in the AWS documentation.

2. Create a persistent volume claim (PVC) with the storage class specifying the KMS key:

```
$ cat << EOF | oc create -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  storageClassName: <storage-class-name>
  resources:
    requests:
      storage: 1Gi
EOF
```

3. Create workload containers to consume the PVC:

```
$ cat << EOF | oc create -f -
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: httpd
      image: quay.io/centos7/httpd-24-centos7
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: /mnt/storage
          name: data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: mypvc
EOF
```

4.1.6. Additional resources

- See AWS Elastic Block Store CSI Driver Operator for information about accessing additional storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

4.2. PERSISTENT STORAGE USING AZURE

OpenShift Container Platform supports Microsoft Azure Disk volumes. You can provision your OpenShift Container Platform cluster with persistent storage using Azure. Some familiarity with Kubernetes and Azure is assumed. The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. Azure Disk volumes can be provisioned dynamically. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users.

IMPORTANT

OpenShift Container Platform defaults to using an in-tree (non-CSI) plugin to provision Azure Disk storage.

In future OpenShift Container Platform versions, volumes provisioned using existing in-tree plugins are planned for migration to their equivalent CSI driver. CSI automatic migration should be seamless. Migration does not change how you use all existing API objects, such as persistent volumes, persistent volume claims, and storage classes. For more information about migration, see CSI automatic migration .

After full migration, in-tree plugins will eventually be removed in future versions of OpenShift Container Platform.

IMPORTANT

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

Additional resources

- Microsoft Azure Disk

4.2.1. Creating the Azure storage class

Storage classes are used to differentiate and delineate storage levels and usages. By defining a storage class, users can obtain dynamically provisioned persistent volumes.

Procedure

1. In the OpenShift Container Platform console, click **Storage** → **Storage Classes**.

2. In the storage class overview, click **Create Storage Class**

3. Define the desired options on the page that appears.

   a. Enter a name to reference the storage class.

   b. Enter an optional description.

   c. Select the reclaim policy.

d. Select **kubernetes.io/azure-disk** from the drop down list.

i. Enter the storage account type. This corresponds to your Azure storage account SKU tier. Valid options are **Premium_LRS**, **Standard_LRS**, **StandardSSD_LRS**, and **UltraSSD_LRS**.

ii. Enter the kind of account. Valid options are **shared**, **dedicated,** and **managed**.

IMPORTANT

Red Hat only supports the use of **kind: Managed** in the storage class.

With **Shared** and **Dedicated**, Azure creates unmanaged disks, while OpenShift Container Platform creates a managed disk for machine OS (root) disks. But because Azure Disk does not allow the use of both managed and unmanaged disks on a node, unmanaged disks created with **Shared** or **Dedicated** cannot be attached to OpenShift Container Platform nodes.

e. Enter additional parameters for the storage class as desired.

4. Click **Create** to create the storage class.

Additional resources

- Azure Disk Storage Class

4.2.2. Creating the persistent volume claim

Prerequisites

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure

1. In the OpenShift Container Platform console, click **Storage** → **Persistent Volume Claims**.

2. In the persistent volume claims overview, click **Create Persistent Volume Claim**.

3. Define the desired options on the page that appears.

   a. Select the previously-created storage class from the drop-down menu.

   b. Enter a unique name for the storage claim.

   c. Select the access mode. This selection determines the read and write access for the storage claim.

   d. Define the size of the storage claim.

4. Click **Create** to create the persistent volume claim and generate a persistent volume.

4.2.3. Volume format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted Azure volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

4.2.4. Machine sets that deploy machines with ultra disks using PVCs

You can create a machine set running on Azure that deploys machines with ultra disks. Ultra disks are high-performance storage that are intended for use with the most demanding data workloads.

Both the in-tree plugin and CSI driver support using PVCs to enable ultra disks. You can also deploy machines with ultra disks as data disks without creating a PVC.

Additional resources

- Microsoft Azure ultra disks documentation
- Machine sets that deploy machines on ultra disks using CSI PVCs
- Machine sets that deploy machines on ultra disks as data disks

4.2.4.1. Creating machines with ultra disks by using machine sets

You can deploy machines with ultra disks on Azure by editing your machine set YAML file.

Prerequisites

- Have an existing Microsoft Azure cluster.

Procedure

1. Copy an existing Azure **MachineSet** custom resource (CR) and edit it by running the following command:

   ```
   $ oc edit machineset <machine-set-name>
   ```

   where **<machine-set-name>** is the machine set that you want to provision machines with ultra disks.

2. Add the following lines in the positions indicated:

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   spec:
     template:
       spec:
         metadata:
           labels:
             disk: ultrassd 1
         providerSpec:
           value:
             ultraSSDCapability: Enabled 2
   ```

1. Specify a label to use to select a node that is created by this machine set. This procedure uses **disk.ultrassd** for this value.

2. These lines enable the use of ultra disks.

3. Create a machine set using the updated configuration by running the following command:

```
$ oc create -f <machine-set-name>.yaml
```

4. Create a storage class that contains the following YAML definition:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ultra-disk-sc 1
parameters:
  cachingMode: None
  diskIopsReadWrite: "2000" 2
  diskMbpsReadWrite: "320" 3
  kind: managed
  skuname: UltraSSD_LRS
provisioner: disk.csi.azure.com 4
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer 5
```

1. Specify the name of the storage class. This procedure uses **ultra-disk-sc** for this value.

2. Specify the number of IOPS for the storage class.

3. Specify the throughput in MBps for the storage class.

4. For Azure Kubernetes Service (AKS) version 1.21 or later, use **disk.csi.azure.com**. For earlier versions of AKS, use **kubernetes.io/azure-disk**.

5. Optional: Specify this parameter to wait for the creation of the pod that will use the disk.

5. Create a persistent volume claim (PVC) to reference the **ultra-disk-sc** storage class that contains the following YAML definition:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ultra-disk 1
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: ultra-disk-sc 2
  resources:
   requests:
     storage: 4Gi 3
```

1 Specify the name of the PVC. This procedure uses **ultra-disk** for this value.

2 This PVC references the **ultra-disk-sc** storage class.

3 Specify the size for the storage class. The minimum value is **4Gi**.

6. Create a pod that contains the following YAML definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-ultra
spec:
  nodeSelector:
    disk: ultrassd 1
  containers:
  - name: nginx-ultra
    image: alpine:latest
    command:
      - "sleep"
      - "infinity"
    volumeMounts:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: ultra-disk 2
```

1 Specify the label of the machine set that enables the use of ultra disks. This procedure uses **disk.ultrassd** for this value.

2 This pod references the **ultra-disk** PVC.

Verification

1. Validate that the machines are created by running the following command:

```
$ oc get machines
```

The machines should be in the **Running** state.

2. For a machine that is running and has a node attached, validate the partition by running the following command:

```
$ oc debug node/<node-name> -- chroot /host lsblk
```

In this command, **oc debug node/<node-name>** starts a debugging shell on the node **<node-name>** and passes a command with **--**. The passed command **chroot /host** provides access to the underlying host OS binaries, and **lsblk** shows the block devices that are attached to the host OS machine.

Next steps

- To use an ultra disk from within a pod, create a workload that uses the mount point. Create a YAML file similar to the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: ssd-benchmark1
spec:
  containers:
  - name: ssd-benchmark1
    image: nginx
    ports:
      - containerPort: 80
        name: "http-server"
    volumeMounts:
    - name: lun0p1
      mountPath: "/tmp"
  volumes:
    - name: lun0p1
      hostPath:
        path: /var/lib/lun0p1
        type: DirectoryOrCreate
  nodeSelector:
    disktype: ultrassd
```

4.2.4.2. Troubleshooting resources for machine sets that enable ultra disks

Use the information in this section to understand and recover from issues you might encounter.

4.2.4.2.1. Unable to mount a persistent volume claim backed by an ultra disk

If there is an issue mounting a persistent volume claim backed by an ultra disk, the pod becomes stuck in the **ContainerCreating** state and an alert is triggered.

For example, if the **additionalCapabilities.ultraSSDEnabled** parameter is not set on the machine that backs the node that hosts the pod, the following error message appears:

```
StorageAccountType UltraSSD_LRS can be used only when additionalCapabilities.ultraSSDEnabled
is set.
```

- To resolve this issue, describe the pod by running the following command:

```
$ oc -n <stuck_pod_namespace> describe pod <stuck_pod_name>
```

4.3. PERSISTENT STORAGE USING AZURE FILE

OpenShift Container Platform supports Microsoft Azure File volumes. You can provision your OpenShift Container Platform cluster with persistent storage using Azure. Some familiarity with Kubernetes and Azure is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. You can provision Azure File volumes dynamically.

Persistent volumes are not bound to a single project or namespace, and you can share them across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace, and can be requested by users for use in applications.

IMPORTANT

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

IMPORTANT

Azure File volumes use Server Message Block.

IMPORTANT

In future OpenShift Container Platform versions, volumes provisioned using existing in-tree plugins are planned for migration to their equivalent CSI driver. CSI automatic migration should be seamless. Migration does not change how you use all existing API objects, such as persistent volumes, persistent volume claims, and storage classes. For more information about migration, see CSI automatic migration .

After full migration, in-tree plugins will eventually be removed in future versions of OpenShift Container Platform.

Additional resources

- Azure Files

4.3.1. Create the Azure File share persistent volume claim

To create the persistent volume claim, you must first define a **Secret** object that contains the Azure account and key. This secret is used in the **PersistentVolume** definition, and will be referenced by the persistent volume claim for use in applications.

Prerequisites

- An Azure File share exists.
- The credentials to access this share, specifically the storage account and key, are available.

Procedure

1. Create a **Secret** object that contains the Azure File credentials:

```
$ oc create secret generic <secret-name> --from-literal=azurestorageaccountname=
<storage-account> \ 1
  --from-literal=azurestorageaccountkey=<storage-account-key> 2
```

1 The Azure File storage account name.

2 The Azure File storage account key.

2. Create a **PersistentVolume** object that references the **Secret** object you created:

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  storageClassName: azure-file-sc
  azureFile:
    secretName: <secret-name> ❸
    shareName: share-1 ❹
    readOnly: false
```

1. The name of the persistent volume.
2. The size of this persistent volume.
3. The name of the secret that contains the Azure File share credentials.
4. The name of the Azure File share.

3. Create a **PersistentVolumeClaim** object that maps to the persistent volume you created:

```
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "claim1" ❶
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi" ❷
  storageClassName: azure-file-sc ❸
  volumeName: "pv0001" ❹
```

1. The name of the persistent volume claim.
2. The size of this persistent volume claim.
3. The name of the storage class that is used to provision the persistent volume. Specify the storage class used in the **PersistentVolume** definition.
4. The name of the existing **PersistentVolume** object that references the Azure File share.

4.3.2. Mount the Azure File share in a pod

After the persistent volume claim has been created, it can be used inside by an application. The following example demonstrates mounting this share inside of a pod.

Prerequisites

- A persistent volume claim exists that is mapped to the underlying Azure File share.

Procedure

- Create a pod that mounts the existing persistent volume claim:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name ①
spec:
  containers:
    ...
    volumeMounts:
    - mountPath: "/data" ②
      name: azure-file-share
  volumes:
    - name: azure-file-share
      persistentVolumeClaim:
        claimName: claim1 ③
```

① The name of the pod.

② The path to mount the Azure File share inside the pod. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host **/dev/pts** files. It is safe to mount the host by using **/host**.

③ The name of the **PersistentVolumeClaim** object that has been previously created.

4.4. PERSISTENT STORAGE USING CINDER

OpenShift Container Platform supports OpenStack Cinder. Some familiarity with Kubernetes and OpenStack is assumed.

Cinder volumes can be provisioned dynamically. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users.

IMPORTANT

OpenShift Container Platform defaults to using an in-tree (non-CSI) plugin to provision Cinder storage.

In future OpenShift Container Platform versions, volumes provisioned using existing in-tree plugins are planned for migration to their equivalent CSI driver. CSI automatic migration should be seamless. Migration does not change how you use all existing API objects, such as persistent volumes, persistent volume claims, and storage classes. For more information about migration, see CSI automatic migration .

After full migration, in-tree plugins will eventually be removed in future versions of OpenShift Container Platform.

Additional resources

- For more information about how OpenStack Block Storage provides persistent block storage management for virtual hard drives, see OpenStack Cinder.

4.4.1. Manual provisioning with Cinder

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Prerequisites

- OpenShift Container Platform configured for Red Hat OpenStack Platform (RHOSP)
- Cinder volume ID

4.4.1.1. Creating the persistent volume

You must define your persistent volume (PV) in an object definition before creating it in OpenShift Container Platform:

Procedure

1. Save your object definition to a file.

cinder-persistentvolume.yaml

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ①
spec:
  capacity:
    storage: "5Gi" ②
  accessModes:
    - "ReadWriteOnce"
  cinder: ③
    fsType: "ext3" ④
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" ⑤
```

① The name of the volume that is used by persistent volume claims or pods.

② The amount of storage allocated to this volume.

③ Indicates **cinder** for Red Hat OpenStack Platform (RHOSP) Cinder volumes.

④ The file system that is created when the volume is mounted for the first time.

⑤ The Cinder volume to use.

> IMPORTANT
>
> Do not change the **fstype** parameter value after the volume is formatted and provisioned. Changing this value can result in data loss and pod failure.

2. Create the object definition file you saved in the previous step.

```
$ oc create -f cinder-persistentvolume.yaml
```

4.4.1.2. Persistent volume formatting

You can use unformatted Cinder volumes as PVs because OpenShift Container Platform formats them before the first use.

Before OpenShift Container Platform mounts the volume and passes it to a container, the system checks that it contains a file system as specified by the **fsType** parameter in the PV definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

4.4.1.3. Cinder volume security

If you use Cinder PVs in your application, configure security for their deployment configurations.

Prerequisites

- An SCC must be created that uses the appropriate **fsGroup** strategy.

Procedure

1. Create a service account and add it to the SCC:

```
$ oc create serviceaccount <service_account>
```

```
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n <project>
```

2. In your application's deployment configuration, provide the service account name and **securityContext**:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 (1)
  selector:   (2)
    name: frontend
  template:   (3)
    metadata:
      labels: (4)
        name: frontend (5)
    spec:
      containers:
      - image: openshift/hello-openshift
```

```
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
      serviceAccountName: <service_account> ❻
      securityContext:
        fsGroup: 7777 ❼
```

1. The number of copies of the pod to run.
2. The label selector of the pod to run.
3. A template for the pod that the controller creates.
4. The labels on the pod. They must include labels from the label selector.
5. The maximum name length after expanding any parameters is 63 characters.
6. Specifies the service account you created.
7. Specifies an **fsGroup** for the pods.

4.5. PERSISTENT STORAGE USING FIBRE CHANNEL

OpenShift Container Platform supports Fibre Channel, allowing you to provision your OpenShift Container Platform cluster with persistent storage using Fibre channel volumes. Some familiarity with Kubernetes and Fibre Channel is assumed.

> **IMPORTANT**
>
> Persistent storage using Fibre Channel is not supported on ARM architecture based infrastructures.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users.

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

Additional resources

- Using Fibre Channel devices

4.5.1. Provisioning

To provision Fibre Channel volumes using the **PersistentVolume** API the following must be available:

- The **targetWWNs** (array of Fibre Channel target's World Wide Names).

- A valid LUN number.
- The filesystem type.

A persistent volume and a LUN have a one-to-one mapping between them.

Prerequisites

- Fibre Channel LUNs must exist in the underlying infrastructure.

PersistentVolume object definition

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    wwids: [scsi-3600508b400105e210000900000490000] ①
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] ②
    lun: 2 ③
    fsType: ext4
```

① World wide identifiers (WWIDs). Either FC **wwids** or a combination of FC **targetWWNs** and **lun** must be set, but not both simultaneously. The FC WWID identifier is recommended over the WWNs target because it is guaranteed to be unique for every storage device, and independent of the path that is used to access the device. The WWID identifier can be obtained by issuing a SCSI Inquiry to retrieve the Device Identification Vital Product Data (**page 0x83**) or Unit Serial Number (**page 0x80**). FC WWIDs are identified as **/dev/disk/by-id/** to reference the data on the disk, even if the path to the device changes and even when accessing the device from different systems.

② ③ Fibre Channel WWNs are identified as **/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>**, but you do not need to provide any part of the path leading up to the **WWN**, including the **0x**, and anything after, including the **-** (hyphen).

IMPORTANT

Changing the value of the **fstype** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

4.5.1.1. Enforcing disk quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is mapped to a single persistent volume, and unique names must be used for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount, such as 10Gi, and be matched with a corresponding volume of equal or greater capacity.

4.5.1.2. Fibre Channel volume security

Users request storage with a persistent volume claim. This claim only lives in the user's namespace, and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each Fibre Channel LUN must be accessible by all nodes in the cluster.

4.6. PERSISTENT STORAGE USING FLEXVOLUME

IMPORTANT

FlexVolume is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

Out-of-tree Container Storage Interface (CSI) driver is the recommended way to write volume drivers in OpenShift Container Platform. Maintainers of FlexVolume drivers should implement a CSI driver and move users of FlexVolume to CSI. Users of FlexVolume should move their workloads to CSI driver.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

OpenShift Container Platform supports FlexVolume, an out-of-tree plugin that uses an executable model to interface with drivers.

To use storage from a back-end that does not have a built-in plugin, you can extend OpenShift Container Platform through FlexVolume drivers and provide persistent storage to applications.

Pods interact with FlexVolume drivers through the **flexvolume** in-tree plugin.

Additional resources

- Expanding persistent volumes

4.6.1. About FlexVolume drivers

A FlexVolume driver is an executable file that resides in a well-defined directory on all nodes in the cluster. OpenShift Container Platform calls the FlexVolume driver whenever it needs to mount or unmount a volume represented by a **PersistentVolume** object with **flexVolume** as the source.

IMPORTANT

Attach and detach operations are not supported in OpenShift Container Platform for FlexVolume.

4.6.2. FlexVolume driver example

The first command-line argument of the FlexVolume driver is always an operation name. Other parameters are specific to each operation. Most of the operations take a JavaScript Object Notation (JSON) string as a parameter. This parameter is a complete JSON string, and not the name of a file with the JSON data.

The FlexVolume driver contains:

- All **flexVolume.options**.
- Some options from **flexVolume** prefixed by **kubernetes.io/**, such as **fsType** and **readwrite**.
- The content of the referenced secret, if specified, prefixed by **kubernetes.io/secret/**.

FlexVolume driver JSON input example

```
{
 "fooServer": "192.168.0.1:1234", 1
        "fooVolumeName": "bar",
 "kubernetes.io/fsType": "ext4", 2
 "kubernetes.io/readwrite": "ro", 3
 "kubernetes.io/secret/<key name>": "<key value>", 4
 "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

1 All options from **flexVolume.options**.

2 The value of **flexVolume.fsType**.

3 **ro**/**rw** based on **flexVolume.readOnly**.

4 All keys and their values from the secret referenced by **flexVolume.secretRef**.

OpenShift Container Platform expects JSON data on standard output of the driver. When not specified, the output describes the result of the operation.

FlexVolume driver default output example

```
{
 "status": "<Success/Failure/Not supported>",
 "message": "<Reason for success/failure>"
}
```

Exit code of the driver should be **0** for success and **1** for error.

Operations should be idempotent, which means that the mounting of an already mounted volume should result in a successful operation.

4.6.3. Installing FlexVolume drivers

FlexVolume drivers that are used to extend OpenShift Container Platform are executed only on the node. To implement FlexVolumes, a list of operations to call and the installation path are all that is required.

Prerequisites

- FlexVolume drivers must implement these operations:

  init

Initializes the driver. It is called during initialization of all nodes.

- Arguments: none
- Executed on: node
- Expected output: default JSON

mount

Mounts a volume to directory. This can include anything that is necessary to mount the volume, including finding the device and then mounting the device.

- Arguments: **<mount-dir> <json>**
- Executed on: node
- Expected output: default JSON

unmount

Unmounts a volume from a directory. This can include anything that is necessary to clean up the volume after unmounting.

- Arguments: **<mount-dir>**
- Executed on: node
- Expected output: default JSON

mountdevice

Mounts a volume's device to a directory where individual pods can then bind mount.

This call-out does not pass "secrets" specified in the FlexVolume spec. If your driver requires secrets, do not implement this call-out.

- Arguments: **<mount-dir> <json>**
- Executed on: node
- Expected output: default JSON

unmountdevice

Unmounts a volume's device from a directory.

- Arguments: **<mount-dir>**
- Executed on: node
- Expected output: default JSON
  - All other operations should return JSON with **{"status": "Not supported"}** and exit code **1**.

Procedure

To install the FlexVolume driver:

1. Ensure that the executable file exists on all nodes in the cluster.

2. Place the executable file at the volume plugin path: **/etc/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>**.

For example, to install the FlexVolume driver for the storage **foo**, place the executable file at: **/etc/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo**.

4.6.4. Consuming storage using FlexVolume drivers

Each **PersistentVolume** object in OpenShift Container Platform represents one storage asset in the storage back-end, such as a volume.

Procedure

- Use the **PersistentVolume** object to reference the installed storage.

Persistent volume object definition using FlexVolume drivers example

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ①
spec:
  capacity:
    storage: 1Gi ②
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo ③
    fsType: "ext4" ④
    secretRef: foo-secret ⑤
    readOnly: true ⑥
    options: ⑦
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

① The name of the volume. This is how it is identified through persistent volume claims or from pods. This name can be different from the name of the volume on back-end storage.

② The amount of storage allocated to this volume.

③ The name of the driver. This field is mandatory.

④ The file system that is present on the volume. This field is optional.

⑤ The reference to a secret. Keys and values from this secret are provided to the FlexVolume driver on invocation. This field is optional.

⑥ The read-only flag. This field is optional.

⑦ The additional options for the FlexVolume driver. In addition to the flags specified by the user in the **options** field, the following flags are also passed to the executable:

```
"fsType":"<FS type>",
"readwrite":"<rw>",
"secret/key1":"<secret1>"
...
"secret/keyN":"<secretN>"
```

NOTE

Secrets are passed only to mount or unmount call-outs.

4.7. PERSISTENT STORAGE USING GCE PERSISTENT DISK

OpenShift Container Platform supports GCE Persistent Disk volumes (gcePD). You can provision your OpenShift Container Platform cluster with persistent storage using GCE. Some familiarity with Kubernetes and GCE is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

GCE Persistent Disk volumes can be provisioned dynamically.

Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users.

IMPORTANT

OpenShift Container Platform defaults to using an in-tree (non-CSI) plugin to provision gcePD storage.

In future OpenShift Container Platform versions, volumes provisioned using existing in-tree plug-ins are planned for migration to their equivalent CSI driver. CSI automatic migration should be seamless. Migration does not change how you use all existing API objects, such as persistent volumes, persistent volume claims, and storage classes. For more information about migration, see CSI automatic migration .

After full migration, in-tree plugins will eventually be removed in future versions of OpenShift Container Platform.

IMPORTANT

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

Additional resources

- GCE Persistent Disk

4.7.1. Creating the GCE storage class

Storage classes are used to differentiate and delineate storage levels and usages. By defining a storage class, users can obtain dynamically provisioned persistent volumes.

4.7.2. Creating the persistent volume claim

Prerequisites

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure

1. In the OpenShift Container Platform console, click **Storage → Persistent Volume Claims**.
2. In the persistent volume claims overview, click **Create Persistent Volume Claim**.
3. Define the desired options on the page that appears.
    a. Select the previously-created storage class from the drop-down menu.
    b. Enter a unique name for the storage claim.
    c. Select the access mode. This selection determines the read and write access for the storage claim.
    d. Define the size of the storage claim.
4. Click **Create** to create the persistent volume claim and generate a persistent volume.

4.7.3. Volume format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that the volume contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This verification enables you to use unformatted GCE volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

4.8. PERSISTENT STORAGE USING ISCSI

You can provision your OpenShift Container Platform cluster with persistent storage using iSCSI. Some familiarity with Kubernetes and iSCSI is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

IMPORTANT

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

IMPORTANT

When you use iSCSI on Amazon Web Services, you must update the default security policy to include TCP traffic between nodes on the iSCSI ports. By default, they are ports **860** and **3260**.

IMPORTANT

Users must ensure that the iSCSI initiator is already configured on all OpenShift Container Platform nodes by installing the **iscsi-initiator-utils** package and configuring their initiator name in **/etc/iscsi/initiatorname.iscsi**. The **iscsi-initiator-utils** package is already installed on deployments that use Red Hat Enterprise Linux CoreOS (RHCOS).

For more information, see Managing Storage Devices.

4.8.1. Provisioning

Verify that the storage exists in the underlying infrastructure before mounting it as a volume in OpenShift Container Platform. All that is required for the iSCSI is the iSCSI target portal, a valid iSCSI Qualified Name (IQN), a valid LUN number, the filesystem type, and the **PersistentVolume** API.

PersistentVolume object definition

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
     targetPortal: 10.16.154.81:3260
     iqn: iqn.2014-12.example.server:storage.target00
     lun: 0
     fsType: 'ext4'
```

4.8.2. Enforcing disk quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is one persistent volume. Kubernetes enforces unique names for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (for example, **10Gi**) and be matched with a corresponding volume of equal or greater capacity.

4.8.3. iSCSI volume security

Users request storage with a **PersistentVolumeClaim** object. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume claim across a namespace causes the pod to fail.

Each iSCSI LUN must be accessible by all nodes in the cluster.

4.8.3.1. Challenge Handshake Authentication Protocol (CHAP) configuration

Optionally, OpenShift Container Platform can use CHAP to authenticate itself to iSCSI targets:

```
apiVersion: v1
kind: PersistentVolume
```

```
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    chapAuthDiscovery: true ①
    chapAuthSession: true ②
    secretRef:
      name: chap-secret ③
```

① Enable CHAP authentication of iSCSI discovery.

② Enable CHAP authentication of iSCSI session.

③ Specify name of Secrets object with user name + password. This **Secret** object must be available in all namespaces that can use the referenced volume.

4.8.4. iSCSI multipathing

For iSCSI-based storage, you can configure multiple paths by using the same IQN for more than one target portal IP address. Multipathing ensures access to the persistent volume when one or more of the components in a path fail.

To specify multi-paths in the pod specification, use the **portals** field. For example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] ①
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

① Add additional target portals using the **portals** field.

4.8.5. iSCSI custom initiator IQN

Configure the custom initiator iSCSI Qualified Name (IQN) if the iSCSI targets are restricted to certain IQNs, but the nodes that the iSCSI PVs are attached to are not guaranteed to have these IQNs.

To specify a custom initiator IQN, use **initiatorName** field.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn 1
    fsType: ext4
    readOnly: false
```

1 Specify the name of the initiator.

4.9. PERSISTENT STORAGE USING NFS

OpenShift Container Platform clusters can be provisioned with persistent storage using NFS. Persistent volumes (PVs) and persistent volume claims (PVCs) provide a convenient method for sharing a volume across a project. While the NFS-specific information contained in a PV definition could also be defined directly in a **Pod** definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

Additional resources

- Network File System (NFS)

4.9.1. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. To provision NFS volumes, a list of NFS servers and export paths are all that is required.

Procedure

1. Create an object definition for the PV:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
```

```
  storage: 5Gi ❷
accessModes:
- ReadWriteOnce ❸
nfs: ❹
  path: /tmp ❺
  server: 172.17.0.2 ❻
persistentVolumeReclaimPolicy: Retain ❼
```

❶ The name of the volume. This is the PV identity in various **oc <command> pod** commands.

❷ The amount of storage allocated to this volume.

❸ Though this appears to be related to controlling access to the volume, it is actually used similarly to labels and used to match a PVC to a PV. Currently, no access rules are enforced based on the **accessModes**.

❹ The volume type being used, in this case the **nfs** plugin.

❺ The path that is exported by the NFS server.

❻ The hostname or IP address of the NFS server.

❼ The reclaim policy for the PV. This defines what happens to a volume when released.

NOTE

Each NFS volume must be mountable by all schedulable nodes in the cluster.

2. Verify that the PV was created:

```
$ oc get pv
```

Example output

```
NAME    LABELS    CAPACITY    ACCESSMODES  STATUS     CLAIM  REASON    AGE
pv0001  <none>    5Gi         RWO          Available                   31s
```

3. Create a persistent volume claim that binds to the new PV:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce ❶
  resources:
    requests:
      storage: 5Gi ❷
  volumeName: pv0001
  storageClassName: ""
```

1 The access modes do not enforce security, but rather act as labels to match a PV to a PVC.

2 This claim looks for PVs offering **5Gi** or greater capacity.

4. Verify that the persistent volume claim was created:

```
$ oc get pvc
```

Example output

```
NAME         STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-claim1   Bound    pv0001   5Gi        RWO                           2m
```

4.9.2. Enforcing disk quotas

You can use disk partitions to enforce disk quotas and size constraints. Each partition can be its own export. Each export is one PV. OpenShift Container Platform enforces unique names for PVs, but the uniqueness of the NFS volume's server and path is up to the administrator.

Enforcing quotas in this way allows the developer to request persistent storage by a specific amount, such as 10Gi, and be matched with a corresponding volume of equal or greater capacity.

4.9.3. NFS volume security

This section covers NFS volume security, including matching permissions and SELinux considerations. The user is expected to understand the basics of POSIX permissions, process UIDs, supplemental groups, and SELinux.

Developers request NFS storage by referencing either a PVC by name or the NFS volume plugin directly in the **volumes** section of their **Pod** definition.

The **/etc/exports** file on the NFS server contains the accessible NFS directories. The target NFS directory has POSIX owner and group IDs. The OpenShift Container Platform NFS plugin mounts the container's NFS directory with the same POSIX ownership and permissions found on the exported NFS directory. However, the container is not run with its effective UID equal to the owner of the NFS mount, which is the desired behavior.

As an example, if the target NFS directory appears on the NFS server as:

```
$ ls -lZ /opt/nfs -d
```

Example output

```
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

```
$ id nfsnobody
```

Example output

```
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

Then the container must match SELinux labels, and either run with a UID of **65534**, the **nfsnobody** owner, or with **5555** in its supplemental groups to access the directory.

NOTE

The owner ID of **65534** is used as an example. Even though NFS's **root_squash** maps **root**, uid **0**, to **nfsnobody**, uid **65534**, NFS exports can have arbitrary owner IDs. Owner **65534** is not required for NFS exports.

4.9.3.1. Group IDs

The recommended way to handle NFS access, assuming it is not an option to change permissions on the NFS export, is to use supplemental groups. Supplemental groups in OpenShift Container Platform are used for shared storage, of which NFS is an example. In contrast, block storage such as iSCSI uses the **fsGroup** SCC strategy and the **fsGroup** value in the **securityContext** of the pod.

NOTE

To gain access to persistent storage, it is generally preferable to use supplemental group IDs versus user IDs.

Because the group ID on the example target NFS directory is **5555**, the pod can define that group ID using **supplementalGroups** under the **securityContext** definition of the pod. For example:

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1 **securityContext** must be defined at the pod level, not under a specific container.

2 An array of GIDs defined for the pod. In this case, there is one element in the array. Additional GIDs would be comma-separated.

Assuming there are no custom SCCs that might satisfy the pod requirements, the pod likely matches the **restricted** SCC. This SCC has the **supplementalGroups** strategy set to **RunAsAny**, meaning that any supplied group ID is accepted without range checking.

As a result, the above pod passes admissions and is launched. However, if group ID range checking is desired, a custom SCC is the preferred solution. A custom SCC can be created such that minimum and maximum group IDs are defined, group ID range checking is enforced, and a group ID of **5555** is allowed.

NOTE

To use a custom SCC, you must first add it to the appropriate service account. For example, use the **default** service account in the given project unless another has been specified on the **Pod** specification.

4.9.3.2. User IDs

User IDs can be defined in the container image or in the **Pod** definition.

NOTE

It is generally preferable to use supplemental group IDs to gain access to persistent storage versus using user IDs.

In the example target NFS directory shown above, the container needs its UID set to **65534**, ignoring group IDs for the moment, so the following can be added to the **Pod** definition:

```
spec:
  containers: ①
  - name:
  ...
    securityContext:
      runAsUser: 65534 ②
```

① Pods contain a **securityContext** definition specific to each container and a pod's **securityContext** which applies to all containers defined in the pod.

② **65534** is the **nfsnobody** user.

Assuming that the project is **default** and the SCC is **restricted**, the user ID of **65534** as requested by the pod is not allowed. Therefore, the pod fails for the following reasons:

- It requests **65534** as its user ID.
- All SCCs available to the pod are examined to see which SCC allows a user ID of **65534**. While all policies of the SCCs are checked, the focus here is on user ID.
- Because all available SCCs use **MustRunAsRange** for their **runAsUser** strategy, UID range checking is required.
- **65534** is not included in the SCC or project's user ID range.

It is generally considered a good practice not to modify the predefined SCCs. The preferred way to fix this situation is to create a custom SCC A custom SCC can be created such that minimum and maximum user IDs are defined, UID range checking is still enforced, and the UID of **65534** is allowed.

NOTE

To use a custom SCC, you must first add it to the appropriate service account. For example, use the **default** service account in the given project unless another has been specified on the **Pod** specification.

4.9.3.3. SELinux

Red Hat Enterprise Linux (RHEL) and Red Hat Enterprise Linux CoreOS (RHCOS) systems are configured to use SELinux on remote NFS servers by default.

For non-RHEL and non-RHCOS systems, SELinux does not allow writing from a pod to a remote NFS server. The NFS volume mounts correctly but it is read-only. You will need to enable the correct SELinux permissions by using the following procedure.

Prerequisites

- The **container-selinux** package must be installed. This package provides the **virt_use_nfs** SELinux boolean.

Procedure

- Enable the **virt_use_nfs** boolean using the following command. The **-P** option makes this boolean persistent across reboots.

  ```
  # setsebool -P virt_use_nfs 1
  ```

4.9.3.4. Export settings

To enable arbitrary container users to read and write the volume, each exported volume on the NFS server should conform to the following conditions:

- Every export must be exported using the following format:

  ```
  /<example_fs> *(rw,root_squash)
  ```

- The firewall must be configured to allow traffic to the mount point.
  - For NFSv4, configure the default port **2049** (**nfs**).

    NFSv4

    ```
    # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
    ```

  - For NFSv3, there are three ports to configure: **2049** (**nfs**), **20048** (**mountd**), and **111** (**portmapper**).

    NFSv3

    ```
    # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
    ```

    ```
    # iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
    ```

    ```
    # iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
    ```

- The NFS export and directory must be set up so that they are accessible by the target pods. Either set the export to be owned by the container's primary UID, or supply the pod group access using **supplementalGroups**, as shown in the group IDs above.

4.9.4. Reclaiming resources

NFS implements the OpenShift Container Platform **Recyclable** plugin interface. Automatic processes handle reclamation tasks based on policies set on each persistent volume.

By default, PVs are set to **Retain**.

Once claim to a PVC is deleted, and the PV is released, the PV object should not be reused. Instead, a new PV should be created with the same basic volume details as the original.

For example, the administrator creates a PV named **nfs1**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

The user creates **PVC1**, which binds to **nfs1**. The user then deletes **PVC1**, releasing claim to **nfs1**. This results in **nfs1** being **Released**. If the administrator wants to make the same NFS share available, they should create a new PV with the same NFS server details, but a different PV name:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

Deleting the original PV and re-creating it with the same name is discouraged. Attempting to manually change the status of a PV from **Released** to **Available** causes errors and potential data loss.

4.9.5. Additional configuration and troubleshooting

Depending on what version of NFS is being used and how it is configured, there may be additional configuration steps needed for proper export and security mapping. The following are some that may apply:

NFSv4 mount incorrectly shows all files with ownership of **nobody:nobody**	● Could be attributed to the ID mapping settings, found in **/etc/idmapd.conf** on your NFS. ● See this Red Hat Solution.
Disabling ID mapping on NFSv4	● On both the NFS client and server, run: `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

4.10. RED HAT OPENSHIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation is a provider of agnostic persistent storage for OpenShift Container Platform supporting file, block, and object storage, either in-house or in hybrid clouds. As a Red Hat storage solution, Red Hat OpenShift Data Foundation is completely integrated with OpenShift Container Platform for deployment, management, and monitoring.

Red Hat OpenShift Data Foundation provides its own documentation library. The complete set of Red Hat OpenShift Data Foundation documentation is available at https://access.redhat.com/documentation/en-us/red_hat_openshift_data_foundation.

IMPORTANT

OpenShift Data Foundation on top of Red Hat Hyperconverged Infrastructure (RHHI) for Virtualization, which uses hyperconverged nodes that host virtual machines installed with OpenShift Container Platform, is not a supported configuration. For more information about supported platforms, see the Red Hat OpenShift Data Foundation Supportability and Interoperability Guide.

4.11. PERSISTENT STORAGE USING VMWARE VSPHERE VOLUMES

OpenShift Container Platform allows use of VMware vSphere's Virtual Machine Disk (VMDK) volumes. You can provision your OpenShift Container Platform cluster with persistent storage using VMware vSphere. Some familiarity with Kubernetes and VMware vSphere is assumed.

VMware vSphere volumes can be provisioned dynamically. OpenShift Container Platform creates the disk in vSphere and attaches this disk to the correct image.

NOTE

OpenShift Container Platform provisions new volumes as independent persistent disks that can freely attach and detach the volume on any node in the cluster. Consequently, you cannot back up volumes that use snapshots, or restore volumes from snapshots. See Snapshot Limitations for more information.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. Persistent volume claims are specific to a project or namespace and can be requested by users.

IMPORTANT

OpenShift Container Platform defaults to using an in-tree (non-CSI) plugin to provision vSphere storage.

In future OpenShift Container Platform versions, volumes provisioned using existing in-tree plugins are planned for migration to their equivalent CSI driver. CSI automatic migration should be seamless. Migration does not change how you use all existing API objects, such as persistent volumes, persistent volume claims, and storage classes. For more information about migration, see CSI automatic migration .

After full migration, in-tree plugins will eventually be removed in future versions of OpenShift Container Platform.

Additional resources

- VMware vSphere

4.11.1. Dynamically provisioning VMware vSphere volumes

Dynamically provisioning VMware vSphere volumes is the recommended method.

4.11.2. Prerequisites

- An OpenShift Container Platform cluster installed on a VMware vSphere version that meets the requirements for the components that you use. See Installing a cluster on vSphere for information about vSphere version support.

You can use either of the following procedures to dynamically provision these volumes using the default storage class.

4.11.2.1. Dynamically provisioning VMware vSphere volumes using the UI

OpenShift Container Platform installs a default storage class, named **thin**, that uses the **thin** disk format for provisioning volumes.

Prerequisites

- Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure

1. In the OpenShift Container Platform console, click **Storage** → **Persistent Volume Claims**.
2. In the persistent volume claims overview, click **Create Persistent Volume Claim**.
3. Define the required options on the resulting page.
   a. Select the **thin** storage class.
   b. Enter a unique name for the storage claim.
   c. Select the access mode to determine the read and write access for the created storage claim.
   d. Define the size of the storage claim.
4. Click **Create** to create the persistent volume claim and generate a persistent volume.

4.11.2.2. Dynamically provisioning VMware vSphere volumes using the CLI

OpenShift Container Platform installs a default StorageClass, named **thin**, that uses the **thin** disk format for provisioning volumes.

Prerequisites

- Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure (CLI)

1. You can define a VMware vSphere PersistentVolumeClaim by creating a file, **pvc.yaml**, with the following contents:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc ①
spec:
  accessModes:
  - ReadWriteOnce ②
  resources:
    requests:
      storage: 1Gi ③
```

① A unique name that represents the persistent volume claim.

② The access mode of the persistent volume claim. With **ReadWriteOnce**, the volume can be mounted with read and write permissions by a single node.

③ The size of the persistent volume claim.

2. Create the **PersistentVolumeClaim** object from the file:

```
$ oc create -f pvc.yaml
```

4.11.3. Statically provisioning VMware vSphere volumes

To statically provision VMware vSphere volumes you must create the virtual machine disks for reference by the persistent volume framework.

Prerequisites

- Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform.

Procedure

1. Create the virtual machine disks. Virtual machine disks (VMDKs) must be created manually before statically provisioning VMware vSphere volumes. Use either of the following methods:

   - Create using **vmkfstools**. Access ESX through Secure Shell (SSH) and then use following command to create a VMDK volume:

   ```
   $ vmkfstools -c <size> /vmfs/volumes/<datastore-name>/volumes/<disk-name>.vmdk
   ```

   - Create using **vmware-diskmanager**:

   ```
   $ shell vmware-vdiskmanager -c -t 0 -s <size> -a lsilogic <disk-name>.vmdk
   ```

2. Create a persistent volume that references the VMDKs. Create a file, **pv1.yaml**, with the **PersistentVolume** object definition:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1 ①
spec:
  capacity:
    storage: 1Gi ②
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: ③
    volumePath: "[datastore1] volumes/myDisk"  ④
    fsType: ext4  ⑤
```

① The name of the volume. This name is how it is identified by persistent volume claims or pods.

② The amount of storage allocated to this volume.

③ The volume type used, with **vsphereVolume** for vSphere volumes. The label is used to mount a vSphere VMDK volume into pods. The contents of a volume are preserved when it is unmounted. The volume type supports VMFS and VSAN datastore.

④ The existing VMDK volume to use. If you used **vmkfstools**, you must enclose the datastore name in square brackets, **[]**, in the volume definition, as shown previously.

⑤ The file system type to mount. For example, ext4, xfs, or other file systems.

IMPORTANT

Changing the value of the fsType parameter after the volume is formatted and provisioned can result in data loss and pod failure.

3. Create the **PersistentVolume** object from the file:

```
$ oc create -f pv1.yaml
```

4. Create a persistent volume claim that maps to the persistent volume you created in the previous step. Create a file, **pvc1.yaml**, with the **PersistentVolumeClaim** object definition:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1 ①
spec:
  accessModes:
    - ReadWriteOnce ②
  resources:
   requests:
     storage: "1Gi" ③
  volumeName: pv1 ④
```

1. A unique name that represents the persistent volume claim.
2. The access mode of the persistent volume claim. With ReadWriteOnce, the volume can be mounted with read and write permissions by a single node.
3. The size of the persistent volume claim.
4. The name of the existing persistent volume.

5. Create the **PersistentVolumeClaim** object from the file:

```
$ oc create -f pvc1.yaml
```

4.11.3.1. Formatting VMware vSphere volumes

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that the volume contains a file system that is specified by the **fsType** parameter value in the **PersistentVolume** (PV) definition. If the device is not formatted with the file system, all data from the device is erased, and the device is automatically formatted with the specified file system.

Because OpenShift Container Platform formats them before the first use, you can use unformatted vSphere volumes as PVs.

4.12. PERSISTENT STORAGE USING LOCAL STORAGE

4.12.1. Persistent storage using local volumes

OpenShift Container Platform can be provisioned with persistent storage by using local volumes. Local persistent volumes allow you to access local storage devices, such as a disk or partition, by using the standard persistent volume claim interface.

Local volumes can be used without manually scheduling pods to nodes because the system is aware of the volume node constraints. However, local volumes are still subject to the availability of the underlying node and are not suitable for all applications.

NOTE

Local volumes can only be used as a statically created persistent volume.

4.12.1.1. Installing the Local Storage Operator

The Local Storage Operator is not installed in OpenShift Container Platform by default. Use the following procedure to install and configure this Operator to enable local volumes in your cluster.

Prerequisites

- Access to the OpenShift Container Platform web console or command-line interface (CLI).

Procedure

1. Create the **openshift-local-storage** project:

```
$ oc adm new-project openshift-local-storage
```

2. Optional: Allow local storage creation on infrastructure nodes.
   You might want to use the Local Storage Operator to create volumes on infrastructure nodes in support of components such as logging and monitoring.

   You must adjust the default node selector so that the Local Storage Operator includes the infrastructure nodes, and not just worker nodes.

   To block the Local Storage Operator from inheriting the cluster-wide default selector, enter the following command:

   ```
   $ oc annotate namespace openshift-local-storage openshift.io/node-selector=''
   ```

3. Optional: Allow local storage to run on the management pool of CPUs in single-node deployment.
   Use the Local Storage Operator in single-node deployments and allow the use of CPUs that belong to the **management** pool. Perform this step on single-node installations that use management workload partitioning.

   To allow Local Storage Operator to run on the management CPU pool, run following commands:

   ```
   $ oc annotate namespace openshift-local-storage
   workload.openshift.io/allowed='management'
   ```

From the UI

To install the Local Storage Operator from the web console, follow these steps:

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators** → **OperatorHub**.
3. Type **Local Storage** into the filter box to locate the Local Storage Operator.
4. Click **Install**.
5. On the **Install Operator** page, select **A specific namespace on the cluster**. Select **openshift-local-storage** from the drop-down menu.
6. Adjust the values for **Update Channel** and **Approval Strategy** to the values that you want.
7. Click **Install**.

Once finished, the Local Storage Operator will be listed in the **Installed Operators** section of the web console.

From the CLI

1. Install the Local Storage Operator from the CLI.
   a. Create an object YAML file to define an Operator group and subscription for the Local Storage Operator, such as **openshift-local-storage.yaml**:

Example openshift-local-storage.yaml

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: local-operator-group
  namespace: openshift-local-storage
spec:
  targetNamespaces:
    - openshift-local-storage
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: stable
  installPlanApproval: Automatic 1
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

1 The user approval policy for an install plan.

2. Create the Local Storage Operator object by entering the following command:

   ```
   $ oc apply -f openshift-local-storage.yaml
   ```

   At this point, the Operator Lifecycle Manager (OLM) is now aware of the Local Storage Operator. A ClusterServiceVersion (CSV) for the Operator should appear in the target namespace, and APIs provided by the Operator should be available for creation.

3. Verify local storage installation by checking that all pods and the Local Storage Operator have been created:

   a. Check that all the required pods have been created:

   ```
   $ oc -n openshift-local-storage get pods
   ```

   ### Example output

   ```
   NAME                                      READY   STATUS    RESTARTS   AGE
   local-storage-operator-746bf599c9-vlt5t   1/1     Running   0          19m
   ```

   b. Check the ClusterServiceVersion (CSV) YAML manifest to see that the Local Storage Operator is available in the **openshift-local-storage** project:

   ```
   $ oc get csvs -n openshift-local-storage
   ```

   ### Example output

```
NAME                                     DISPLAY        VERSION                 REPLACES   PHASE
local-storage-operator.4.2.26-202003230335   Local Storage   4.2.26-202003230335
Succeeded
```

After all checks have passed, the Local Storage Operator is installed successfully.

4.12.1.2. Provisioning local volumes by using the Local Storage Operator

Local volumes cannot be created by dynamic provisioning. Instead, persistent volumes can be created by the Local Storage Operator. The local volume provisioner looks for any file system or block volume devices at the paths specified in the defined resource.

Prerequisites

- The Local Storage Operator is installed.
- You have a local disk that meets the following conditions:
  - It is attached to a node.
  - It is not mounted.
  - It does not contain partitions.

Procedure

1. Create the local volume resource. This resource must define the nodes and paths to the local volumes.

   NOTE

   Do not use different storage class names for the same device. Doing so will create multiple persistent volumes (PVs).

   Example: Filesystem

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" 1
spec:
  nodeSelector: 2
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-140-183
          - ip-10-0-158-139
          - ip-10-0-164-33
  storageClassDevices:
    - storageClassName: "local-sc" 3
      volumeMode: Filesystem 4
      fsType: xfs 5
      devicePaths: 6
        - /path/to/device 7
```

1. The namespace where the Local Storage Operator is installed.
2. Optional: A node selector containing a list of nodes where the local storage volumes are attached. This example uses the node hostnames, obtained from **oc get node**. If a value is not defined, then the Local Storage Operator will attempt to find matching disks on all available nodes.
3. The name of the storage class to use when creating persistent volume objects. The Local Storage Operator automatically creates the storage class if it does not exist. Be sure to use a storage class that uniquely identifies this set of local volumes.
4. The volume mode, either **Filesystem** or **Block**, that defines the type of local volumes.

NOTE

A raw block volume (**volumeMode: Block**) is not formatted with a file system. Use this mode only if any application running on the pod can use raw block devices.

5. The file system that is created when the local volume is mounted for the first time.
6. The path containing a list of local storage devices to choose from.
7. Replace this value with your actual local disks filepath to the **LocalVolume** resource **by-id**, such as **/dev/disk/by-id/wwn**. PVs are created for these local disks when the provisioner is deployed successfully.

NOTE

If you are running OpenShift Container Platform with RHEL KVM, you must assign a serial number to your VM disk. Otherwise, the VM disk can not be identified after reboot. You can use the **virsh edit <VM>** command to add the **<serial>mydisk</serial>** definition.

Example: Block

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" 1
spec:
  nodeSelector: 2
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-136-143
        - ip-10-0-140-255
        - ip-10-0-144-180
storageClassDevices:
  - storageClassName: "localblock-sc" ③
    volumeMode: Block ④
    devicePaths: ⑤
      - /path/to/device ⑥
```

① The namespace where the Local Storage Operator is installed.

② Optional: A node selector containing a list of nodes where the local storage volumes are attached. This example uses the node hostnames, obtained from **oc get node**. If a value is not defined, then the Local Storage Operator will attempt to find matching disks on all available nodes.

③ The name of the storage class to use when creating persistent volume objects.

④ The volume mode, either **Filesystem** or **Block**, that defines the type of local volumes.

⑤ The path containing a list of local storage devices to choose from.

⑥ Replace this value with your actual local disks filepath to the **LocalVolume** resource **by-id**, such as **dev/disk/by-id/wwn**. PVs are created for these local disks when the provisioner is deployed successfully.

NOTE

If you are running OpenShift Container Platform with RHEL KVM, you must assign a serial number to your VM disk. Otherwise, the VM disk can not be identified after reboot. You can use the **virsh edit <VM>** command to add the **<serial>mydisk</serial>** definition.

2. Create the local volume resource in your OpenShift Container Platform cluster. Specify the file you just created:

```
$ oc create -f <local-volume>.yaml
```

3. Verify that the provisioner was created and that the corresponding daemon sets were created:

```
$ oc get all -n openshift-local-storage
```

Example output

```
NAME                                          READY   STATUS    RESTARTS   AGE
pod/diskmaker-manager-9wzms                   1/1     Running   0          5m43s
pod/diskmaker-manager-jgvjp                   1/1     Running   0          5m43s
pod/diskmaker-manager-tbdsj                   1/1     Running   0          5m43s
pod/local-storage-operator-7db4bd9f79-t6k87   1/1     Running   0          14m

NAME                                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/local-storage-operator-metrics   ClusterIP   172.30.135.36   <none>
8383/TCP,8686/TCP   14m
```

```
NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR   AGE
daemonset.apps/diskmaker-manager   3        3        3      3           3          <none>
5m43s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/local-storage-operator   1/1     1           1           14m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/local-storage-operator-7db4bd9f79   1        1        1       14m
```

Note the desired and current number of daemon set processes. A desired count of **0** indicates that the label selectors were invalid.

4. Verify that the persistent volumes were created:

```
$ oc get pv
```

Example output

```
NAME              CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM
STORAGECLASS   REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available           local-sc            88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available           local-sc
82m
local-pv-3fa1c73    100Gi      RWO            Delete           Available           local-sc            48m
```

IMPORTANT

Editing the **LocalVolume** object does not change the **fsType** or **volumeMode** of existing persistent volumes because doing so might result in a destructive operation.

4.12.1.3. Provisioning local volumes without the Local Storage Operator

Local volumes cannot be created by dynamic provisioning. Instead, persistent volumes can be created by defining the persistent volume (PV) in an object definition. The local volume provisioner looks for any file system or block volume devices at the paths specified in the defined resource.

IMPORTANT

Manual provisioning of PVs includes the risk of potential data leaks across PV reuse when PVCs are deleted. The Local Storage Operator is recommended for automating the life cycle of devices when provisioning local PVs.

Prerequisites

- Local disks are attached to the OpenShift Container Platform nodes.

Procedure

1. Define the PV. Create a file, such as **example-pv-filesystem.yaml** or **example-pv-block.yaml**, with the **PersistentVolume** object definition. This resource must define the nodes and paths to the local volumes.

   NOTE

   Do not use different storage class names for the same device. Doing so will create multiple PVs.

   ### example-pv-filesystem.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv-filesystem
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem ❶
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage ❷
  local:
    path: /dev/xvdf ❸
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

❶ The volume mode, either **Filesystem** or **Block**, that defines the type of PVs.

❷ The name of the storage class to use when creating PV resources. Use a storage class that uniquely identifies this set of PVs.

❸ The path containing a list of local storage devices to choose from, or a directory. You can only specify a directory with **Filesystem volumeMode**.

   NOTE

   A raw block volume (**volumeMode: block**) is not formatted with a file system. Use this mode only if any application running on the pod can use raw block devices.

   ### example-pv-block.yaml

```
apiVersion: v1
kind: PersistentVolume
```

```
metadata:
  name: example-pv-block
spec:
  capacity:
    storage: 100Gi
  volumeMode: Block 1
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage 2
  local:
    path: /dev/xvdf 3
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

1. The volume mode, either **Filesystem** or **Block**, that defines the type of PVs.
2. The name of the storage class to use when creating PV resources. Be sure to use a storage class that uniquely identifies this set of PVs.
3. The path containing a list of local storage devices to choose from.

2. Create the PV resource in your OpenShift Container Platform cluster. Specify the file you just created:

```
$ oc create -f <example-pv>.yaml
```

3. Verify that the local PV was created:

```
$ oc get pv
```

Example output

```
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS    REASON   AGE
example-pv-filesystem   100Gi      RWO          Delete          Available                     local-
storage         3m47s
example-pv1             1Gi      RWO          Delete          Bound       local-storage/pvc1   local-
storage         12h
example-pv2             1Gi      RWO          Delete          Bound       local-storage/pvc2   local-
storage         12h
example-pv3             1Gi      RWO          Delete          Bound       local-storage/pvc3   local-
storage         12h
```

4.12.1.4. Creating the local volume persistent volume claim

Local volumes must be statically created as a persistent volume claim (PVC) to be accessed by the pod.

Prerequisites

- Persistent volumes have been created using the local volume provisioner.

Procedure

1. Create the PVC using the corresponding storage class:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: local-pvc-name ①
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem ②
  resources:
    requests:
      storage: 100Gi ③
  storageClassName: local-sc ④
```

① Name of the PVC.

② The type of the PVC. Defaults to **Filesystem**.

③ The amount of storage available to the PVC.

④ Name of the storage class required by the claim.

2. Create the PVC in the OpenShift Container Platform cluster, specifying the file you just created:

```
$ oc create -f <local-pvc>.yaml
```

4.12.1.5. Attach the local claim

After a local volume has been mapped to a persistent volume claim it can be specified inside of a resource.

Prerequisites

- A persistent volume claim exists in the same namespace.

Procedure

1. Include the defined claim in the resource spec. The following example declares the persistent volume claim inside a pod:

```
apiVersion: v1
kind: Pod
spec:
  ...
  containers:
    volumeMounts:
    - name: local-disks 1
      mountPath: /data 2
  volumes:
  - name: localpvc
    persistentVolumeClaim:
      claimName: local-pvc-name 3
```

1. The name of the volume to mount.

2. The path inside the pod where the volume is mounted. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host **/dev/pts** files. It is safe to mount the host by using **/host**.

3. The name of the existing persistent volume claim to use.

2. Create the resource in the OpenShift Container Platform cluster, specifying the file you just created:

```
$ oc create -f <local-pod>.yaml
```

4.12.1.6. Automating discovery and provisioning for local storage devices

The Local Storage Operator automates local storage discovery and provisioning. With this feature, you can simplify installation when dynamic provisioning is not available during deployment, such as with bare metal, VMware, or AWS store instances with attached devices.

> **IMPORTANT**
>
> Automatic discovery and provisioning is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Use the following procedure to automatically discover local devices, and to automatically provision local volumes for selected devices.

WARNING

Use the **LocalVolumeSet** object with caution. When you automatically provision persistent volumes (PVs) from local disks, the local PVs might claim all devices that match. If you are using a **LocalVolumeSet** object, make sure the Local Storage Operator is the only entity managing local devices on the node. Creating multiple instances of a **LocalVolumeSet** that target a node more than once is not supported.

Prerequisites

- You have cluster administrator permissions.
- You have installed the Local Storage Operator.
- You have attached local disks to OpenShift Container Platform nodes.
- You have access to the OpenShift Container Platform web console and the **oc** command-line interface (CLI).

Procedure

1. To enable automatic discovery of local devices from the web console:

    a. In the *Administrator* perspective, navigate to **Operators → Installed Operators** and click on the **Local Volume Discovery** tab.

    b. Click **Create Local Volume Discovery**.

    c. Select either **All nodes** or **Select nodes**, depending on whether you want to discover available disks on all or specific nodes.

    NOTE

    Only worker nodes are available, regardless of whether you filter using **All nodes** or **Select nodes**.

    d. Click **Create**.

A local volume discovery instance named **auto-discover-devices** is displayed.

1. To display a continuous list of available devices on a node:

    a. Log in to the OpenShift Container Platform web console.

    b. Navigate to **Compute → Nodes**.

    c. Click the node name that you want to open. The "Node Details" page is displayed.

    d. Select the **Disks** tab to display the list of the selected devices.
    The device list updates continuously as local disks are added or removed. You can filter the devices by name, status, type, model, capacity, and mode.

2. To automatically provision local volumes for the discovered devices from the web console:

   a. Navigate to **Operators → Installed Operators** and select **Local Storage** from the list of Operators.

   b. Select **Local Volume Set → Create Local Volume Set**

   c. Enter a volume set name and a storage class name.

   d. Choose **All nodes** or **Select nodes** to apply filters accordingly.

   NOTE

   Only worker nodes are available, regardless of whether you filter using **All nodes** or **Select nodes**.

   e. Select the disk type, mode, size, and limit you want to apply to the local volume set, and click **Create**.
   A message displays after several minutes, indicating that the "Operator reconciled successfully."

3. Alternatively, to provision local volumes for the discovered devices from the CLI:

   a. Create an object YAML file to define the local volume set, such as **local-volume-set.yaml**, as shown in the following example:

```
apiVersion: local.storage.openshift.io/v1alpha1
kind: LocalVolumeSet
metadata:
  name: example-autodetect
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-0
              - worker-1
  storageClassName: example-storageclass 1
  volumeMode: Filesystem
  fsType: ext4
  maxDeviceCount: 10
  deviceInclusionSpec:
    deviceTypes: 2
      - disk
      - part
    deviceMechanicalProperties:
      - NonRotational
    minSize: 10G
    maxSize: 100G
    models:
      - SAMSUNG
      - Crucial_CT525MX3
    vendors:
      - ATA
      - ST2000LM
```

1 Determines the storage class that is created for persistent volumes that are provisioned from discovered devices. The Local Storage Operator automatically creates the storage class if it does not exist. Be sure to use a storage class that uniquely identifies this set of local volumes.

2 When using the local volume set feature, the Local Storage Operator does not support the use of logical volume management (LVM) devices.

b. Create the local volume set object:

```
$ oc apply -f local-volume-set.yaml
```

c. Verify that the local persistent volumes were dynamically provisioned based on the storage class:

```
$ oc get pv
```

Example output

```
NAME              CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS          REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available           example-
storageclass      88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available           example-
storageclass      82m
local-pv-3fa1c73   100Gi      RWO            Delete           Available           example-
storageclass      48m
```

NOTE

Results are deleted after they are removed from the node. Symlinks must be manually removed.

4.12.1.7. Using tolerations with Local Storage Operator pods

Taints can be applied to nodes to prevent them from running general workloads. To allow the Local Storage Operator to use tainted nodes, you must add tolerations to the **Pod** or **DaemonSet** definition. This allows the created resources to run on these tainted nodes.

You apply tolerations to the Local Storage Operator pod through the **LocalVolume** resource and apply taints to a node through the node specification. A taint on a node instructs the node to repel all pods that do not tolerate the taint. Using a specific taint that is not on other pods ensures that the Local Storage Operator pod can also run on that node.

IMPORTANT

Taints and tolerations consist of a key, value, and effect. As an argument, it is expressed as **key=value:effect**. An operator allows you to leave one of these parameters empty.

Prerequisites

- The Local Storage Operator is installed.
- Local disks are attached to OpenShift Container Platform nodes with a taint.
- Tainted nodes are expected to provision local storage.

Procedure

To configure local volumes for scheduling on tainted nodes:

1. Modify the YAML file that defines the **Pod** and add the **LocalVolume** spec, as shown in the following example:

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage"
spec:
  tolerations:
    - key: localstorage ①
      operator: Equal ②
      value: "localstorage" ③
  storageClassDevices:
      - storageClassName: "localblock-sc"
        volumeMode: Block ④
        devicePaths: ⑤
          - /dev/xvdg
```

① Specify the key that you added to the node.

② Specify the **Equal** operator to require the **key**/**value** parameters to match. If operator is **Exists**, the system checks that the key exists and ignores the value. If operator is **Equal**, then the key and value must match.

③ Specify the value **local** of the tainted node.

④ The volume mode, either **Filesystem** or **Block**, defining the type of the local volumes.

⑤ The path containing a list of local storage devices to choose from.

2. Optional: To create local persistent volumes on only tainted nodes, modify the YAML file and add the **LocalVolume** spec, as shown in the following example:

```
spec:
  tolerations:
    - key: node-role.kubernetes.io/master
      operator: Exists
```

The defined tolerations will be passed to the resulting daemon sets, allowing the diskmaker and provisioner pods to be created for nodes that contain the specified taints.

4.12.1.8. Local Storage Operator Metrics

OpenShift Container Platform provides the following metrics for the Local Storage Operator:

- **lso_discovery_disk_count**: total number of discovered devices on each node
- **lso_lvset_provisioned_PV_count**: total number of PVs created by **LocalVolumeSet** objects
- **lso_lvset_unmatched_disk_count**: total number of disks that Local Storage Operator did not select for provisioning because of mismatching criteria
- **lso_lvset_orphaned_symlink_count**: number of devices with PVs that no longer match **LocalVolumeSet** object criteria
- **lso_lv_orphaned_symlink_count**: number of devices with PVs that no longer match **LocalVolume** object criteria
- **lso_lv_provisioned_PV_count**: total number of provisioned PVs for **LocalVolume**

To use these metrics, be sure to:

- Enable support for monitoring when installing the Local Storage Operator.
- When upgrading to OpenShift Container Platform 4.9 or later, enable metric support manually by adding the **operator-metering=true** label to the namespace.

For more information about metrics, see Managing metrics.

4.12.1.9. Deleting the Local Storage Operator resources

4.12.1.9.1. Removing a local volume or local volume set

Occasionally, local volumes and local volume sets must be deleted. While removing the entry in the resource and deleting the persistent volume is typically enough, if you want to reuse the same device path or have it managed by a different storage class, then additional steps are needed.

NOTE

The following procedure outlines an example for removing a local volume. The same procedure can also be used to remove symlinks for a local volume set custom resource.

Prerequisites

- The persistent volume must be in a **Released** or **Available** state.

WARNING

Deleting a persistent volume that is still in use can result in data loss or corruption.

Procedure

1. Edit the previously created local volume to remove any unwanted disks.
   a. Edit the cluster resource:

      ```
      $ oc edit localvolume <name> -n openshift-local-storage
      ```

   b. Navigate to the lines under **devicePaths**, and delete any representing unwanted disks.
2. Delete any persistent volumes created.

   ```
   $ oc delete pv <pv-name>
   ```

3. Delete any symlinks on the node.

   > **WARNING**
   >
   > The following step involves accessing a node as the root user. Modifying the state of the node beyond the steps in this procedure could result in cluster instability.

   a. Create a debug pod on the node:

      ```
      $ oc debug node/<node-name>
      ```

   b. Change your root directory to **/host**:

      ```
      $ chroot /host
      ```

   c. Navigate to the directory containing the local volume symlinks.

      ```
      $ cd /mnt/openshift-local-storage/<sc-name> 1
      ```

      1 The name of the storage class used to create the local volumes.

   d. Delete the symlink belonging to the removed device.

      ```
      $ rm <symlink>
      ```

4.12.1.9.2. Uninstalling the Local Storage Operator

To uninstall the Local Storage Operator, you must remove the Operator and all created resources in the **openshift-local-storage** project.

> **WARNING**
>
> Uninstalling the Local Storage Operator while local storage PVs are still in use is not recommended. While the PVs will remain after the Operator's removal, there might be indeterminate behavior if the Operator is uninstalled and reinstalled without removing the PVs and local storage resources.

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

1. Delete any local volume resources installed in the project, such as **localvolume**, **localvolumeset**, and **localvolumediscovery**:

   ```
   $ oc delete localvolume --all --all-namespaces
   $ oc delete localvolumeset --all --all-namespaces
   $ oc delete localvolumediscovery --all --all-namespaces
   ```

2. Uninstall the Local Storage Operator from the web console.

   a. Log in to the OpenShift Container Platform web console.

   b. Navigate to **Operators** → **Installed Operators**.

   c. Type **Local Storage** into the filter box to locate the Local Storage Operator.

   d. Click the Options menu ⋮ at the end of the Local Storage Operator.

   e. Click **Uninstall Operator**.

   f. Click **Remove** in the window that appears.

3. The PVs created by the Local Storage Operator will remain in the cluster until deleted. After these volumes are no longer in use, delete them by running the following command:

   ```
   $ oc delete pv <pv-name>
   ```

4. Delete the **openshift-local-storage** project:

   ```
   $ oc delete project openshift-local-storage
   ```

4.12.2. Persistent storage using hostPath

A hostPath volume in an OpenShift Container Platform cluster mounts a file or directory from the host node's filesystem into your pod. Most pods will not need a hostPath volume, but it does offer a quick option for testing should an application require it.

IMPORTANT

The cluster administrator must configure pods to run as privileged. This grants access to pods in the same node.

4.12.2.1. Overview

OpenShift Container Platform supports hostPath mounting for development and testing on a single-node cluster.

In a production cluster, you would not use hostPath. Instead, a cluster administrator would provision a network resource, such as a GCE Persistent Disk volume, an NFS share, or an Amazon EBS volume. Network resources support the use of storage classes to set up dynamic provisioning.

A hostPath volume must be provisioned statically.

IMPORTANT

Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged. It is safe to mount the host by using /**host**. The following example shows the / directory from the host being mounted into the container at /**host**.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-host-mount
spec:
  containers:
  - image: registry.access.redhat.com/ubi9/ubi
    name: test-container
    command: ['sh', '-c', 'sleep 3600']
    volumeMounts:
    - mountPath: /host
      name: host-slash
  volumes:
   - name: host-slash
     hostPath:
       path: /
       type: ''
```

4.12.2.2. Statically provisioning hostPath volumes

A pod that uses a hostPath volume must be referenced by manual (static) provisioning.

Procedure

1. Define the persistent volume (PV). Create a file, **pv.yaml**, with the **PersistentVolume** object definition:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume 1
  labels:
    type: local
spec:
  storageClassName: manual ②
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce ③
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data" ④
```

① The name of the volume. This name is how it is identified by persistent volume claims or pods.

② Used to bind persistent volume claim requests to this persistent volume.

③ The volume can be mounted as **read-write** by a single node.

④ The configuration file specifies that the volume is at **/mnt/data** on the cluster's node. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system. It is safe to mount the host by using **/host**.

2. Create the PV from the file:

```
$ oc create -f pv.yaml
```

3. Define the persistent volume claim (PVC). Create a file, **pvc.yaml**, with the **PersistentVolumeClaim** object definition:

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

4. Create the PVC from the file:

```
$ oc create -f pvc.yaml
```

4.12.2.3. Mounting the hostPath share in a privileged pod

After the persistent volume claim has been created, it can be used inside by an application. The following example demonstrates mounting this share inside of a pod.

Prerequisites

- A persistent volume claim exists that is mapped to the underlying hostPath share.

Procedure

- Create a privileged pod that mounts the existing persistent volume claim:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name 1
spec:
  containers:
    ...
    securityContext:
      privileged: true 2
    volumeMounts:
    - mountPath: /data 3
      name: hostpath-privileged
  ...
  securityContext: {}
  volumes:
    - name: hostpath-privileged
      persistentVolumeClaim:
        claimName: task-pvc-volume 4
```

1 The name of the pod.

2 The pod must run as privileged to access the node's storage.

3 The path to mount the host path share inside the privileged pod. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host **/dev/pts** files. It is safe to mount the host by using **/host**.

4 The name of the **PersistentVolumeClaim** object that has been previously created.

4.12.3. Persistent storage using logical volume manager storage

Logical volume manager storage (LVM Storage) uses the TopoLVM CSI driver to dynamically provision local storage on single-node OpenShift clusters.

LVM Storage creates thin-provisioned volumes using Logical Volume Manager and provides dynamic provisioning of block storage on a limited resources single-node OpenShift cluster.

4.12.3.1. Deploying LVM Storage on single-node OpenShift clusters

You can deploy LVM Storage on a single-node OpenShift bare-metal or user-provisioned infrastructure cluster and configure it to dynamically provision storage for your workloads.

LVM Storage creates a volume group using all the available unused disks and creates a single thin pool with a size of 90% of the volume group. The remaining 10% of the volume group is left free to enable data recovery by expanding the thin pool when required. You might need to manually perform such recovery.

You can use persistent volume claims (PVCs) and volume snapshots provisioned by LVM Storage to request storage and create volume snapshots.

LVM Storage configures a default overprovisioning limit of 10 to take advantage of the thin-provisioning feature. The total size of the volumes and volume snapshots that can be created on the single-node OpenShift clusters is 10 times the size of the thin pool.

You can deploy LVM Storage on single-node OpenShift clusters using one of the following:

- Red Hat Advanced Cluster Management (RHACM)
- OpenShift Container Platform Web Console

4.12.3.1.1. Requirements

Before you begin deploying LVM Storage on single-node OpenShift clusters, ensure that the following requirements are met:

- You have installed Red Hat Advanced Cluster Management (RHACM) on an OpenShift Container Platform cluster.
- Every managed single-node OpenShift cluster has dedicated disks that are used to provision storage.

Before you deploy LVM Storage on single-node OpenShift clusters, be aware of the following limitations:

- You can only create a single instance of the **LVMCluster** custom resource (CR) on an OpenShift Container Platform cluster.
- When a device becomes part of the **LVMCluster** CR, it cannot be removed.

4.12.3.1.2. Limitations

For deploying single-node OpenShift, LVM Storage has the following limitations:

- The total storage size is limited by the size of the underlying Logical Volume Manager (LVM) thin pool and the overprovisioning factor.
- The size of the logical volume depends on the size of the Physical Extent (PE) and the Logical Extent (LE).
  - It is possible to define the size of PE and LE during the physical and logical device creation.
  - The default PE and LE size is 4 MB.
  - If the size of the PE is increased, the maximum size of the LVM is determined by the kernel limits and your disk space.

Table 4.1. Size limits for different architectures using the default PE and LE size

Architecture	RHEL 6	RHEL 7	RHEL 8	RHEL 9
32-bit	16 TB	-	-	-

Architecture	RHEL 6	RHEL 7	RHEL 8	RHEL 9
64-bit	8 EB [1] 100 TB [2]	8 EB [1] 500 TB [2]	8 EB	8 EB

1. Theoretical size.
2. Tested size.

Additional resources

- Red Hat Advanced Cluster Management for Kubernetes: Installing while connected online

4.12.3.1.3. Installing LVM Storage with the CLI

As a cluster administrator, you can install Logical volume manager storage (LVM Storage) by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Create a namespace for the LVM Storage Operator.
   a. Save the following YAML in the **lvms-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
  name: openshift-storage
```

   b. Create the **Namespace** CR:

```
$ oc create -f lvms-namespace.yaml
```

2. Create an Operator group for the LVM Storage Operator.
   a. Save the following YAML in the **lvms-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-storage-operatorgroup
  namespace: openshift-storage
spec:
  targetNamespaces:
  - openshift-storage
```

b. Create the **OperatorGroup** CR:

```
$ oc create -f lvms-operatorgroup.yaml
```

3. Subscribe to the LVM Storage Operator.

a. Save the following YAML in the **lvms-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: lvms
  namespace: openshift-storage
spec:
  installPlanApproval: Automatic
  name: lvms-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** CR:

```
$ oc create -f lvms-sub.yaml
```

4. Create the **LVMCluster** resource:

a. Save the following YAML in the **lvmcluster.yaml** file:

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
 name: my-lvmcluster
 namespace: openshift-storage
spec:
 storage:
   deviceClasses:
   - name: vg1
     deviceSelector:
       paths:
       - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
       - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
       optionalPaths:
       - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
       - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
     thinPoolConfig:
       name: thin-pool-1
       sizePercent: 90
       overprovisionRatio: 10
     nodeSelector:
       nodeSelectorTerms:
       - matchExpressions:
            - key: app
              operator: In
              values:
              - test1
```

b. Create the **LVMCluster** CR:

```
$ oc create -f lvmcluster.yaml
```

5. To verify that the Operator is installed, enter the following command:

```
$ oc get csv -n openshift-storage -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

Example output

```
Name                         Phase
4.13.0-202301261535          Succeeded
```

4.12.3.1.4. Installing LVM Storage with the web console

You can install Logical volume manager storage (LVM Storage) by using the Red Hat OpenShift Container Platform OperatorHub.

Prerequisites

- You have access to the single-node OpenShift cluster.
- You are using an account with the **cluster-admin** and Operator installation permissions.

Procedure

1. Log in to the OpenShift Container Platform Web Console.
2. Click **Operators → OperatorHub**.
3. Scroll or type **LVM Storage** into the **Filter by keyword** box to find LVM Storage.
4. Click **Install**.
5. Set the following options on the **Install Operator** page:
    a. **Update Channel** as **stable-4.14**.
    b. **Installation Mode** as **A specific namespace on the cluster**.
    c. **Installed Namespace** as **Operator recommended namespace openshift-storage**. If the **openshift-storage** namespace does not exist, it is created during the operator installation.
    d. **Approval Strategy** as **Automatic** or **Manual**.
    If you select **Automatic** updates, then the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention.

If you select **Manual** updates, then the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to update the Operator to a newer version.

6. Click **Install**.

Verification steps

- Verify that LVM Storage shows a green tick, indicating successful installation.

4.12.3.1.5. Uninstalling LVM Storage installed using the OpenShift Web Console

You can unstall LVM Storage using the Red Hat OpenShift Container Platform Web Console.

Prerequisites

- You deleted all the applications on the clusters that are using the storage provisioned by LVM Storage.
- You deleted the persistent volume claims (PVCs) and persistent volumes (PVs) provisioned using LVM Storage.
- You deleted all volume snapshots provisioned by LVM Storage.
- You verified that no logical volume resources exist by using the **oc get logicalvolume** command.
- You have access to the single-node OpenShift cluster using an account with **cluster-admin** permissions.

Procedure

1. From the **Operators → Installed Operators** page, scroll to **LVM Storage** or type **LVM Storage** into the **Filter by name** to find and click on it.
2. Click on the **LVMCluster** tab.
3. On the right-hand side of the **LVMCluster** page, select **Delete LVMCluster** from the **Actions** drop-down menu.
4. Click on the **Details** tab.
5. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
6. Select **Remove**. LVM Storage stops running and is completely removed.

4.12.3.1.6. Installing LVM Storage in a disconnected environment

You can install LVM Storage on OpenShift Container Platform 4.14 in a disconnected environment. All sections referenced in this procedure are linked in *Additional resources*.

Prerequisites

- You read the *About disconnected installation mirroring* section.

- You have access to the OpenShift Container Platform image repository.
- You created a mirror registry.

Procedure

1. Follow the steps in the *Creating the image set configuration* procedure. To create an **ImageSetConfiguration** resource for LVM Storage, you can use the following example YAML file:

Example ImageSetConfiguration file for LVM Storage

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
archiveSize: 4 ①
storageConfig: ②
  registry:
    imageURL: example.com/mirror/oc-mirror-metadata ③
    skipTLS: false
mirror:
  platform:
    channels:
    - name: stable-4.14 ④
      type: ocp
    graph: true ⑤
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.14 ⑥
    packages:
    - name: lvms-operator ⑦
      channels:
      - name: stable ⑧
  additionalImages:
  - name: registry.redhat.io/ubi9/ubi:latest ⑨
  helm: {}
```

① Add **archiveSize** to set the maximum size, in GiB, of each file within the image set.

② Set the back-end location to save the image set metadata to. This location can be a registry or local directory. It is required to specify **storageConfig** values, unless you are using the Technology Preview OCI feature.

③ Set the registry URL for the storage backend.

④ Set the channel to retrieve the OpenShift Container Platform images from.

⑤ Add **graph: true** to generate the OpenShift Update Service (OSUS) graph image to allow for an improved cluster update experience when using the web console. For more information, see *About the OpenShift Update Service* .

⑥ Set the Operator catalog to retrieve the OpenShift Container Platform images from.

⑦ Specify only certain Operator packages to include in the image set. Remove this field to retrieve all packages in the catalog.

8 Specify only certain channels of the Operator packages to include in the image set. You must always include the default channel for the Operator package even if you do not use

9 Specify any additional images to include in image set.

2. Follow the procedure in the *Mirroring an image set to a mirror registry* section.

3. Follow the procedure in the *Configuring image registry repository mirroring* section.

Additional resources

- About disconnected installation mirroring
- Creating a mirror registry with mirror registry for Red Hat OpenShift
- Mirroring the OpenShift Container Platform image repository
- Creating the image set configuration
- Mirroring an image set to a mirror registry
- Configuring image registry repository mirroring

4.12.3.1.7. Installing LVM Storage using RHACM

LVM Storage is deployed on single-node OpenShift clusters using Red Hat Advanced Cluster Management (RHACM). You create a **Policy** object on RHACM that deploys and configures the Operator when it is applied to managed clusters which match the selector specified in the **PlacementRule** resource. The policy is also applied to clusters that are imported later and satisfy the placement rule.

Prerequisites

- Access to the RHACM cluster using an account with **cluster-admin** and Operator installation permissions.
- Dedicated disks on each single-node OpenShift cluster to be used by LVM Storage.
- The single-node OpenShift cluster needs to be managed by RHACM, either imported or created.

Procedure

1. Log in to the RHACM CLI using your OpenShift Container Platform credentials.

2. Create a namespace in which you will create policies.

```
# oc create ns lvms-policy-ns
```

3. To create a policy, save the following YAML to a file with a name such as **policy-lvms-operator.yaml**:

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-install-lvms
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector: 1
    matchExpressions:
    - key: mykey
      operator: In
      values:
      - myvalue
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-install-lvms
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-install-lvms
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: Policy
  name: install-lvms
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
  name: install-lvms
spec:
  disabled: false
  remediationAction: enforce
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: install-lvms
      spec:
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: v1
            kind: Namespace
            metadata:
              labels:
                openshift.io/cluster-monitoring: "true"
                pod-security.kubernetes.io/enforce: privileged
                pod-security.kubernetes.io/audit: privileged
                pod-security.kubernetes.io/warn: privileged
              name: openshift-storage
        - complianceType: musthave
        objectDefinition:
          apiVersion: operators.coreos.com/v1
          kind: OperatorGroup
          metadata:
            name: openshift-storage-operatorgroup
            namespace: openshift-storage
          spec:
            targetNamespaces:
            - openshift-storage
      - complianceType: musthave
        objectDefinition:
          apiVersion: operators.coreos.com/v1alpha1
          kind: Subscription
          metadata:
            name: lvms
            namespace: openshift-storage
          spec:
            installPlanApproval: Automatic
            name: lvms-operator
            source: redhat-operators
            sourceNamespace: openshift-marketplace
      remediationAction: enforce
      severity: low
- objectDefinition:
    apiVersion: policy.open-cluster-management.io/v1
    kind: ConfigurationPolicy
    metadata:
      name: lvms
    spec:
      object-templates:
         - complianceType: musthave
           objectDefinition:
             apiVersion: lvm.topolvm.io/v1alpha1
             kind: LVMCluster
             metadata:
               name: my-lvmcluster
               namespace: openshift-storage
             spec:
               storage:
                 deviceClasses:
                 - name: vg1
                   default: true
                   deviceSelector: 2
                     paths:
                     - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
                     - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
                     optionalPaths:
                     - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
                     - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
                   thinPoolConfig:
                     name: thin-pool-1
                     sizePercent: 90
                     overprovisionRatio: 10
                   nodeSelector: 3
                     nodeSelectorTerms:
                     - matchExpressions:
```

```
                - key: app
                  operator: In
                  values:
                  - test1
    remediationAction: enforce
    severity: low
```

1. Replace the key and value in **PlacementRule.spec.clusterSelector** to match the labels set on the single-node OpenShift clusters on which you want to install LVM Storage.

2. Optional. To control or restrict the volume group to your preferred devices, you can manually specify the local paths of the devices in the **deviceSelector** section of the **LVMCluster** YAML. The **paths** section refers to devices the **LVMCluster** adds, which means those paths must exist. The **optionalPaths** section refers to devices the **LVMCluster** might add. You must specify at least one of **paths** or **optionalPaths** when specifying the **deviceSelector** section. If you specify **paths**, it is not mandatory to specify **optionalPaths**. If you specify **optionalPaths**, it is not mandatory to specify **paths** but at least one optional path must be present on the node. If you do not specify any paths, it will add all unused devices on the node.

3. To add a node filter, which is a subset of the additional worker nodes, specify the required filter in the **nodeSelector** section. LVM Storage detects and uses the additional worker nodes when the new nodes show up.

IMPORTANT

This **nodeSelector** node filter matching is not the same as the pod label matching.

4. Create the policy in the namespace by running the following command:

```
# oc create -f policy-lvms-operator.yaml -n lvms-policy-ns 1
```

1. The **policy-lvms-operator.yaml** is the name of the file to which the policy is saved.

This creates a **Policy**, a **PlacementRule**, and a **PlacementBinding** object in the **lvms-policy-ns** namespace. The policy creates a **Namespace**, **OperatorGroup**, **Subscription**, and **LVMCluster** resource on the clusters that match the placement rule. This deploys the Operator on the single-node OpenShift clusters which match the selection criteria and configures it to set up the required resources to provision storage. The Operator uses all the disks specified in the **LVMCluster** CR. If no disks are specified, the Operator uses all the unused disks on the single-node OpenShift node.

IMPORTANT

After a device is added to the **LVMCluster**, it cannot be removed.

Additional resources

- Red Hat Advanced Cluster Management for Kubernetes: Installing while connected online
- LVM Storage reference YAML file

4.12.3.1.8. Uninstalling LVM Storage installed using RHACM

To uninstall LVM Storage that you installed using RHACM, you need to delete the RHACM policy that you created for deploying and configuring the Operator.

When you delete the RHACM policy, the resources that the policy has created are not removed. You need to create additional policies to remove the resources.

As the created resources are not removed when you delete the policy, you need to perform the following steps:

1. Remove all the Persistent volume claims (PVCs) and volume snapshots provisioned by LVM Storage.

2. Remove the **LVMCluster** resources to clean up Logical Volume Manager resources created on the disks.

3. Create an additional policy to uninstall the Operator.

Prerequisites

- Ensure that the following are deleted before deleting the policy:
  - All the applications on the managed clusters that are using the storage provisioned by LVM Storage.
  - PVCs and persistent volumes (PVs) provisioned using LVM Storage.
  - All volume snapshots provisioned by LVM Storage.
- Ensure you have access to the RHACM cluster using an account with a **cluster-admin** role.

Procedure

1. In the OpenShift CLI (**oc**), delete the RHACM policy that you created for deploying and configuring LVM Storage on the hub cluster by using the following command:

   ```
   # oc delete -f policy-lvms-operator.yaml -n lvms-policy-ns 1
   ```

   1 The **policy-lvms-operator.yaml** is the name of the file to which the policy was saved.

2. To create a policy for removing the **LVMCluster** resource, save the following YAML to a file with a name such as **lvms-remove-policy.yaml**. This enables the Operator to clean up all Logical Volume Manager resources that it created on the cluster.

   ```
   apiVersion: policy.open-cluster-management.io/v1
   kind: Policy
   metadata:
     name: policy-lvmcluster-delete
     annotations:
       policy.open-cluster-management.io/standards: NIST SP 800-53
       policy.open-cluster-management.io/categories: CM Configuration Management
       policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
   spec:
     remediationAction: enforce
   ```

```
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-lvmcluster-removal
        spec:
          remediationAction: enforce ①
          severity: low
          object-templates:
            - complianceType: mustnothave
              objectDefinition:
                kind: LVMCluster
                apiVersion: lvm.topolvm.io/v1alpha1
                metadata:
                  name: my-lvmcluster
                  namespace: openshift-storage ②
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-lvmcluster-delete
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-policy-lvmcluster-delete
subjects:
  - apiGroup: policy.open-cluster-management.io
    kind: Policy
    name: policy-lvmcluster-delete
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-lvmcluster-delete
spec:
  clusterConditions:
    - status: "True"
      type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - key: mykey
        operator: In
        values:
          - myvalue
```

① The **policy-template spec.remediationAction** is overridden by the preceding parameter value for **spec.remediationAction**.

② This **namespace** field must have the **openshift-storage** value.

3. Set the value of the **PlacementRule.spec.clusterSelector** field to select the clusters from which to uninstall LVM Storage.

4. Create the policy by running the following command:

```
# oc create -f lvms-remove-policy.yaml -n lvms-policy-ns
```

5. To create a policy to check if the **LVMCluster** CR has been removed, save the following YAML to a file with a name such as **check-lvms-remove-policy.yaml**:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-lvmcluster-inform
  annotations:
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
spec:
  remediationAction: inform
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-lvmcluster-removal-inform
        spec:
          remediationAction: inform 1
          severity: low
          object-templates:
            - complianceType: mustnothave
              objectDefinition:
                kind: LVMCluster
                apiVersion: lvm.topolvm.io/v1alpha1
                metadata:
                  name: my-lvmcluster
                  namespace: openshift-storage 2
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-lvmcluster-check
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-policy-lvmcluster-check
subjects:
  - apiGroup: policy.open-cluster-management.io
    kind: Policy
    name: policy-lvmcluster-inform
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-lvmcluster-check
spec:
  clusterConditions:
    - status: "True"
      type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - key: mykey
        operator: In
        values:
          - myvalue
```

1. The **policy-template spec.remediationAction** is overridden by the preceding parameter value for **spec.remediationAction**.

2. The **namespace** field must have the **openshift-storage** value.

6. Create the policy by running the following command:

```
# oc create -f check-lvms-remove-policy.yaml -n lvms-policy-ns
```

7. Check the policy status by running the following command:

```
# oc get policy -n lvms-policy-ns
```

Example output

```
NAME                       REMEDIATION ACTION   COMPLIANCE STATE   AGE
policy-lvmcluster-delete   enforce              Compliant          15m
policy-lvmcluster-inform   inform               Compliant          15m
```

8. After both the policies are compliant, save the following YAML to a file with a name such as **lvms-uninstall-policy.yaml** to create a policy to uninstall LVM Storage.

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-uninstall-lvms
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
    - key: mykey
      operator: In
      values:
      - myvalue
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-uninstall-lvms
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-uninstall-lvms
```

```
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: Policy
  name: uninstall-lvms
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
  name: uninstall-lvms
spec:
  disabled: false
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: uninstall-lvms
      spec:
        object-templates:
        - complianceType: mustnothave
          objectDefinition:
            apiVersion: v1
            kind: Namespace
            metadata:
              name: openshift-storage
        - complianceType: mustnothave
          objectDefinition:
            apiVersion: operators.coreos.com/v1
            kind: OperatorGroup
            metadata:
              name: openshift-storage-operatorgroup
              namespace: openshift-storage
            spec:
              targetNamespaces:
              - openshift-storage
        - complianceType: mustnothave
          objectDefinition:
            apiVersion: operators.coreos.com/v1alpha1
            kind: Subscription
            metadata:
              name: lvms-operator
              namespace: openshift-storage
        remediationAction: enforce
        severity: low
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: policy-remove-lvms-crds
      spec:
        object-templates:
        - complianceType: mustnothave
  objectDefinition:
    apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    metadata:
      name: logicalvolumes.topolvm.io
- complianceType: mustnothave
  objectDefinition:
    apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    metadata:
      name: lvmclusters.lvm.topolvm.io
- complianceType: mustnothave
  objectDefinition:
    apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    metadata:
      name: lvmvolumegroupnodestatuses.lvm.topolvm.io
- complianceType: mustnothave
  objectDefinition:
    apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    metadata:
      name: lvmvolumegroups.lvm.topolvm.io
remediationAction: enforce
severity: high
```

9. Create the policy by running the following command:

```
# oc create -f lvms-uninstall-policy.yaml -ns lvms-policy-ns
```

Additional resources

- LVM Storage reference YAML file

4.12.3.2. Creating a Logical Volume Manager cluster on a single-node OpenShift worker node

You can configure a single-node OpenShift worker node as a Logical Volume Manager cluster. On the control-plane single-node OpenShift node, LVM Storage detects and uses the additional worker nodes when the new nodes become active in the cluster.

NOTE

When you create a Logical Volume Manager cluster, **StorageClass** and **LVMVolumeGroup** resources work together to provide dynamic provisioning of storage. **StorageClass** CRs define the properties of the storage that you can dynamically provision. **LVMVolumeGroup** is a specific type of persistent volume (PV) that is backed by an LVM Volume Group. **LVMVolumeGroup** CRs provide the back-end storage for the persistent volumes that you create.

Perform the following procedure to create a Logical Volume Manager cluster on a single-node OpenShift worker node.

NOTE

You also can perform the same task by using the OpenShift Container Platform web console.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You installed LVM Storage in a single-node OpenShift cluster and have installed a worker node for use in the single-node OpenShift cluster.

Procedure

1. Create the **LVMCluster** custom resource (CR).

   a. Save the following YAML in the **lvmcluster.yaml** file:

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: lvmcluster
spec:
  storage:
    deviceClasses: 1
    - name: vg1
      fstype: ext4 2
      default: true 3
      deviceSelector: 4
        paths:
        - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
        optionalPaths:
        - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
      thinPoolConfig:
        name: thin-pool-1
        sizePercent: 90
        overprovisionRatio: 10
      nodeSelector: 5
        nodeSelectorTerms:
          - matchExpressions:
            - key: app
              operator: In
              values:
              - test1
```

1 To create multiple device storage classes in the cluster, create a YAML array under **deviceClasses** for each required storage class. Configure the local device paths of the disks as an array of values in the **deviceSelector** field. When configuring multiple device classes, you must specify the device path for each device.

2 Set **fstype** to **ext4** or **xfs**. By default, it is set to **xfs** if the setting is not specified.

3 Mandatory: The **LVMCluster** resource must contain a single default storage class. Set **default: false** for secondary device storage classes. If you are upgrading the

4 Optional. To control or restrict the volume group to your preferred devices, you can manually specify the local paths of the devices in the **deviceSelector** section of the **LVMCluster** YAML. The **paths** section refers to devices the **LVMCluster** adds, which means those paths must exist. The **optionalPaths** section refers to devices the **LVMCluster** might add. You must specify at least one of **paths** or **optionalPaths** when specifying the **deviceSelector** section. If you specify **paths**, it is not mandatory to specify **optionalPaths**. If you specify **optionalPaths**, it is not mandatory to specify **paths** but at least one optional path must be present on the node. If you do not specify any paths, it will add all unused devices on the node.

5 Optional: To control what worker nodes the **LVMCluster** CR is applied to, specify a set of node selector labels. The specified labels must be present on the node in order for the **LVMCluster** to be scheduled on that node.

b. Create the **LVMCluster** CR:

```
$ oc create -f lvmcluster.yaml
```

Example output

```
lvmcluster/lvmcluster created
```

The **LVMCluster** resource creates the following system-managed CRs:

LVMVolumeGroup

Tracks individual volume groups across multiple nodes.

LVMVolumeGroupNodeStatus

Tracks the status of the volume groups on a node.

Verification

Verify that the **LVMCluster** resource has created the **StorageClass**, **LVMVolumeGroup**, and **LVMVolumeGroupNodeStatus** CRs.

IMPORTANT

LVMVolumeGroup and **LVMVolumeGroupNodeStatus** are managed by LVM Storage. Do not edit these CRs directly.

1. Check that the **LVMCluster** CR is in a **ready** state by running the following command:

```
$ oc get lvmclusters.lvm.topolvm.io -o jsonpath='{.items[*].status.deviceClassStatuses[*]}'
```

Example output

```
{
    "name": "vg1",
    "nodeStatus": [
        {
```

```
      "devices": [
        "/dev/nvme0n1",
        "/dev/nvme1n1",
        "/dev/nvme2n1"
      ],
      "node": "kube-node",
      "status": "Ready"
    }
  ]
}
```

2. Check that the storage class is created:

```
$ oc get storageclass
```

Example output

```
NAME        PROVISIONER        RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
lvms-vg1      topolvm.io          Delete          WaitForFirstConsumer   true                 31m
```

3. Check that the volume snapshot class is created:

```
$ oc get volumesnapshotclass
```

Example output

```
NAME          DRIVER             DELETIONPOLICY   AGE
lvms-vg1      topolvm.io          Delete           24h
```

4. Check that the **LVMVolumeGroup** resource is created:

```
$ oc get lvmvolumegroup vg1 -o yaml
```

Example output

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  creationTimestamp: "2022-02-02T05:16:42Z"
  generation: 1
  name: vg1
  namespace: lvm-operator-system
  resourceVersion: "17242461"
  uid: 88e8ad7d-1544-41fb-9a8e-12b1a66ab157
spec: {}
```

5. Check that the **LVMVolumeGroupNodeStatus** resource is created:

```
$ oc get lvmvolumegroupnodestatuses.lvm.topolvm.io kube-node -o yaml
```

Example output

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMVolumeGroupNodeStatus
metadata:
  creationTimestamp: "2022-02-02T05:17:59Z"
  generation: 1
  name: kube-node
  namespace: lvm-operator-system
  resourceVersion: "17242882"
  uid: 292de9bb-3a9b-4ee8-946a-9b587986dafd
spec:
  nodeStatus:
    - devices:
        - /dev/nvme0n1
        - /dev/nvme1n1
        - /dev/nvme2n1
      name: vg1
      status: Ready
```

Additional resources

- Adding worker nodes to single-node OpenShift clusters
- LVM Storage reference YAML file

4.12.3.3. Adding a storage class

You can add a storage class to an OpenShift Container Platform cluster. A storage class describes a class of storage in the cluster and how the cluster dynamically provisions the persistent volumes (PVs) when the user specifies the storage class. A storage class describes the type of device classes, the quality-of-service level, the filesystem type, and other details.

Procedure

1. Create a YAML file:

   ```
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
     name: lvm-storageclass
   parameters:
     csi.storage.k8s.io/fstype: ext4
     topolvm.io/device-class: vg1
   provisioner: topolvm.io
   reclaimPolicy: Delete
   allowVolumeExpansion: true
   volumeBindingMode: WaitForFirstConsumer
   ```

   Save the file by using a name similar to the storage class name. For example, **lvm-storageclass.yaml**.

2. Apply the YAML file by using the **oc** command:

   ```
   $ oc apply -f <file_name> 1
   ```

1 Replace **<file_name>** with the name of the YAML file. For example, **lvm-storageclass.yaml**.

The cluster will create the storage class.

3. Verify that the cluster created the storage class by using the following command:

```
$ oc get storageclass <name> 1
```

1 Replace **<name>** with the name of the storage class. For example, **lvm-storageclass**.

Example output

```
NAME             PROVISIONER  RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
lvm-storageclass  topolvm.io   Delete         WaitForFirstConsumer  true                 1s
```

4.12.3.4. Provisioning storage using LVM Storage

You can provision persistent volume claims (PVCs) using the storage class that is created during the Operator installation. You can provision block and file PVCs, however, the storage is allocated only when a pod that uses the PVC is created.

NOTE

LVM Storage provisions PVCs in units of 1 GiB. The requested storage is rounded up to the nearest GiB.

Procedure

1. Identify the **StorageClass** that is created when LVM Storage is deployed.
   The **StorageClass** name is in the format, **lvms-<device-class-name>**. The **device-class-name** is the name of the device class that you provided in the **LVMCluster** of the **Policy** YAML. For example, if the **deviceClass** is called **vg1**, then the **storageClass** name is **lvms-vg1**.

   The **volumeBindingMode** of the storage class is set to **WaitForFirstConsumer**.

2. To create a PVC where the application requires storage, save the following YAML to a file with a name such as **pvc.yaml**.

Example YAML to create a PVC

```
# block pvc
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: lvm-block-1
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 10Gi
  storageClassName: lvms-vg1
---
# file pvc
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: lvm-file-1
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi
  storageClassName: lvms-vg1
```

3. Create the PVC by running the following command:

```
# oc create -f pvc.yaml -ns <application_namespace>
```

The created PVCs remain in **pending** state until you deploy the pods that use them.

4.12.3.5. Monitoring LVM Storage

When you use RHACM to install LVM Storage, you must configure RHACM Observability to monitor all the single-node OpenShift clusters from one place.

4.12.3.5.1. Metrics

You can monitor LVM Storage by viewing the metrics exported by the Operator on the RHACM dashboards and the alerts that are triggered.

- Add the following **topolvm** metrics to the **allow** list:

```
topolvm_thinpool_data_percent
topolvm_thinpool_metadata_percent
topolvm_thinpool_size_bytes
```

NOTE

Metrics are updated every 10 minutes or when there is a change in the thin pool, such as a new logical volume creation.

4.12.3.5.2. Alerts

When the thin pool and volume group are filled up, further operations fail and might lead to data loss. LVM Storage sends the following alerts about the usage of the thin pool and volume group when utilization crosses a certain value:

Alerts for Logical Volume Manager cluster in RHACM

Alert	Description
VolumeGroupUsageAtThresholdNearFull	This alert is triggered when both the volume group and thin pool utilization cross 75% on nodes. Data deletion or volume group expansion is required.
VolumeGroupUsageAtThresholdCritical	This alert is triggered when both the volume group and thin pool utilization cross 85% on nodes. **VolumeGroup** is critically full. Data deletion or volume group expansion is required.
ThinPoolDataUsageAtThresholdNearFull	This alert is triggered when the thin pool data utilization in the volume group crosses 75% on nodes. Data deletion or thin pool expansion is required.
ThinPoolDataUsageAtThresholdCritical	This alert is triggered when the thin pool data utilization in the volume group crosses 85% on nodes. Data deletion or thin pool expansion is required.
ThinPoolMetaDataUsageAtThresholdNearFull	This alert is triggered when the thin pool metadata utilization in the volume group crosses 75% on nodes. Data deletion or thin pool expansion is required.
ThinPoolMetaDataUsageAtThresholdCritical	This alert is triggered when the thin pool metadata utilization in the volume group crosses 85% on nodes. Data deletion or thin pool expansion is required.

Additional resources

- Observability
- Adding custom metrics

4.12.3.6. Scaling storage of single-node OpenShift clusters

The OpenShift Container Platform supports additional worker nodes for single-node OpenShift clusters on bare-metal user-provisioned infrastructure. LVM Storage detects and uses the new additional worker nodes when the nodes show up.

Additional resources

- Adding worker nodes to single-node OpenShift clusters

4.12.3.6.1. Scaling up storage by adding capacity to your single-node OpenShift cluster

To scale the storage capacity of your configured worker nodes on a single-node OpenShift cluster, you can increase the capacity by adding disks.

Prerequisites

- You have additional unused disks on each single-node OpenShift cluster to be used by LVM Storage.

Procedure

1. Log in to OpenShift Container Platform console of the single-node OpenShift cluster.
2. From the **Operators → Installed Operators** page, click on the **LVM Storage Operator** in the **openshift-storage** namespace.
3. Click on the **LVMCluster** tab to list the **LVMCluster** CR created on the cluster.
4. Select **Edit LVMCluster** from the **Actions** drop-down menu.
5. Click on the **YAML** tab.
6. Edit the **LVMCluster** CR YAML to add the new device path in the **deviceSelector** section:

> **NOTE**
>
> In case the **deviceSelector** field is not included during the **LVMCluster** creation, it is not possible to add the **deviceSelector** section to the CR. You need to remove the **LVMCluster** and then create a new CR.

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
spec:
  storage:
    deviceClasses:
    - name: vg1
      default: true
      deviceSelector: 1
        paths:
        - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
        optionalPaths:
        - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
      thinPoolConfig:
        name: thin-pool-1
        sizePercent: 90
        overprovisionRatio: 10
```

1 Optional. To control or restrict the volume group to your preferred devices, you can manually specify the local paths of the devices in the **deviceSelector** section of the **LVMCluster** YAML. The **paths** section refers to devices the **LVMCluster** adds, which means those paths must exist. The **optionalPaths** section refers to devices the **LVMCluster** might add. You must specify at least one of **paths** or **optionalPaths** when specifying the **deviceSelector** section. If you specify **paths**, it is not mandatory to specify **optionalPaths**. If you specify **optionalPaths**, it is not mandatory to specify **paths** but at least one optional path must be present on the node. If you do not specify any paths, it will

add all unused devices on the node.

Additional resources

- LVM Storage reference YAML file

4.12.3.6.2. Scaling up storage by adding capacity to your single-node OpenShift cluster using RHACM

You can scale the storage capacity of your configured worker nodes on a single-node OpenShift cluster using RHACM.

Prerequisites

- You have access to the RHACM cluster using an account with **cluster-admin** privileges.
- You have additional unused devices on each single-node OpenShift cluster that LVM Storage can use.

Procedure

1. Log in to the RHACM CLI using your OpenShift Container Platform credentials.
2. Find the device that you want to add. The device to be added needs to match with the device name and path of the existing devices.
3. To add capacity to the single-node OpenShift cluster, edit the **deviceSelector** section of the existing policy YAML, for example, **policy-lvms-operator.yaml**.

NOTE

In case the **deviceSelector** field is not included during the **LVMCluster** creation, it is not possible to add the **deviceSelector** section to the CR. You need to remove the **LVMCluster** and then recreate it from the new CR.

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-install-lvms
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
    - key: mykey
      operator: In
      values:
      - myvalue
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-install-lvms
placementRef:
  apiGroup: apps.open-cluster-management.io
  kind: PlacementRule
  name: placement-install-lvms
subjects:
- apiGroup: policy.open-cluster-management.io
  kind: Policy
  name: install-lvms
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
  name: install-lvms
spec:
  disabled: false
  remediationAction: enforce
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: install-lvms
      spec:
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: v1
            kind: Namespace
            metadata:
              labels:
                openshift.io/cluster-monitoring: "true"
                pod-security.kubernetes.io/enforce: privileged
                pod-security.kubernetes.io/audit: privileged
                pod-security.kubernetes.io/warn: privileged
              name: openshift-storage
        - complianceType: musthave
          objectDefinition:
            apiVersion: operators.coreos.com/v1
            kind: OperatorGroup
            metadata:
              name: openshift-storage-operatorgroup
              namespace: openshift-storage
            spec:
              targetNamespaces:
              - openshift-storage
        - complianceType: musthave
          objectDefinition:
            apiVersion: operators.coreos.com/v1alpha1
            kind: Subscription
            metadata:
              name: lvms
```

```
                namespace: openshift-storage
              spec:
                installPlanApproval: Automatic
                name: lvms-operator
                source: redhat-operators
                sourceNamespace: openshift-marketplace
          remediationAction: enforce
          severity: low
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: lvms
        spec:
          object-templates:
             - complianceType: musthave
               objectDefinition:
                 apiVersion: lvm.topolvm.io/v1alpha1
                 kind: LVMCluster
                 metadata:
                   name: my-lvmcluster
                   namespace: openshift-storage
                 spec:
                   storage:
                     deviceClasses:
                     - name: vg1
                       default: true
                       deviceSelector: (1)
                         paths:
                         - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
                         - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
                         optionalPaths:
                         - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
                         - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
                       thinPoolConfig:
                         name: thin-pool-1
                         sizePercent: 90
                         overprovisionRatio: 10
                       nodeSelector:
                         nodeSelectorTerms:
                         - matchExpressions:
                             - key: app
                               operator: In
                               values:
                               - test1
          remediationAction: enforce
          severity: low
```

(1) Optional. To control or restrict the volume group to your preferred devices, you can manually specify the local paths of the devices in the **deviceSelector** section of the **LVMCluster** YAML. The **paths** section refers to devices the **LVMCluster** adds, which means those paths must exist. The **optionalPaths** section refers to devices the **LVMCluster** might add. You must specify at least one of **paths** or **optionalPaths** when specifying the **deviceSelector** section. If you specify **paths**, it is not mandatory to specify **optionalPaths**. If you specify **optionalPaths**, it is not mandatory to specify **paths** but at least one optional path must be present on the node. If you do not specify any paths, it will

add all unused devices on the node.

4. Edit the policy by running the following command:

```
# oc edit -f policy-lvms-operator.yaml -ns lvms-policy-ns 1
```

1 The **policy-lvms-operator.yaml** is the name of the existing policy.

This uses the new disk specified in the **LVMCluster** CR to provision storage.

Additional resources

- Red Hat Advanced Cluster Management for Kubernetes: Installing while connected online
- LVM Storage reference YAML file

4.12.3.6.3. Expanding PVCs

To leverage the new storage after adding additional capacity, you can expand existing persistent volume claims (PVCs) with LVM Storage.

Prerequisites

- Dynamic provisioning is used.
- The controlling **StorageClass** object has **allowVolumeExpansion** set to **true**.

Procedure

1. Modify the **.spec.resources.requests.storage** field in the desired PVC resource to the new size by running the following command:

```
oc patch <pvc_name> -n <application_namespace> -p '{ "spec": { "resources": { "requests": {
"storage": "<desired_size>" }}}}'
```

2. Watch the **status.conditions** field of the PVC to see if the resize has completed. OpenShift Container Platform adds the **Resizing** condition to the PVC during expansion, which is removed after the expansion completes.

Additional resources

- Scaling up storage by adding capacity to your single-node OpenShift cluster
- Scaling up storage by adding capacity to your single-node OpenShift cluster using RHACM
- Enabling volume expansion support

4.12.3.7. Upgrading LVM Storage on single-node OpenShift clusters

You can upgrade the Logical volume manager storage (LVM Storage) Operator to ensure compatibility with your single-node OpenShift version.

Prerequisites

- You have upgraded your single-node OpenShift cluster.
- You have installed a previous version of the LVM Storage Operator.
- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Update the **Subscription** resource for the LVM Storage Operator by running the following command:

   ```
   $ oc patch subscription lvms-operator -n openshift-storage --type merge --patch '{"spec":
   {"channel":"<update-channel>"}}' 1
   ```

   1 Replace **<update-channel>** with the version of the LVM Storage Operator that you want to install, for example **stable-4.14**.

2. View the upgrade events to check that the installation is complete by running the following command:

   ```
   $ oc get events -n openshift-storage
   ```

   #### Example output

   ```
   ...
   8m13s     Normal   RequirementsUnknown   clusterserviceversion/lvms-operator.v4.14
   requirements not yet checked
   8m11s     Normal   RequirementsNotMet    clusterserviceversion/lvms-operator.v4.14   one
   or more requirements couldn't be found
   7m50s     Normal   AllRequirementsMet    clusterserviceversion/lvms-operator.v4.14   all
   requirements found, attempting install
   7m50s     Normal   InstallSucceeded      clusterserviceversion/lvms-operator.v4.14   waiting
   for install components to report healthy
   7m49s     Normal   InstallWaiting        clusterserviceversion/lvms-operator.v4.14   installing:
   waiting for deployment lvms-operator to become ready: deployment "lvms-operator" waiting
   for 1 outdated replica(s) to be terminated
   7m39s     Normal   InstallSucceeded      clusterserviceversion/lvms-operator.v4.14   install
   strategy completed with no errors
   ...
   ```

Verification

- Verify the version of the LVM Storage Operator by running the following command:

  ```
  $ oc get subscription lvms-operator -n openshift-storage -o jsonpath='{.status.installedCSV}'
  ```

  #### Example output

  ```
  lvms-operator.v4.14
  ```

4.12.3.8. Volume snapshots for single-node OpenShift

You can take volume snapshots of persistent volumes (PVs) that are provisioned by LVM Storage. You can also create volume snapshots of the cloned volumes. Volume snapshots help you to do the following:

- Back up your application data.

> IMPORTANT
>
> Volume snapshots are located on the same devices as the original data. To use the volume snapshots as backups, you need to move the snapshots to a secure location. You can use OpenShift API for Data Protection backup and restore solutions.

- Revert to a state at which the volume snapshot was taken.

Additional resources

- OADP features

4.12.3.8.1. Creating volume snapshots in single-node OpenShift

You can create volume snapshots based on the available capacity of the thin pool and the overprovisioning limits. LVM Storage creates a **VolumeSnapshotClass** with the **lvms-<deviceclass-name>** name.

Prerequisites

- You ensured that the persistent volume claim (PVC) is in **Bound** state. This is required for a consistent snapshot.

- You stopped all the I/O to the PVC before taking the snapshot.

Procedure

1. Log in to the single-node OpenShift for which you need to run the **oc** command.

2. Save the following YAML to a file with a name such as **lvms-vol-snapshot.yaml**.

   Example YAML to create a volume snapshot

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: lvm-block-1-snap
spec:
  volumeSnapshotClassName: lvms-vg1
  source:
    persistentVolumeClaimName: lvm-block-1
```

3. Create the snapshot by running the following command in the same namespace as the PVC:

```
# oc create -f lvms-vol-snapshot.yaml
```

A read-only copy of the PVC is created as a volume snapshot.

4.12.3.8.2. Restoring volume snapshots in single-node OpenShift

When you restore a volume snapshot, a new persistent volume claim (PVC) is created. The restored PVC is independent of the volume snapshot and the source PVC.

Prerequisites

- The storage class must be the same as that of the source PVC.
- The size of the requested PVC must be the same as that of the source volume of the snapshot.

> **IMPORTANT**
>
> A snapshot must be restored to a PVC of the same size as the source volume of the snapshot. If a larger PVC is required, you can resize the PVC after the snapshot is restored successfully.

Procedure

1. Identify the storage class name of the source PVC and volume snapshot name.
2. Save the following YAML to a file with a name such as **lvms-vol-restore.yaml** to restore the snapshot.

Example YAML to restore a PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: lvm-block-1-restore
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Block
  Resources:
    Requests:
      storage: 2Gi
  storageClassName: lvms-vg1
  dataSource:
    name: lvm-block-1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

3. Create the policy by running the following command in the same namespace as the snapshot:

```
# oc create -f lvms-vol-restore.yaml
```

4.12.3.8.3. Deleting volume snapshots in single-node OpenShift

You can delete volume snapshots resources and persistent volume claims (PVCs).

Procedure

1. Delete the volume snapshot resource by running the following command:

```
# oc delete volumesnapshot <volume_snapshot_name> -n <namespace>
```

> **NOTE**
>
> When you delete a persistent volume claim (PVC), the snapshots of the PVC are not deleted.

2. To delete the restored volume snapshot, delete the PVC that was created to restore the volume snapshot by running the following command:

```
# oc delete pvc <pvc_name> -n <namespace>
```

4.12.3.9. Volume cloning for single-node OpenShift

A clone is a duplicate of an existing storage volume that can be used like any standard volume.

4.12.3.9.1. Creating volume clones in single-node OpenShift

You create a clone of a volume to make a point-in-time copy of the data. A persistent volume claim (PVC) cannot be cloned with a different size.

> **IMPORTANT**
>
> The cloned PVC has write access.

Prerequisites

- You ensured that the PVC is in **Bound** state. This is required for a consistent snapshot.
- You ensured that the **StorageClass** is the same as that of the source PVC.

Procedure

1. Identify the storage class of the source PVC.
2. To create a volume clone, save the following YAML to a file with a name such as **lvms-vol-clone.yaml**:

Example YAML to clone a volume

```
apiVersion: v1
kind: PersistentVolumeClaim
Metadata:
  name: lvm-block-1-clone
Spec:
  storageClassName: lvms-vg1
  dataSource:
    name: lvm-block-1
    kind: PersistentVolumeClaim
  accessModes:
   - ReadWriteOnce
```

```
volumeMode: Block
Resources:
  Requests:
    storage: 2Gi
```

3. Create the policy in the same namespace as the source PVC by running the following command:

```
# oc create -f lvms-vol-clone.yaml
```

4.12.3.9.2. Deleting cloned volumes in single-node OpenShift

You can delete cloned volumes.

Procedure

- To delete the cloned volume, delete the cloned PVC by running the following command:

```
# oc delete pvc <clone_pvc_name> -n <namespace>
```

4.12.3.10. Downloading log files and diagnostic information using must-gather

When LVM Storage is unable to automatically resolve a problem, use the must-gather tool to collect the log files and diagnostic information so that you or the Red Hat Support can review the problem and determine a solution.

- Run the must-gather command from the client connected to LVM Storage cluster by running the following command:

```
$ oc adm must-gather --image=registry.redhat.io/lvms4/lvms-must-gather-rhel9:v4.14 --dest-dir=<directory-name>
```

Additional resources

- About the must-gather tool

4.12.3.11. LVM Storage reference YAML file

The sample **LVMCluster** custom resource (CR) describes all the fields in the YAML file.

Example LVMCluster CR

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
spec:
  tolerations:
  - effect: NoSchedule
    key: xyz
    operator: Equal
    value: "true"
  storage:
    deviceClasses:    1
  - name: vg1 2
   default: true
   nodeSelector: 3
    nodeSelectorTerms: 4
    - matchExpressions:
     - key: mykey
      operator: In
      values:
      - ssd
   deviceSelector: 5
    paths:
    - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
    - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
    optionalPaths:
    - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
    - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
   thinPoolConfig: 6
    name: thin-pool-1 7
    sizePercent: 90 8
    overprovisionRatio: 10 9
status:
  deviceClassStatuses: 10
  - name: vg1
   nodeStatus: 11
   - devices: 12
    - /dev/nvme0n1
    - /dev/nvme1n1
    - /dev/nvme2n1
    node: my-node.example.com 13
    status: Ready 14
  ready: true 15
  state: Ready 16
```

1 The LVM volume groups to be created on the cluster. Currently, only a single **deviceClass** is supported.

2 The name of the LVM volume group to be created on the nodes.

3 The nodes on which to create the LVM volume group. If the field is empty, all nodes are considered.

4 A list of node selector requirements.

5 A list of device paths which is used to create the LVM volume group. If this field is empty, all unused disks on the node will be used.

6 The LVM thin pool configuration.

7 The name of the thin pool to be created in the LVM volume group.

8 The percentage of remaining space in the LVM volume group that should be used for creating the thin pool.

9 The factor by which additional storage can be provisioned compared to the available storage in the thin pool.

10 The status of the **deviceClass**.

11 The status of the LVM volume group on each node.

12 The list of devices used to create the LVM volume group.

13 The node on which the **deviceClass** was created.

14 The status of the LVM volume group on the node.

15 This field is deprecated.

16 The status of the **LVMCluster**.

4.12.4. Troubleshooting local persistent storage using LVMS

Because OpenShift Container Platform does not scope a persistent volume (PV) to a single project, it can be shared across the cluster and claimed by any project using a persistent volume claim (PVC). This can lead to a number of issues that require troubleshooting.

4.12.4.1. Investigating a PVC stuck in the Pending state

A persistent volume claim (PVC) can get stuck in a **Pending** state for a number of reasons. For example:

- Insufficient computing resources
- Network problems
- Mismatched storage class or node selector
- No available volumes
- The node with the persistent volume (PV) is in a **Not Ready** state

Identify the cause by using the **oc describe** command to review details about the stuck PVC.

Procedure

1. Retrieve the list of PVCs by running the following command:

```
$ oc get pvc
```

Example output

```
NAME        STATUS    VOLUME    CAPACITY    ACCESS MODES    STORAGECLASS    AGE
lvms-test   Pending                                lvms-vg1      11s
```

2. Inspect the events associated with a PVC stuck in the **Pending** state by running the following command:

```
$ oc describe pvc <pvc_name> 1
```

1 Replace **<pvc_name>** with the name of the PVC. For example, **lvms-vg1**.

Example output

```
Type     Reason              Age               From                      Message
----     ------              ----              ----                      -------
Warning  ProvisioningFailed  4s (x2 over 17s)  persistentvolume-controller
storageclass.storage.k8s.io "lvms-vg1" not found
```

4.12.4.2. Recovering from missing LVMS or Operator components

If you encounter a storage class "not found" error, check the **LVMCluster** resource and ensure that all the logical volume manager storage (LVMS) pods are running. You can create an **LVMCluster** resource if it does not exist.

Procedure

1. Verify the presence of the LVMCluster resource by running the following command:

```
$ oc get lvmcluster -n openshift-storage
```

Example output

```
NAME            AGE
my-lvmcluster   65m
```

2. If the cluster doesn’t have an **LVMCluster** resource, create one by running the following command:

```
$ oc create -n openshift-storage -f <custom_resource> 1
```

1 Replace **<custom_resource>** with a custom resource URL or file tailored to your requirements.

Example custom resource

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
spec:
  storage:
    deviceClasses:
    - name: vg1
      default: true
      thinPoolConfig:
        name: thin-pool-1
        sizePercent: 90
        overprovisionRatio: 10
```

3. Check that all the pods from LVMS are in the **Running** state in the **openshift-storage** namespace by running the following command:

```
$ oc get pods -n openshift-storage
```

Example output

```
NAME                                  READY   STATUS    RESTARTS     AGE
lvms-operator-7b9fb858cb-6nsml        3/3     Running   0            70m
topolvm-controller-5dd9cf78b5-7wwr2   5/5     Running   0            66m
topolvm-node-dr26h                    4/4     Running   0            66m
vg-manager-r6zdv                      1/1     Running   0            66m
```

The expected output is one running instance of **lvms-operator** and **vg-manager**. One instance of **topolvm-controller** and **topolvm-node** is expected for each node.

If **topolvm-node** is stuck in the **Init** state, there is a failure to locate an available disk for LVMS to use. To retrieve the information necessary to troubleshoot, review the logs of the **vg-manager** pod by running the following command:

```
$ oc logs -l app.kubernetes.io/component=vg-manager -n openshift-storage
```

4.12.4.3. Recovering from node failure

Sometimes a persistent volume claim (PVC) is stuck in a **Pending** state because a particular node in the cluster has failed. To identify the failed node, you can examine the restart count of the **topolvm-node** pod. An increased restart count indicates potential problems with the underlying node, which may require further investigation and troubleshooting.

Procedure

- Examine the restart count of the **topolvm-node** pod instances by running the following command:

```
$ oc get pods -n openshift-storage
```

Example output

```
NAME                                  READY   STATUS    RESTARTS       AGE
lvms-operator-7b9fb858cb-6nsml        3/3     Running   0              70m
topolvm-controller-5dd9cf78b5-7wwr2   5/5     Running   0              66m
topolvm-node-dr26h                    4/4     Running   0              66m
topolvm-node-54as8                    4/4     Running   0              66m
topolvm-node-78fft                    4/4     Running   17 (8s ago)    66m
vg-manager-r6zdv                      1/1     Running   0              66m
vg-manager-990ut                      1/1     Running   0              66m
vg-manager-an118                      1/1     Running   0              66m
```

After you resolve any issues with the node, you might need to perform the forced cleanup procedure if the PVC is still stuck in a **Pending** state.

Additional resources

- Performing a forced cleanup

4.12.4.4. Recovering from disk failure

If you see a failure message while inspecting the events associated with the persistent volume claim (PVC), there might be a problem with the underlying volume or disk. Disk and volume provisioning issues often result with a generic error first, such as **Failed to provision volume with StorageClass <storage_class_name>**. A second, more specific error message usually follows.

Procedure

1. Inspect the events associated with a PVC by running the following command:

   ```
   $ oc describe pvc <pvc_name> 1
   ```

   1 Replace **<pvc_name>** with the name of the PVC. Here are some examples of disk or volume failure error messages and their causes:

   - **Failed to check volume existence:** Indicates a problem in verifying whether the volume already exists. Volume verification failure can be caused by network connectivity problems or other failures.
   - **Failed to bind volume:** Failure to bind a volume can happen if the persistent volume (PV) that is available does not match the requirements of the PVC.
   - **FailedMount or FailedUnMount:** This error indicates problems when trying to mount the volume to a node or unmount a volume from a node. If the disk has failed, this error might appear when a pod tries to use the PVC.
   - **Volume is already exclusively attached to one node and can't be attached to another:** This error can appear with storage solutions that do not support **ReadWriteMany** access modes.

2. Establish a direct connection to the host where the problem is occurring.

3. Resolve the disk issue.

After you have resolved the issue with the disk, you might need to perform the forced cleanup procedure if failure messages persist or reoccur.

Additional resources

- Performing a forced cleanup

4.12.4.5. Performing a forced cleanup

If disk- or node-related problems persist after you complete the troubleshooting procedures, it might be necessary to perform a forced cleanup procedure. A forced cleanup is used to comprehensively address persistent issues and ensure the proper functioning of the LVMS.

Prerequisites

1. All of the persistent volume claims (PVCs) created using the logical volume manager storage (LVMS) driver have been removed.

2. The pods using those PVCs have been stopped.

Procedure

1. Switch to the **openshift-storage** namespace by running the following command:

   ```
   $ oc project openshift-storage
   ```

2. Ensure there is no **Logical Volume** custom resource (CR) remaining by running the following command:

   ```
   $ oc get logicalvolume
   ```

   #### Example output

   ```
   No resources found
   ```

   a. If there are any **LogicalVolume** CRs remaining, remove their finalizers by running the following command:

      ```
      $ oc patch logicalvolume <name> -p '{"metadata":{"finalizers":[]}}' --type=merge ❶
      ```

      ❶ Replace **<name>** with the name of the CR.

   b. After removing their finalizers, delete the CRs by running the following command:

      ```
      $ oc delete logicalvolume <name> ❶
      ```

      ❶ Replace **<name>** with the name of the CR.

3. Make sure there are no **LVMVolumeGroup** CRs left by running the following command:

   ```
   $ oc get lvmvolumegroup
   ```

   #### Example output

   ```
   No resources found
   ```

   a. If there are any **LVMVolumeGroup** CRs left, remove their finalizers by running the following command:

      ```
      $ oc patch lvmvolumegroup <name> -p '{"metadata":{"finalizers":[]}}' --type=merge ❶
      ```

      ❶ Replace **<name>** with the name of the CR.

   b. After removing their finalizers, delete the CRs by running the following command:

      ```
      $ oc delete lvmvolumegroup <name> ❶
      ```

      ❶ Replace **<name>** with the name of the CR.

4. Remove any **LVMVolumeGroupNodeStatus** CRs by running the following command:

   ```
   $ oc delete lvmvolumegroupnodestatus --all
   ```

5. Remove the **LVMCluster** CR by running the following command:

   ```
   $ oc delete lvmcluster --all
   ```

CHAPTER 5. USING CONTAINER STORAGE INTERFACE (CSI)

5.1. CONFIGURING CSI VOLUMES

The Container Storage Interface (CSI) allows OpenShift Container Platform to consume storage from storage back ends that implement the CSI interface as persistent storage.

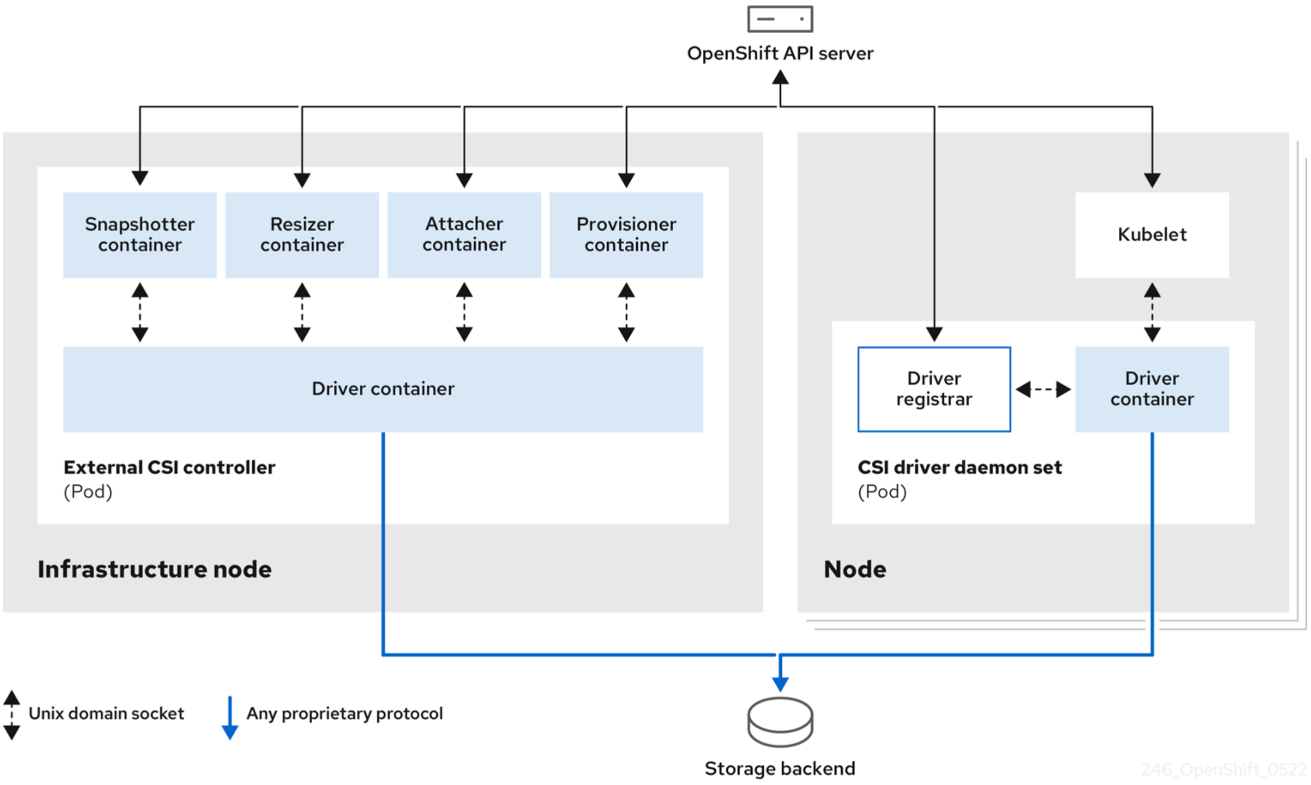
NOTE

OpenShift Container Platform 4.14 supports version 1.6.0 of the CSI specification.

5.1.1. CSI architecture

CSI drivers are typically shipped as container images. These containers are not aware of OpenShift Container Platform where they run. To use CSI-compatible storage back end in OpenShift Container Platform, the cluster administrator must deploy several components that serve as a bridge between OpenShift Container Platform and the storage driver.

The following diagram provides a high-level overview about the components running in pods in the OpenShift Container Platform cluster.



It is possible to run multiple CSI drivers for different storage back ends. Each driver needs its own external controllers deployment and daemon set with the driver and CSI registrar.

5.1.1.1. External CSI controllers

External CSI controllers is a deployment that deploys one or more pods with five containers:

- The snapshotter container watches **VolumeSnapshot** and **VolumeSnapshotContent** objects and is responsible for the creation and deletion of **VolumeSnapshotContent** object.

- The resizer container is a sidecar container that watches for **PersistentVolumeClaim** updates and triggers **ControllerExpandVolume** operations against a CSI endpoint if you request more storage on **PersistentVolumeClaim** object.

- An external CSI attacher container translates **attach** and **detach** calls from OpenShift Container Platform to respective **ControllerPublish** and **ControllerUnpublish** calls to the CSI driver.

- An external CSI provisioner container that translates **provision** and **delete** calls from OpenShift Container Platform to respective **CreateVolume** and **DeleteVolume** calls to the CSI driver.

- A CSI driver container.

The CSI attacher and CSI provisioner containers communicate with the CSI driver container using UNIX Domain Sockets, ensuring that no CSI communication leaves the pod. The CSI driver is not accessible from outside of the pod.

NOTE

The **attach**, **detach**, **provision**, and **delete** operations typically require the CSI driver to use credentials to the storage backend. Run the CSI controller pods on infrastructure nodes so the credentials are never leaked to user processes, even in the event of a catastrophic security breach on a compute node.

NOTE

The external attacher must also run for CSI drivers that do not support third-party **attach** or **detach** operations. The external attacher will not issue any **ControllerPublish** or **ControllerUnpublish** operations to the CSI driver. However, it still must run to implement the necessary OpenShift Container Platform attachment API.

5.1.1.2. CSI driver daemon set

The CSI driver daemon set runs a pod on every node that allows OpenShift Container Platform to mount storage provided by the CSI driver to the node and use it in user workloads (pods) as persistent volumes (PVs). The pod with the CSI driver installed contains the following containers:

- A CSI driver registrar, which registers the CSI driver into the **openshift-node** service running on the node. The **openshift-node** process running on the node then directly connects with the CSI driver using the UNIX Domain Socket available on the node.

- A CSI driver.

The CSI driver deployed on the node should have as few credentials to the storage back end as possible. OpenShift Container Platform will only use the node plugin set of CSI calls such as **NodePublish**/**NodeUnpublish** and **NodeStage**/**NodeUnstage**, if these calls are implemented.

5.1.2. CSI drivers supported by OpenShift Container Platform

OpenShift Container Platform installs certain CSI drivers by default, giving users storage options that are not possible with in-tree volume plugins.

To create CSI-provisioned persistent volumes that mount to these supported storage assets, OpenShift Container Platform installs the necessary CSI driver Operator, the CSI driver, and the required storage class by default. For more details about the default namespace of the Operator and driver, see the documentation for the specific CSI Driver Operator.

IMPORTANT

The AWS EFS and GCP Filestore CSI drivers are not installed by default, and must be installed manually. For instructions on installing the AWS EFS CSI driver, see Setting up AWS Elastic File Service CSI Driver Operator. For instructions on installing the GCP Filestore CSI driver, see Google Compute Platform Filestore CSI Driver Operator .

The following table describes the CSI drivers that are installed with OpenShift Container Platform supported by OpenShift Container Platform and which CSI features they support, such as volume snapshots and resize.

Table 5.1. Supported CSI drivers and features in OpenShift Container Platform

CSI driver	CSI volume snapshots	CSI cloning	CSI resize	Inline ephemeral volumes
AliCloud Disk	▮	-	▮	-
AWS EBS	▮	-	▮	-
AWS EFS	-	-	-	-
Google Compute Platform (GCP) persistent disk (PD)	▮	▮	▮	-
GCP Filestore	▮	-	▮	-
IBM Power® Virtual Server Block	-	-	▮	-
IBM Cloud® Block	▮[3]	-	▮[3]	-
Microsoft Azure Disk	▮	▮	▮	-
Microsoft Azure Stack Hub	▮	▮	▮	-
Microsoft Azure File	-	-	▮	▮
OpenStack Cinder	▮	▮	▮	-

CSI driver	CSI volume snapshots	CSI cloning	CSI resize	Inline ephemeral volumes
OpenShift Data Foundation	✓	✓	✓	-
OpenStack Manila	✓	-	-	-
Shared Resource	-	-	-	✓
VMware vSphere	✓[1]	-	✓[2]	-

1.

- Requires vSphere version 7.0 Update 3 or later for both vCenter Server and ESXi.
- Does not support fileshare volumes.

2.

- Offline volume expansion: minimum required vSphere version is 6.7 Update 3 P06
- Online volume expansion: minimum required vSphere version is 7.0 Update 2.

3.

- Does not support offline snapshots or resize. Volume must be attached to a running pod.

IMPORTANT

If your CSI driver is not listed in the preceding table, you must follow the installation instructions provided by your CSI storage vendor to use their supported CSI features.

5.1.3. Dynamic provisioning

Dynamic provisioning of persistent storage depends on the capabilities of the CSI driver and underlying storage back end. The provider of the CSI driver should document how to create a storage class in OpenShift Container Platform and the parameters available for configuration.

The created storage class can be configured to enable dynamic provisioning.

Procedure

- Create a default storage class that ensures all PVCs that do not require any special storage class are provisioned by the installed CSI driver.

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class> 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: <provisioner-name> 2
parameters:
EOF
```

1 The name of the storage class that will be created.

2 The name of the CSI driver that has been installed.

5.1.4. Example using the CSI driver

The following example installs a default MySQL template without any changes to the template.

Prerequisites

- The CSI driver has been deployed.
- A storage class has been created for dynamic provisioning.

Procedure

- Create the MySQL template:

```
# oc new-app mysql-persistent
```

Example output

```
--> Deploying template "openshift/mysql-persistent" to project default
...
```

```
# oc get pvc
```

Example output

```
NAME          STATUS    VOLUME                                 CAPACITY
ACCESS MODES   STORAGECLASS   AGE
mysql         Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi
RWO           cinder        3s
```

5.1.5. Volume populators

Volume populators use the **datasource** field in a persistent volume claim (PVC) spec to create pre-populated volumes.

Volume population is currently enabled, and supported as a Technology Preview feature. However, OpenShift Container Platform does not ship with any volume populators.

IMPORTANT

Volume populators is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more information about volume populators, see Kubernetes volume populators .

5.2. CSI INLINE EPHEMERAL VOLUMES

Container Storage Interface (CSI) inline ephemeral volumes allow you to define a **Pod** spec that creates inline ephemeral volumes when a pod is deployed and delete them when a pod is destroyed.

This feature is only available with supported Container Storage Interface (CSI) drivers:

- Shared Resource CSI driver
- Azure File CSI driver
- Secrets Store CSI driver

5.2.1. Overview of CSI inline ephemeral volumes

Traditionally, volumes that are backed by Container Storage Interface (CSI) drivers can only be used with a **PersistentVolume** and **PersistentVolumeClaim** object combination.

This feature allows you to specify CSI volumes directly in the **Pod** specification, rather than in a **PersistentVolume** object. Inline volumes are ephemeral and do not persist across pod restarts.

5.2.1.1. Support limitations

By default, OpenShift Container Platform supports CSI inline ephemeral volumes with these limitations:

- Support is only available for CSI drivers. In-tree and FlexVolumes are not supported.
- The Shared Resource CSI Driver supports using inline ephemeral volumes only to access **Secrets** or **ConfigMaps** across multiple namespaces as a Technology Preview feature.
- Community or storage vendors provide other CSI drivers that support these volumes. Follow the installation instructions provided by the CSI driver provider.

CSI drivers might not have implemented the inline volume functionality, including **Ephemeral** capacity. For details, see the CSI driver documentation.

IMPORTANT

Shared Resource CSI Driver is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

5.2.2. CSI Volume Admission plugin

The Container Storage Interface (CSI) Volume Admission plugin allows you to restrict the use of an individual CSI driver capable of provisioning CSI ephemeral volumes on pod admission. Administrators can add a **csi-ephemeral-volume-profile** label, and this label is then inspected by the Admission plugin and used in enforcement, warning, and audit decisions.

5.2.2.1. Overview

To use the CSI Volume Admission plugin, administrators add the **security.openshift.io/csi-ephemeral-volume-profile** label to a **CSIDriver** object, which declares the CSI driver's effective pod security profile when it is used to provide CSI ephemeral volumes, as shown in the following example:

```
kind: CSIDriver
metadata:
  name: csi.mydriver.company.org
  labels:
    security.openshift.io/csi-ephemeral-volume-profile: restricted 1
```

1 CSI driver object YAML file with the **csi-ephemeral-volume-profile** label set to "restricted"

This “effective profile” communicates that a pod can use the CSI driver to mount CSI ephemeral volumes when the pod's namespace is governed by a pod security standard.

The CSI Volume Admission plugin inspects pod volumes when pods are created; existing pods that use CSI volumes are not affected. If a pod uses a container storage interface (CSI) volume, the plugin looks up the **CSIDriver** object and inspects the **csi-ephemeral-volume-profile** label, and then use the label's value in its enforcement, warning, and audit decisions.

5.2.2.2. Pod security profile enforcement

When a CSI driver has the **csi-ephemeral-volume-profile** label, pods using the CSI driver to mount CSI ephemeral volumes must run in a namespace that enforces a pod security standard of equal or greater permission. If the namespace enforces a more restrictive standard, the CSI Volume Admission plugin denies admission. The following table describes the enforcement behavior for different pod security profiles for given label values.

Table 5.2. Pod security profile enforcement

Pod security profile	Driver label: restricted	Driver label: baseline	Driver label: privileged

Pod security profile	Driver label: restricted	Driver label: baseline	Driver label: privileged
Restricted	Allowed	Denied	Denied
Baseline	Allowed	Allowed	Denied
Privileged	Allowed	Allowed	Allowed

5.2.2.3. Pod security profile warning

The CSI Volume Admission plugin can warn you if the CSI driver’s effective profile is more permissive than the pod security warning profile for the pod namespace. The following table shows when a warning occurs for different pod security profiles for given label values.

Table 5.3. Pod security profile warning

Pod security profile	Driver label: restricted	Driver label: baseline	Driver label: privileged
Restricted	No warning	Warning	Warning
Baseline	No warning	No warning	Warning
Privileged	No warning	No warning	No warning

5.2.2.4. Pod security profile audit

The CSI Volume Admission plugin can apply audit annotations to the pod if the CSI driver’s effective profile is more permissive than the pod security audit profile for the pod namespace. The following table shows the audit annotation applied for different pod security profiles for given label values.

Table 5.4. Pod security profile audit

Pod security profile	Driver label: restricted	Driver label: baseline	Driver label: privileged
Restricted	No audit	Audit	Audit
Baseline	No audit	No audit	Audit
Privileged	No audit	No audit	No audit

5.2.2.5. Default behavior for the CSI Volume Admission plugin

If the referenced CSI driver for a CSI ephemeral volume does not have the **csi-ephemeral-volume-profile** label, the CSI Volume Admission plugin considers the driver to have the privileged profile for enforcement, warning, and audit behaviors. Likewise, if the pod's namespace does not have the pod security admission label set, the Admission plugin assumes the restricted profile is allowed for enforcement, warning, and audit decisions. Therefore, if no labels are set, CSI ephemeral volumes using that CSI driver are only usable in privileged namespaces by default.

The CSI drivers that ship with OpenShift Container Platform and support ephemeral volumes have a reasonable default set for the **csi-ephemeral-volume-profile** label:

- Shared Resource CSI driver: restricted
- Azure File CSI driver: privileged

An admin can change the default value of the label if desired.

5.2.3. Embedding a CSI inline ephemeral volume in the pod specification

You can embed a CSI inline ephemeral volume in the **Pod** specification in OpenShift Container Platform. At runtime, nested inline volumes follow the ephemeral lifecycle of their associated pods so that the CSI driver handles all phases of volume operations as pods are created and destroyed.

Procedure

1. Create the **Pod** object definition and save it to a file.

2. Embed the CSI inline ephemeral volume in the file.

   my-csi-app.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-csi-app
spec:
  containers:
    - name: my-frontend
      image: busybox
      volumeMounts:
      - mountPath: "/data"
        name: my-csi-inline-vol
      command: [ "sleep", "1000000" ]
  volumes: 1
    - name: my-csi-inline-vol
      csi:
        driver: inline.storage.kubernetes.io
        volumeAttributes:
          foo: bar
```

   1 The name of the volume that is used by pods.

3. Create the object definition file that you saved in the previous step.

```
$ oc create -f my-csi-app.yaml
```

5.2.4. Additional resources

- Pod Security Standards

5.3. SHARED RESOURCE CSI DRIVER OPERATOR

As a cluster administrator, you can use the Shared Resource CSI Driver in OpenShift Container Platform to provision inline ephemeral volumes that contain the contents of **Secret** or **ConfigMap** objects. This way, pods and other Kubernetes types that expose volume mounts, and OpenShift Container Platform Builds can securely use the contents of those objects across potentially any namespace in the cluster. To accomplish this, there are currently two types of shared resources: a **SharedSecret** custom resource for **Secret** objects, and a **SharedConfigMap** custom resource for **ConfigMap** objects.

IMPORTANT

The Shared Resource CSI Driver is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

NOTE

To enable the Shared Resource CSI Driver, you must enable features using feature gates.

5.3.1. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.3.2. Sharing secrets across namespaces

To share a secret across namespaces in a cluster, you create a **SharedSecret** custom resource (CR) instance for the **Secret** object that you want to share.

Prerequisites

You must have permission to perform the following actions:

- Create instances of the **sharedsecrets.sharedresource.openshift.io** custom resource definition (CRD) at a cluster-scoped level.

- Manage roles and role bindings across the namespaces in the cluster to control which users can get, list, and watch those instances.

- Manage roles and role bindings to control whether the service account specified by a pod can mount a Container Storage Interface (CSI) volume that references the **SharedSecret** CR instance you want to use.

- Access the namespaces that contain the Secrets you want to share.

Procedure

- Create a **SharedSecret** CR instance for the **Secret** object you want to share across namespaces in the cluster:

```
$ oc apply -f - <<EOF
apiVersion: sharedresource.openshift.io/v1alpha1
kind: SharedSecret
metadata:
  name: my-share
spec:
  secretRef:
    name: <name of secret>
    namespace: <namespace of secret>
EOF
```

5.3.3. Using a SharedSecret instance in a pod

To access a **SharedSecret** custom resource (CR) instance from a pod, you grant a given service account RBAC permissions to use that **SharedSecret** CR instance.

Prerequisites

- You have created a **SharedSecret** CR instance for the secret you want to share across namespaces in the cluster.

- You must have permission to perform the following actions
  - Discover which **SharedSecret** CR instances are available by entering the **oc get sharedsecrets** command and getting a non-empty list back.
  - Determine if the service account your pod specifies is allowed to use the given **SharedSecret** CR instance. That is, you can run **oc adm policy who-can use <identifier of specific SharedSecret>** to see if the service account in your namespace is listed.
  - Determine if the service account your pod specifies is allowed to use **csi** volumes, or if you, as the requesting user who created the pod directly, are allowed to use **csi** volumes. See "Understanding and managing pod security admission" for details.

NOTE

If neither of the last two prerequisites in this list are met, create, or ask someone to create, the necessary role-based access control (RBAC) so that you can discover **SharedSecret** CR instances and enable service accounts to use **SharedSecret** CR instances.

Procedure

1. Grant a given service account RBAC permissions to use the **SharedSecret** CR instance in its pod by using **oc apply** with YAML content:

   NOTE

   Currently, **kubectl** and **oc** have hard-coded special case logic restricting the **use** verb to roles centered around pod security. Therefore, you cannot use **oc create role …** to create the role needed for consuming **SharedSecret** CR instances.

```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-my-share
  namespace: my-namespace
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedsecrets
    resourceNames:
      - my-share
    verbs:
      - use
EOF
```

2. Create the **RoleBinding** associated with the role by using the **oc** command:

```
$ oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. Access the **SharedSecret** CR instance from a pod:

```
$ oc apply -f - <<EOF
kind: Pod
apiVersion: v1
metadata:
  name: my-app
  namespace: my-namespace
spec:
  serviceAccountName: default

# containers omitted …. Follow standard use of ‘volumeMounts’ for referencing your shared
resource volume

    volumes:
    - name: my-csi-volume
      csi:
        readOnly: true
        driver: csi.sharedresource.openshift.io
        volumeAttributes:
          sharedSecret: my-share

EOF
```

5.3.4. Sharing a config map across namespaces

To share a config map across namespaces in a cluster, you create a **SharedConfigMap** custom resource (CR) instance for that config map.

Prerequisites

You must have permission to perform the following actions:

- Create instances of the **sharedconfigmaps.sharedresource.openshift.io** custom resource definition (CRD) at a cluster-scoped level.
- Manage roles and role bindings across the namespaces in the cluster to control which users can get, list, and watch those instances.
- Manage roles and role bindings across the namespaces in the cluster to control which service accounts in pods that mount your Container Storage Interface (CSI) volume can use those instances.
- Access the namespaces that contain the Secrets you want to share.

Procedure

1. Create a **SharedConfigMap** CR instance for the config map that you want to share across namespaces in the cluster:

```
$ oc apply -f - <<EOF
apiVersion: sharedresource.openshift.io/v1alpha1
kind: SharedConfigMap
metadata:
  name: my-share
spec:
  configMapRef:
    name: <name of configmap>
    namespace: <namespace of configmap>
EOF
```

5.3.5. Using a SharedConfigMap instance in a pod

Next steps

To access a **SharedConfigMap** custom resource (CR) instance from a pod, you grant a given service account RBAC permissions to use that **SharedConfigMap** CR instance.

Prerequisites

- You have created a **SharedConfigMap** CR instance for the config map that you want to share across namespaces in the cluster.
- You must have permission to perform the following actions:
  - Discover which **SharedConfigMap** CR instances are available by entering the **oc get sharedconfigmaps** command and getting a non-empty list back.

- Determine if the service account your pod specifies is allowed to use the given **SharedSecret** CR instance. That is, you can run **oc adm policy who-can use <identifier of specific SharedSecret>** to see if the service account in your namespace is listed.
- Determine if the service account your pod specifies is allowed to use **csi** volumes, or if you, as the requesting user who created the pod directly, are allowed to use **csi** volumes. See "Understanding and managing pod security admission" for details.

NOTE

If neither of the last two prerequisites in this list are met, create, or ask someone to create, the necessary role-based access control (RBAC) so that you can discover **SharedConfigMap** CR instances and enable service accounts to use **SharedConfigMap** CR instances.

Procedure

1. Grant a given service account RBAC permissions to use the **SharedConfigMap** CR instance in its pod by using **oc apply** with YAML content.

   NOTE

   Currently, **kubectl** and **oc** have hard-coded special case logic restricting the **use** verb to roles centered around pod security. Therefore, you cannot use **oc create role …** to create the role needed for consuming a **SharedConfigMap** CR instance.

```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-my-share
  namespace: my-namespace
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedconfigmaps
    resourceNames:
      - my-share
    verbs:
      - use
EOF
```

2. Create the **RoleBinding** associated with the role by using the **oc** command:

```
oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. Access the **SharedConfigMap** CR instance from a pod:

```
$ oc apply -f - <<EOF
kind: Pod
apiVersion: v1
```

```
metadata:
  name: my-app
  namespace: my-namespace
spec:
  serviceAccountName: default

# containers omitted …. Follow standard use of ‘volumeMounts’ for referencing your shared resource volume

    volumes:
    - name: my-csi-volume
      csi:
        readOnly: true
        driver: csi.sharedresource.openshift.io
        volumeAttributes:
          sharedConfigMap: my-share

EOF
```

5.3.6. Additional support limitations for the Shared Resource CSI Driver

The Shared Resource CSI Driver has the following noteworthy limitations:

- The driver is subject to the limitations of Container Storage Interface (CSI) inline ephemeral volumes.
- The value of the **readOnly** field must be **true**. On **Pod** creation, a validating admission webhook rejects the pod creation if **readOnly** is **false**. If for some reason the validating admission webhook cannot be contacted, on volume provisioning during pod startup, the driver returns an error to the kubelet. Requiring **readOnly** is **true** is in keeping with proposed best practices for the upstream Kubernetes CSI Driver to apply SELinux labels to associated volumes.
- The driver ignores the **FSType** field because it only supports **tmpfs** volumes.
- The driver ignores the **NodePublishSecretRef** field. Instead, it uses **SubjectAccessReviews** with the **use** verb to evaluate whether a pod can obtain a volume that contains **SharedSecret** or **SharedConfigMap** custom resource (CR) instances.
- You cannot create **SharedSecret** or **SharedConfigMap** custom resource (CR) instances whose names start with **openshift**.

5.3.7. Additional details about VolumeAttributes on shared resource pod volumes

The following attributes affect shared resource pod volumes in various ways:

- The **refreshResource** attribute in the **volumeAttributes** properties.
- The **refreshResources** attribute in the Shared Resource CSI Driver configuration.
- The **sharedSecret** and **sharedConfigMap** attributes in the **volumeAttributes** properties.

5.3.7.1. The refreshResource attribute

The Shared Resource CSI Driver honors the **refreshResource** attribute in **volumeAttributes** properties of the volume. This attribute controls whether updates to the contents of the underlying **Secret** or

ConfigMap object are copied to the volume **after** the volume is initially provisioned as part of pod startup. The default value of **refreshResource** is **true**, which means that the contents are updated.

IMPORTANT

If the Shared Resource CSI Driver configuration has disabled the refreshing of both the shared **SharedSecret** and **SharedConfigMap** custom resource (CR) instances, then the **refreshResource** attribute in the **volumeAttribute** properties has no effect. The intent of this attribute is to disable refresh for specific volume mounts when refresh is generally allowed.

5.3.7.2. The refreshResources attribute

You can use a global switch to enable or disable refreshing of shared resources. This switch is the **refreshResources** attribute in the **csi-driver-shared-resource-config** config map for the Shared Resource CSI Driver, which you can find in the **openshift-cluster-csi-drivers** namespace. If you set this **refreshResources** attribute to **false**, none of the **Secret** or **ConfigMap** object-related content stored in the volume is updated after the initial provisioning of the volume.

IMPORTANT

Using this Shared Resource CSI Driver configuration to disable refreshing affects all the cluster's volume mounts that use the Shared Resource CSI Driver, regardless of the **refreshResource** attribute in the **volumeAttributes** properties of any of those volumes.

5.3.7.3. Validation of volumeAttributes before provisioning a shared resource volume for a pod

In the **volumeAttributes** of a single volume, you must set either a **sharedSecret** or a **sharedConfigMap** attribute to the value of a **SharedSecret** or a **SharedConfigMap** CS instance. Otherwise, when the volume is provisioned during pod startup, a validation checks the **volumeAttributes** of that volume and returns an error to the kubelet under the following conditions:

- Both **sharedSecret** and **sharedConfigMap** attributes have specified values.
- Neither **sharedSecret** nor **sharedConfigMap** attributes have specified values.
- The value of the **sharedSecret** or **sharedConfigMap** attribute does not correspond to the name of a **SharedSecret** or **SharedConfigMap** CR instance on the cluster.

5.3.8. Integration between shared resources, Insights Operator, and OpenShift Container Platform Builds

Integration between shared resources, Insights Operator, and OpenShift Container Platform Builds makes using Red Hat subscriptions (RHEL entitlements) easier in OpenShift Container Platform Builds.

Previously, in OpenShift Container Platform 4.9.x and earlier, you manually imported your credentials and copied them to each project or namespace where you were running builds.

Now, in OpenShift Container Platform 4.10 and later, OpenShift Container Platform Builds can use Red Hat subscriptions (RHEL entitlements) by referencing shared resources and the simple content access feature provided by Insights Operator:

- The simple content access feature imports your subscription credentials to a well-known **Secret** object. See the links in the following "Additional resources" section.

- The cluster administrator creates a **SharedSecret** custom resource (CR) instance around that **Secret** object and grants permission to particular projects or namespaces. In particular, the cluster administrator gives the **builder** service account permission to use that **SharedSecret** CR instance.

- Builds that run within those projects or namespaces can mount a CSI Volume that references the **SharedSecret** CR instance and its entitled RHEL content.

Additional resources

- Importing simple content access certificates with Insights Operator

- Adding subscription entitlements as a build secret

5.4. CSI VOLUME SNAPSHOTS

This document describes how to use volume snapshots with supported Container Storage Interface (CSI) drivers to help protect against data loss in OpenShift Container Platform. Familiarity with persistent volumes is suggested.

5.4.1. Overview of CSI volume snapshots

A *snapshot* represents the state of the storage volume in a cluster at a particular point in time. Volume snapshots can be used to provision a new volume.

OpenShift Container Platform supports Container Storage Interface (CSI) volume snapshots by default. However, a specific CSI driver is required.

With CSI volume snapshots, a cluster administrator can:

- Deploy a third-party CSI driver that supports snapshots.

- Create a new persistent volume claim (PVC) from an existing volume snapshot.

- Take a snapshot of an existing PVC.

- Restore a snapshot as a different PVC.

- Delete an existing volume snapshot.

With CSI volume snapshots, an app developer can:

- Use volume snapshots as building blocks for developing application- or cluster-level storage backup solutions.

- Rapidly rollback to a previous development version.

- Use storage more efficiently by not having to make a full copy each time.

Be aware of the following when using volume snapshots:

- Support is only available for CSI drivers. In-tree and FlexVolumes are not supported.

- OpenShift Container Platform only ships with select CSI drivers. For CSI drivers that are not provided by an OpenShift Container Platform Driver Operator, it is recommended to use the CSI drivers provided by community or storage vendors. Follow the installation instructions furnished by the CSI driver provider.

- CSI drivers may or may not have implemented the volume snapshot functionality. CSI drivers that have provided support for volume snapshots will likely use the **csi-external-snapshotter** sidecar. See documentation provided by the CSI driver for details.

5.4.2. CSI snapshot controller and sidecar

OpenShift Container Platform provides a snapshot controller that is deployed into the control plane. In addition, your CSI driver vendor provides the CSI snapshot sidecar as a helper container that is installed during the CSI driver installation.

The CSI snapshot controller and sidecar provide volume snapshotting through the OpenShift Container Platform API. These external components run in the cluster.

The external controller is deployed by the CSI Snapshot Controller Operator.

5.4.2.1. External controller

The CSI snapshot controller binds **VolumeSnapshot** and **VolumeSnapshotContent** objects. The controller manages dynamic provisioning by creating and deleting **VolumeSnapshotContent** objects.

5.4.2.2. External sidecar

Your CSI driver vendor provides the **csi-external-snapshotter** sidecar. This is a separate helper container that is deployed with the CSI driver. The sidecar manages snapshots by triggering **CreateSnapshot** and **DeleteSnapshot** operations. Follow the installation instructions provided by your vendor.

5.4.3. About the CSI Snapshot Controller Operator

The CSI Snapshot Controller Operator runs in the **openshift-cluster-storage-operator** namespace. It is installed by the Cluster Version Operator (CVO) in all clusters by default.

The CSI Snapshot Controller Operator installs the CSI snapshot controller, which runs in the **openshift-cluster-storage-operator** namespace.

5.4.3.1. Volume snapshot CRDs

During OpenShift Container Platform installation, the CSI Snapshot Controller Operator creates the following snapshot custom resource definitions (CRDs) in the **snapshot.storage.k8s.io/v1** API group:

VolumeSnapshotContent

A snapshot taken of a volume in the cluster that has been provisioned by a cluster administrator. Similar to the **PersistentVolume** object, the **VolumeSnapshotContent** CRD is a cluster resource that points to a real snapshot in the storage back end.

For manually pre-provisioned snapshots, a cluster administrator creates a number of **VolumeSnapshotContent** CRDs. These carry the details of the real volume snapshot in the storage system.

The **VolumeSnapshotContent** CRD is not namespaced and is for use by a cluster administrator.

VolumeSnapshot

Similar to the **PersistentVolumeClaim** object, the **VolumeSnapshot** CRD defines a developer request for a snapshot. The CSI Snapshot Controller Operator runs the CSI snapshot controller, which handles the binding of a **VolumeSnapshot** CRD with an appropriate **VolumeSnapshotContent** CRD. The binding is a one-to-one mapping.
The **VolumeSnapshot** CRD is namespaced. A developer uses the CRD as a distinct request for a snapshot.

VolumeSnapshotClass

Allows a cluster administrator to specify different attributes belonging to a **VolumeSnapshot** object. These attributes may differ among snapshots taken of the same volume on the storage system, in which case they would not be expressed by using the same storage class of a persistent volume claim.
The **VolumeSnapshotClass** CRD defines the parameters for the **csi-external-snapshotter** sidecar to use when creating a snapshot. This allows the storage back end to know what kind of snapshot to dynamically create if multiple options are supported.

Dynamically provisioned snapshots use the **VolumeSnapshotClass** CRD to specify storage-provider-specific parameters to use when creating a snapshot.

The **VolumeSnapshotContentClass** CRD is not namespaced and is for use by a cluster administrator to enable global configuration options for their storage back end.

5.4.4. Volume snapshot provisioning

There are two ways to provision snapshots: dynamically and manually.

5.4.4.1. Dynamic provisioning

Instead of using a preexisting snapshot, you can request that a snapshot be taken dynamically from a persistent volume claim. Parameters are specified using a **VolumeSnapshotClass** CRD.

5.4.4.2. Manual provisioning

As a cluster administrator, you can manually pre-provision a number of **VolumeSnapshotContent** objects. These carry the real volume snapshot details available to cluster users.

5.4.5. Creating a volume snapshot

When you create a **VolumeSnapshot** object, OpenShift Container Platform creates a volume snapshot.

Prerequisites

- Logged in to a running OpenShift Container Platform cluster.
- A PVC created using a CSI driver that supports **VolumeSnapshot** objects.
- A storage class to provision the storage back end.
- No pods are using the persistent volume claim (PVC) that you want to take a snapshot of.

NOTE

Do not create a volume snapshot of a PVC if a pod is using it. Doing so might cause data corruption because the PVC is not quiesced (paused). Be sure to first tear down a running pod to ensure consistent snapshots.

Procedure

To dynamically create a volume snapshot:

1. Create a file with the **VolumeSnapshotClass** object described by the following YAML:

volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io ❶
deletionPolicy: Delete
```

❶ The name of the CSI driver that is used to create snapshots of this **VolumeSnapshotClass** object. The name must be the same as the **Provisioner** field of the storage class that is responsible for the PVC that is being snapshotted.

NOTE

Depending on the driver that you used to configure persistent storage, additional parameters might be required. You can also use an existing **VolumeSnapshotClass** object.

2. Create the object you saved in the previous step by entering the following command:

```
$ oc create -f volumesnapshotclass.yaml
```

3. Create a **VolumeSnapshot** object:

volumesnapshot-dynamic.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: mysnap
spec:
  volumeSnapshotClassName: csi-hostpath-snap ❶
  source:
    persistentVolumeClaimName: myclaim ❷
```

❶ The request for a particular class by the volume snapshot. If the **volumeSnapshotClassName** setting is absent and there is a default volume snapshot class, a snapshot is created with the default volume snapshot class name. But if the field is absent and no default volume snapshot class exists, then no snapshot is created.

2 The name of the **PersistentVolumeClaim** object bound to a persistent volume. This defines what you want to create a snapshot of. Required for dynamically provisioning a

4. Create the object you saved in the previous step by entering the following command:

```
$ oc create -f volumesnapshot-dynamic.yaml
```

To manually provision a snapshot:

1. Provide a value for the **volumeSnapshotContentName** parameter as the source for the snapshot, in addition to defining volume snapshot class as shown above.

volumesnapshot-manual.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  source:
    volumeSnapshotContentName: mycontent 1
```

1 The **volumeSnapshotContentName** parameter is required for pre-provisioned snapshots.

2. Create the object you saved in the previous step by entering the following command:

```
$ oc create -f volumesnapshot-manual.yaml
```

Verification

After the snapshot has been created in the cluster, additional details about the snapshot are available.

1. To display details about the volume snapshot that was created, enter the following command:

```
$ oc describe volumesnapshot mysnap
```

The following example displays details about the **mysnap** volume snapshot:

volumesnapshot.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: mysnap
spec:
  source:
    persistentVolumeClaimName: myclaim
  volumeSnapshotClassName: csi-hostpath-snap
status:
  boundVolumeSnapshotContentName: snapcontent-1af4989e-a365-4286-96f8-
d5dcd65d78d6 1
```

```
    creationTime: "2020-01-29T12:24:30Z" 2
    readyToUse: true 3
    restoreSize: 500Mi
```

1 The pointer to the actual storage content that was created by the controller.

2 The time when the snapshot was created. The snapshot contains the volume content that was available at this indicated time.

3 If the value is set to **true**, the snapshot can be used to restore as a new PVC.
If the value is set to **false**, the snapshot was created. However, the storage back end needs to perform additional tasks to make the snapshot usable so that it can be restored as a new volume. For example, Amazon Elastic Block Store data might be moved to a different, less expensive location, which can take several minutes.

2. To verify that the volume snapshot was created, enter the following command:

   ```
   $ oc get volumesnapshotcontent
   ```

   The pointer to the actual content is displayed. If the **boundVolumeSnapshotContentName** field is populated, a **VolumeSnapshotContent** object exists and the snapshot was created.

3. To verify that the snapshot is ready, confirm that the **VolumeSnapshot** object has **readyToUse: true**.

5.4.6. Deleting a volume snapshot

You can configure how OpenShift Container Platform deletes volume snapshots.

Procedure

1. Specify the deletion policy that you require in the **VolumeSnapshotClass** object, as shown in the following example:

   volumesnapshotclass.yaml

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-hostpath-snap
   driver: hostpath.csi.k8s.io
   deletionPolicy: Delete 1
   ```

   1 When deleting the volume snapshot, if the **Delete** value is set, the underlying snapshot is deleted along with the **VolumeSnapshotContent** object. If the **Retain** value is set, both the underlying snapshot and **VolumeSnapshotContent** object remain.
   If the **Retain** value is set and the **VolumeSnapshot** object is deleted without deleting the corresponding **VolumeSnapshotContent** object, the content remains. The snapshot itself is also retained in the storage back end.

2. Delete the volume snapshot by entering the following command:

```
$ oc delete volumesnapshot <volumesnapshot_name>
```

Example output

```
volumesnapshot.snapshot.storage.k8s.io "mysnapshot" deleted
```

3. If the deletion policy is set to **Retain**, delete the volume snapshot content by entering the following command:

```
$ oc delete volumesnapshotcontent <volumesnapshotcontent_name>
```

4. Optional: If the **VolumeSnapshot** object is not successfully deleted, enter the following command to remove any finalizers for the leftover resource so that the delete operation can continue:

IMPORTANT

Only remove the finalizers if you are confident that there are no existing references from either persistent volume claims or volume snapshot contents to the **VolumeSnapshot** object. Even with the **--force** option, the delete operation does not delete snapshot objects until all finalizers are removed.

```
$ oc patch -n $PROJECT volumesnapshot/$NAME --type=merge -p '{"metadata":
{"finalizers":null}}'
```

Example output

```
volumesnapshotclass.snapshot.storage.k8s.io "csi-ocs-rbd-snapclass" deleted
```

The finalizers are removed and the volume snapshot is deleted.

5.4.7. Restoring a volume snapshot

The **VolumeSnapshot** CRD content can be used to restore the existing volume to a previous state.

After your **VolumeSnapshot** CRD is bound and the **readyToUse** value is set to **true**, you can use that resource to provision a new volume that is pre-populated with data from the snapshot.

Prerequisites

- Logged in to a running OpenShift Container Platform cluster.
- A persistent volume claim (PVC) created using a Container Storage Interface (CSI) driver that supports volume snapshots.
- A storage class to provision the storage back end.
- A volume snapshot has been created and is ready to use.

Procedure

1. Specify a **VolumeSnapshot** data source on a PVC as shown in the following:

pvc-restore.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim-restore
spec:
  storageClassName: csi-hostpath-sc
  dataSource:
    name: mysnap 1
    kind: VolumeSnapshot 2
    apiGroup: snapshot.storage.k8s.io 3
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

1 Name of the **VolumeSnapshot** object representing the snapshot to use as source.

2 Must be set to the **VolumeSnapshot** value.

3 Must be set to the **snapshot.storage.k8s.io** value.

2. Create a PVC by entering the following command:

```
$ oc create -f pvc-restore.yaml
```

3. Verify that the restored PVC has been created by entering the following command:

```
$ oc get pvc
```

A new PVC such as **myclaim-restore** is displayed.

5.5. CSI VOLUME CLONING

Volume cloning duplicates an existing persistent volume to help protect against data loss in OpenShift Container Platform. This feature is only available with supported Container Storage Interface (CSI) drivers. You should be familiar with persistent volumes before you provision a CSI volume clone.

5.5.1. Overview of CSI volume cloning

A Container Storage Interface (CSI) volume clone is a duplicate of an existing persistent volume at a particular point in time.

Volume cloning is similar to volume snapshots, although it is more efficient. For example, a cluster administrator can duplicate a cluster volume by creating another instance of the existing cluster volume.

Cloning creates an exact duplicate of the specified volume on the back-end device, rather than creating a new empty volume. After dynamic provisioning, you can use a volume clone just as you would use any standard volume.

No new API objects are required for cloning. The existing **dataSource** field in the **PersistentVolumeClaim** object is expanded so that it can accept the name of an existing PersistentVolumeClaim in the same namespace.

5.5.1.1. Support limitations

By default, OpenShift Container Platform supports CSI volume cloning with these limitations:

- The destination persistent volume claim (PVC) must exist in the same namespace as the source PVC.
- Cloning is supported with a different Storage Class.
  - Destination volume can be the same for a different storage class as the source.
  - You can use the default storage class and omit **storageClassName** in the **spec**.
- Support is only available for CSI drivers. In-tree and FlexVolumes are not supported.
- CSI drivers might not have implemented the volume cloning functionality. For details, see the CSI driver documentation.

5.5.2. Provisioning a CSI volume clone

When you create a cloned persistent volume claim (PVC) API object, you trigger the provisioning of a CSI volume clone. The clone pre-populates with the contents of another PVC, adhering to the same rules as any other persistent volume. The one exception is that you must add a **dataSource** that references an existing PVC in the same namespace.

Prerequisites

- You are logged in to a running OpenShift Container Platform cluster.
- Your PVC is created using a CSI driver that supports volume cloning.
- Your storage back end is configured for dynamic provisioning. Cloning support is not available for static provisioners.

Procedure

To clone a PVC from an existing PVC:

1. Create and save a file with the **PersistentVolumeClaim** object described by the following YAML:

   pvc-clone.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-1-clone
  namespace: mynamespace
spec:
  storageClassName: csi-cloning 1
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: pvc-1
```

1 The name of the storage class that provisions the storage back end. The default storage class can be used and **storageClassName** can be omitted in the spec.

2. Create the object you saved in the previous step by running the following command:

```
$ oc create -f pvc-clone.yaml
```

A new PVC **pvc-1-clone** is created.

3. Verify that the volume clone was created and is ready by running the following command:

```
$ oc get pvc pvc-1-clone
```

The **pvc-1-clone** shows that it is **Bound**.

You are now ready to use the newly cloned PVC to configure a pod.

4. Create and save a file with the **Pod** object described by the YAML. For example:

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

1 The cloned PVC created during the CSI volume cloning operation.

The created **Pod** object is now ready to consume, clone, snapshot, or delete your cloned PVC independently of its original **dataSource** PVC.

5.6. MANAGING THE DEFAULT STORAGE CLASS

5.6.1. Overview

Managing the default storage class allows you to accomplish several different objectives:

- Enforcing static provisioning by disabling dynamic provisioning.
- When you have other preferred storage classes, preventing the storage operator from re-creating the initial default storage class.
- Renaming, or otherwise changing, the default storage class

To accomplish these objectives, you change the setting for the **spec.storageClassState** field in the **ClusterCSIDriver** object. The possible settings for this field are:

- **Managed**: (Default) The Container Storage Interface (CSI) operator is actively managing its default storage class, so that most manual changes made by a cluster administrator to the default storage class are removed, and the default storage class is continuously re-created if you attempt to manually delete it.
- **Unmanaged**: You can modify the default storage class. The CSI operator is not actively managing storage classes, so that it is not reconciling the default storage class it creates automatically.
- **Removed**: The CSI operators deletes the default storage class.

Managing the default storage classes is supported by the following Container Storage Interface (CSI) driver operators:

- AliCloud Disk
- Amazon Web Services (AWS) Elastic Block Storage (EBS)
- Azure Disk
- Azure File
- Google Cloud Platform (GCP) Persistent Disk (PD)
- IBM® VPC Block
- OpenStack Cinder
- VMware vSphere

5.6.2. Managing the default storage class using the web console

Prerequisites

- Access to the OpenShift Container Platform web console.
- Access to the cluster with cluster-admin privileges.

Procedure

To manage the default storage class using the web console:

1. Log in to the web console.
2. Click **Administration** > **CustomResourceDefinitions**.

3. On the **CustomResourceDefinitions** page, type **clustercsidriver** to find the **ClusterCSIDriver** object.

4. Click **ClusterCSIDriver**, and then click the **Instances** tab.

5. Click the name of the desired instance, and then click the **YAML** tab.

6. Add the **spec.storageClassState** field with a value of **Managed**, **Unmanaged**, or **Removed**.

   Example

   ```
   ...
   spec:
     driverConfig:
       driverType: ''
     logLevel: Normal
     managementState: Managed
     observedConfig: null
     operatorLogLevel: Normal
     storageClassState: Unmanaged 1
   ...
   ```

   1 **spec.storageClassState** field set to "Unmanaged"

7. Click **Save**.

5.6.3. Managing the default storage class using the CLI

Prerequisites

- Access to the cluster with cluster-admin privileges.

Procedure

To manage the storage class using the CLI, run the following command:

```
oc patch clustercsidriver $DRIVERNAME --type=merge -p "{\"spec\":
{\"storageClassState\":\"${STATE}\"}}" 1
```

1 Where **${STATE}** is "Removed" or "Managed" or "Unmanaged".

Where **$DRIVERNAME** is the provisioner name. You can find the provisioner name by running the command **oc get sc**.

5.6.4. Absent or multiple default storage classes

5.6.4.1. Multiple default storage classes

Multiple default storage classes can occur if you mark a non-default storage class as default and do not unset the existing default storage class, or you create a default storage class when a default storage class is already present. With multiple default storage classes present, any persistent volume claim (PVC) requesting the default storage class (**pvc.spec.storageClassName**=nil) gets the most recently created default storage class, regardless of the default status of that storage class, and the administrator receives an alert in the alerts dashboard that there are multiple default storage classes, **MultipleDefaultStorageClasses**.

5.6.4.2. Absent default storage class

There are two possible scenarios where PVCs can attempt to use a non-existent default storage class:

- An administrator removes the default storage class or marks it as non-default, and then a user creates a PVC requesting the default storage class.
- During installation, the installer creates a PVC requesting the default storage class, which has not yet been created.

In the preceding scenarios, the PVCs remain in pending state indefinitely.

OpenShift Container Platform provides a feature to retroactively assign the default storage class to PVCs, so that they do not remain in the pending state. With this feature enabled, PVCs requesting the default storage class that are created when no default storage classes exists, remain in the pending state until a default storage class is created, or one of the existing storage classes is declared the default. As soon as the default storage class is created or declared, the PVC gets the new default storage class.

IMPORTANT

Retroactive default storage class assignment is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

5.6.4.2.1. Procedure

To enable retroactive default storage class assignment:

1. Enable feature gates (see *Nodes* → *Working with clusters* → *Enabling features using feature gates*).

   IMPORTANT

   After turning on Technology Preview features using feature gates, they cannot be turned off. As a result, cluster upgrades are prevented.

   The following configuration example enables retroactive default storage class assignment, and all other Technology Preview features:

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
```

```
spec:
  featureSet: TechPreviewNoUpgrade 1
...
```

1 Enables retroactive default storage class assignment.

5.6.5. Changing the default storage class

Use the following procedure to change the default storage class.

For example, if you have two defined storage classes, **gp3** and **standard**, and you want to change the default storage class from **gp3** to **standard**.

Prerequisites

- Access to the cluster with cluster-admin privileges.

Procedure

To change the default storage class:

1. List the storage classes:

```
$ oc get storageclass
```

Example output

```
NAME           TYPE
gp3 (default)  kubernetes.io/aws-ebs 1
standard       kubernetes.io/aws-ebs
```

1 **(default)** indicates the default storage class.

2. Make the desired storage class the default.
For the desired storage class, set the **storageclass.kubernetes.io/is-default-class** annotation to **true** by running the following command:

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

NOTE

You can have multiple default storage classes for a short time. However, you should ensure that only one default storage class exists eventually.

With multiple default storage classes present, any persistent volume claim (PVC) requesting the default storage class (**pvc.spec.storageClassName**=nil) gets the most recently created default storage class, regardless of the default status of that storage class, and the administrator receives an alert in the alerts dashboard that there are multiple default storage classes, **MultipleDefaultStorageClasses**.

3. Remove the default storage class setting from the old default storage class.
   For the old default storage class, change the value of the **storageclass.kubernetes.io/is-default-class** annotation to **false** by running the following command:

   ```
   $ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
   ```

4. Verify the changes:

   ```
   $ oc get storageclass
   ```

   Example output

   ```
   NAME                 TYPE
   gp3                  kubernetes.io/aws-ebs
   standard (default)   kubernetes.io/aws-ebs
   ```

5.7. CSI AUTOMATIC MIGRATION

In-tree storage drivers that are traditionally shipped with OpenShift Container Platform are being deprecated and replaced by their equivalent Container Storage Interface (CSI) drivers. OpenShift Container Platform provides automatic migration for in-tree volume plugins to their equivalent CSI drivers.

5.7.1. Overview

This feature automatically migrates volumes that were provisioned using in-tree storage plugins to their counterpart Container Storage Interface (CSI) drivers.

This process does not perform any data migration; OpenShift Container Platform only translates the persistent volume object in memory. As a result, the translated persistent volume object is not stored on disk, nor is its contents changed. CSI automatic migration should be seamless. This feature does not change how you use all existing API objects: for example, **PersistentVolumes**, **PersistentVolumeClaims**, and **StorageClasses**.

The following in-tree to CSI drivers are automatically migrated:

- Azure Disk
- OpenStack Cinder
- Amazon Web Services (AWS) Elastic Block Storage (EBS)
- Google Compute Engine Persistent Disk (GCP PD)
- Azure File
- VMware vSphere (see information below for specific migration behavior for vSphere)

CSI migration for these volume types is considered generally available (GA), and requires no manual intervention.

CSI automatic migration of in-tree persistent volumes (PVs) or persistent volume claims (PVCs) does not enable any new CSI driver features, such as snapshots or expansion, if the original in-tree storage plugin did not support it.

5.7.2. Storage class implications

For new OpenShift Container Platform 4.13, and later, installations, the default storage class is the CSI storage class. All volumes provisioned using this storage class are CSI persistent volumes (PVs).

For clusters upgraded from 4.12, and earlier, to 4.13, and later, the CSI storage class is created, and is set as the default if no default storage class was set prior to the upgrade. In the very unlikely case that there is a storage class with the same name, the existing storage class remains unchanged. Any existing in-tree storage classes remain, and might be necessary for certain features, such as volume expansion to work for existing in-tree PVs. While storage class referencing to the in-tree storage plugin will continue working, we recommend that you switch the default storage class to the CSI storage class.

To change the default storage class, see Changing the default storage class.

5.7.3. vSphere automatic migration

5.7.3.1. New installations of OpenShift Container Platform

For new installations of OpenShift Container Platform 4.13, or later, automatic migration is enabled by default.

5.7.3.2. Updating from OpenShift Container Platform 4.13 to 4.14

If you are using vSphere in-tree persistent volumes (PVs) and want to update from OpenShift Container Platform 4.13 to 4.14, update vSphere vCenter and ESXI host to 7.0 Update 3L or 8.0 Update 2, otherwise the OpenShift Container Platform update is blocked. After updating vSphere, your OpenShift Container Platform update can occur and automatic migration is enabled by default.

Alternatively, if you do not want to update vSphere, you can proceed with an OpenShift Container Platform update by performing an administrator acknowledgment:

```
oc -n openshift-config patch cm admin-acks --patch '{"data":{"ack-4.13-kube-127-vsphere-migration-in-4.14":"true"}}' --type=merge
```

IMPORTANT

If you do **not** update to vSphere 7.0 Update 3L or 8.0 Update 2 and use an administrator acknowledgment to update to OpenShift Container Platform 4.14, known issues can occur due to CSI migration being enabled by default in OpenShift Container Platform 4.14. Before proceeding with the administrator acknowledgement, carefully read this knowledge base article.

5.7.3.3. Updating from OpenShift Container Platform 4.12 to 4.14

If you are using vSphere in-tree persistent volumes (PVs) and want to update from OpenShift Container Platform 4.12 to 4.14, update vSphere vCenter and ESXI host to 7.0 Update 3L or 8.0 Update 2, otherwise the OpenShift Container Platform update is blocked. After updating vSphere, your OpenShift Container Platform update can occur and automatic migration is enabled by default.

Alternatively, if you do not want to update vSphere, you can proceed with an OpenShift Container Platform update by performing an administrator acknowledgment by running **both** of the following commands:

```
oc -n openshift-config patch cm admin-acks --patch '{"data":{"ack-4.12-kube-126-vsphere-migration-in-4.14":"true"}}' --type=merge
```

```
oc -n openshift-config patch cm admin-acks --patch '{"data":{"ack-4.13-kube-127-vsphere-migration-in-4.14":"true"}}' --type=merge
```

IMPORTANT

If you do **not** update to vSphere 7.0 Update 3L or 8.0 Update 2 and use an administrator acknowledgment to update to OpenShift Container Platform 4.14, known issues can occur due to CSI migration being enabled by default in OpenShift Container Platform 4.14. Before proceeding with the administrator acknowledgement, carefully read this knowledge base article.

Updating from OpenShift Container Platform 4.12 to 4.14 is an Extended Update Support (EUS)-to-EUS update. To understand the ramifications for this type of update and how to perform it, see the *EUS-to-EUS update* link in the *Additional resources* section below.

Additional resources

- Performing an EUS-to-EUS update

5.8. DETACH CSI VOLUMES AFTER NON-GRACEFUL NODE SHUTDOWN

This feature allows Container Storage Interface (CSI) drivers to automatically detach volumes when a node goes down non-gracefully.

IMPORTANT

Detach CSI volumes after non-graceful node shutdown is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

5.8.1. Overview

A graceful node shutdown occurs when the kubelet's node shutdown manager detects the upcoming node shutdown action. Non-graceful shutdowns occur when the kubelet does not detect a node shutdown action, which can occur because of system or hardware failures. Also, the kubelet may not detect a node shutdown action when the shutdown command does not trigger the Inhibitor Locks mechanism used by the kubelet on Linux, or because of a user error, for example, if the shutdownGracePeriod and shutdownGracePeriodCriticalPods details are not configured correctly for that node.

With this feature, when a non-graceful node shutdown occurs, you can manually add an **out-of-service** taint on the node to allow volumes to automatically detach from the node.

5.8.2. Adding an out-of-service taint manually for automatic volume detachment

Prerequisites

- Access to the cluster with cluster-admin privileges.

Procedure

To allow volumes to detach automatically from a node after a non-graceful node shutdown:

1. After a node is detected as unhealthy, shut down the worker node.
2. Ensure that the node is shutdown by running the following command and checking the status:

```
oc get node <node name> ①
```

① <node name> = name of the non-gracefully shutdown node

IMPORTANT

If the node is not completely shut down, do not proceed with tainting the node. If the node is still up and the taint is applied, filesystem corruption can occur.

3. Taint the corresponding node object by running the following command:

```
oc adm taint node <node name> node.kubernetes.io/out-of-
service=nodeshutdown:NoExecute ①
```

① <node name> = name of the non-gracefully shutdown node

After the taint is applied, the volumes detach from the shutdown node allowing their disks to be attached to a different node.

Example

The resulting YAML file resembles the following:

```
spec:
  taints:
  - effect: NoExecute
    key: node.kubernetes.io/out-of-service
    value: nodeshutdown
```

4. Restart the node.
5. Remove the taint.

5.9. ALICLOUD DISK CSI DRIVER OPERATOR

5.9.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Alibaba AliCloud Disk Storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to AliCloud Disk storage assets, OpenShift Container Platform installs the AliCloud Disk CSI Driver Operator and the AliCloud Disk CSI driver, by default, in the **openshift-cluster-csi-drivers** namespace.

- The *AliCloud Disk CSI Driver Operator* provides a storage class (**alicloud-disk**) that you can use to create persistent volume claims (PVCs). The AliCloud Disk CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on demand, eliminating the need for cluster administrators to pre-provision storage. You can disable this default storage class if desired (see Managing the default storage class).

- The *AliCloud Disk CSI driver* enables you to create and mount AliCloud Disk PVs.

5.9.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

Additional resources

- Configuring CSI volumes

5.10. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR

5.10.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the AWS EBS CSI driver.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a Container Storage Interface (CSI) Operator and driver.

To create CSI-provisioned PVs that mount to AWS EBS storage assets, OpenShift Container Platform installs the AWS EBS CSI Driver Operator (a Red Hat operator) and the AWS EBS CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *AWS EBS CSI Driver Operator* provides a StorageClass by default that you can use to create PVCs. You can disable this default storage class if desired (see Managing the default storage class). You also have the option to create the AWS EBS StorageClass as described in Persistent storage using Amazon Elastic Block Store .

- The *AWS EBS CSI driver* enables you to create and mount AWS EBS PVs.

NOTE

If you installed the AWS EBS CSI Operator and driver on an OpenShift Container Platform 4.5 cluster, you must uninstall the 4.5 Operator and driver before you update to OpenShift Container Platform 4.14.

5.10.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

IMPORTANT

OpenShift Container Platform defaults to using the CSI plugin to provision Amazon Elastic Block Store (Amazon EBS) storage.

For information about dynamically provisioning AWS EBS persistent volumes in OpenShift Container Platform, see Persistent storage using Amazon Elastic Block Store .

5.10.3. User-managed encryption

The user-managed encryption feature allows you to provide keys during installation that encrypt OpenShift Container Platform node root volumes, and enables all managed storage classes to use these keys to encrypt provisioned storage volumes. You must specify the custom key in the **platform.<cloud_type>.defaultMachinePlatform** field in the install-config YAML file.

This features supports the following storage types:

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk storage
- Google Cloud Platform (GCP) persistent disk (PD) storage

NOTE

If there is no encrypted key defined in the storage class, only set **encrypted: "true"** in the storage class. The AWS EBS CSI driver uses the AWS managed alias/aws/ebs, which is created by Amazon EBS automatically in each region by default to encrypt provisioned storage volumes. In addition, the managed storage classes all have the **encrypted: "true"** setting.

For information about installing with user-managed encryption for Amazon EBS, see Installation configuration parameters.

Additional resources

- Persistent storage using Amazon Elastic Block Store
- Configuring CSI volumes

5.11. AWS ELASTIC FILE SERVICE CSI DRIVER OPERATOR

5.11.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for AWS Elastic File Service (EFS).

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

After installing the AWS EFS CSI Driver Operator, OpenShift Container Platform installs the AWS EFS CSI Operator and the AWS EFS CSI driver by default in the **openshift-cluster-csi-drivers** namespace. This allows the AWS EFS CSI Driver Operator to create CSI-provisioned PVs that mount to AWS EFS assets.

- The *AWS EFS CSI Driver Operator*, after being installed, does not create a storage class by default to use to create persistent volume claims (PVCs). However, you can manually create the AWS EFS **StorageClass**. The AWS EFS CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on-demand. This eliminates the need for cluster administrators to pre-provision storage.

- The *AWS EFS CSI driver* enables you to create and mount AWS EFS PVs.

NOTE

AWS EFS only supports regional volumes, not zonal volumes.

5.11.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.11.3. Setting up the AWS EFS CSI Driver Operator

1. Install the AWS EFS CSI Driver Operator (a Red Hat operator).

2. If you are using AWS EFS with AWS Secure Token Service (STS), obtain a role Amazon Resource Name (ARN) for STS. This is required for installing the AWS EFS CSI Driver Operator.

3. Install the AWS EFS CSI Driver Operator.

4. Install the AWS EFS CSI Driver.

5.11.3.1. Obtaining a role Amazon Resource Name for Security Token Service

This procedure explains how to obtain a role Amazon Resource Name (ARN) to configure the AWS EFS CSI Driver Operator with OpenShift Container Platform on AWS Security Token Service (STS).

IMPORTANT

Perform this procedure before you install the AWS EFS CSI Driver Operator (see *Installing the AWS EFS CSI Driver Operator* procedure).

Prerequisites

- Access to the cluster as a user with the cluster-admin role.
- AWS account credentials

Procedure

You can obtain the ARN role in multiple ways. The following procedure shows one method that uses the same concept and CCO utility (**ccoctl**) binary tool as cluster installation.

To obtain a role ARN for configuring AWS EFS CSI Driver Operator using STS:

1. Extract the **ccoctl** from the OpenShift Container Platform release image, which you used to install the cluster with STS. For more information, see "Configuring the Cloud Credential Operator utility".

2. Create and save an EFS **CredentialsRequest** YAML file, such as shown in the following example, and then place it in the **credrequests** directory:

   #### Example

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: openshift-aws-efs-csi-driver
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - elasticfilesystem:*
      effect: Allow
      resource: '*'
  secretRef:
    name: aws-efs-cloud-credentials
    namespace: openshift-cluster-csi-drivers
  serviceAccountNames:
  - aws-efs-csi-driver-operator
  - aws-efs-csi-driver-controller-sa
```

3. Run the **ccoctl** tool to generate a new IAM role in AWS, and create a YAML file for it in the local file system (**<path_to_ccoctl_output_dir>/manifests/openshift-cluster-csi-drivers-aws-efs-cloud-credentials-credentials.yaml**).

```
$ ccoctl aws create-iam-roles --name=<name> --region=<aws_region> --credentials-
requests-dir=<path_to_directory_with_list_of_credentials_requests>/credrequests --identity-
provider-arn=arn:aws:iam::<aws_account_id>:oidc-provider/<name>-oidc.s3.
<aws_region>.amazonaws.com
```

- **name=<name>** is the name used to tag any cloud resources that are created for tracking.
- **region=<aws_region>** is the AWS region where cloud resources are created.
- **dir=<path_to_directory_with_list_of_credentials_requests>/credrequests** is the directory containing the EFS CredentialsRequest file in previous step.
- **<aws_account_id>** is the AWS account ID.

  Example

  ```
  $ ccoctl aws create-iam-roles --name my-aws-efs --credentials-requests-dir credrequests
  --identity-provider-arn arn:aws:iam::123456789012:oidc-provider/my-aws-efs-oidc.s3.us-
  east-2.amazonaws.com
  ```

  Example output

  ```
  2022/03/21 06:24:44 Role arn:aws:iam::123456789012:role/my-aws-efs -openshift-
  cluster-csi-drivers-aws-efs-cloud- created
  2022/03/21 06:24:44 Saved credentials configuration to: /manifests/openshift-cluster-csi-
  drivers-aws-efs-cloud-credentials-credentials.yaml
  2022/03/21 06:24:45 Updated Role policy for Role my-aws-efs-openshift-cluster-csi-
  drivers-aws-efs-cloud-
  ```

4. Copy the role ARN from the first line of the *Example output* in the preceding step. The role ARN is between "Role" and "created". In this example, the role ARN is "arn:aws:iam::123456789012:role/my-aws-efs -openshift-cluster-csi-drivers-aws-efs-cloud". You will need the role ARN when you install the AWS EFS CSI Driver Operator.

Next steps

Install the AWS EFS CSI Driver Operator .

Additional resources

- Installing the AWS EFS CSI Driver Operator
- Configuring the Cloud Credential Operator utility
- Installing the AWS EFS CSI Driver

5.11.3.2. Installing the AWS EFS CSI Driver Operator

The AWS EFS CSI Driver Operator (a Red Hat operator) is not installed in OpenShift Container Platform by default. Use the following procedure to install and configure the AWS EFS CSI Driver Operator in your cluster.

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

To install the AWS EFS CSI Driver Operator from the web console:

1. Log in to the web console.

2. Install the AWS EFS CSI Operator:

   a. Click **Operators** → **OperatorHub**.

   b. Locate the AWS EFS CSI Operator by typing **AWS EFS CSI** in the filter box.

   c. Click the **AWS EFS CSI Driver Operator** button.

   > **IMPORTANT**
   >
   > Be sure to select the **AWS EFS CSI Driver Operator** and not the **AWS EFS Operator**. The **AWS EFS Operator** is a community Operator and is not supported by Red Hat.

   d. On the **AWS EFS CSI Driver Operator** page, click **Install**.

   e. On the **Install Operator** page, ensure that:

      - If you are using AWS EFS with AWS Secure Token Service (STS), in the **role ARN** field, enter the ARN role copied from the last step of the *Obtaining a role Amazon Resource Name for Security Token Service* procedure.

      - **All namespaces on the cluster (default)** is selected.

      - **Installed Namespace** is set to **openshift-cluster-csi-drivers**.

   f. Click **Install**.
   After the installation finishes, the AWS EFS CSI Operator is listed in the **Installed Operators** section of the web console.

Next steps

Install the AWS EFS CSI Driver .

5.11.3.3. Installing the AWS EFS CSI Driver

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

1. Click **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**.

2. On the **Instances** tab, click **Create ClusterCSIDriver**.

3. Use the following YAML file:

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
  name: efs.csi.aws.com
spec:
 managementState: Managed
```

4. Click **Create**.

5. Wait for the following Conditions to change to a "True" status:

   - AWSEFSDriverNodeServiceControllerAvailable

   - AWSEFSDriverControllerServiceControllerAvailable

5.11.4. Creating the AWS EFS storage class

Storage classes are used to differentiate and delineate storage levels and usages. By defining a storage class, users can obtain dynamically provisioned persistent volumes.

The *AWS EFS CSI Driver Operator* *(a Red Hat operator)*, after being installed, does not create a storage class by default. However, you can manually create the AWS EFS storage class.

5.11.4.1. Creating the AWS EFS storage class using the console

Procedure

1. In the OpenShift Container Platform console, click **Storage** → **StorageClasses**.

2. On the **StorageClasses** page, click **Create StorageClass**.

3. On the **StorageClass** page, perform the following steps:

   a. Enter a name to reference the storage class.

   b. Optional: Enter the description.

   c. Select the reclaim policy.

   d. Select **efs.csi.aws.com** from the **Provisioner** drop-down list.

   e. Optional: Set the configuration parameters for the selected provisioner.

4. Click **Create**.

5.11.4.2. Creating the AWS EFS storage class using the CLI

Procedure

- Create a **StorageClass** object:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap 1
  fileSystemId: fs-a5324911 2
  directoryPerms: "700" 3
```

```
gidRangeStart: "1000" 4
gidRangeEnd: "2000" 5
basePath: "/dynamic_provisioning" 6
```

1 **provisioningMode** must be **efs-ap** to enable dynamic provisioning.

2 **fileSystemId** must be the ID of the EFS volume created manually.

3 **directoryPerms** is the default permission of the root directory of the volume. In this example, the volume is accessible only by the owner.

4 5 **gidRangeStart** and **gidRangeEnd** set the range of POSIX Group IDs (GIDs) that are used to set the GID of the AWS access point. If not specified, the default range is 50000-7000000. Each provisioned volume, and thus AWS access point, is assigned a unique GID from this range.

6 **basePath** is the directory on the EFS volume that is used to create dynamically provisioned volumes. In this case, a PV is provisioned as "/dynamic_provisioning/<random uuid>" on the EFS volume. Only the subdirectory is mounted to pods that use the PV.

NOTE

A cluster admin can create several **StorageClass** objects, each using a different EFS volume.

5.11.5. AWS EFS CSI cross account support

Cross account support allows you to have an OpenShift Container Platform cluster in one AWS account and mount your file system in another AWS account using the AWS Elastic File System (EFS) Container Storage Interface (CSI) driver.

NOTE

Both the OpenShift Container Platform cluster and EFS file system must be in the same region.

Prerequisites

- Access to an OpenShift Container Platform cluster with administrator rights
- Two valid AWS accounts

Procedure

The following procedure demonstrates how to set up:

- OpenShift Container Platform cluster in AWS account A
- Mount an AWS EFS file system in account B

To use AWS EFS across accounts:

1. Install OpenShift Container Platform cluster with AWS account A and install the EFS CSI Driver Operator.

2. Create an EFS volume in AWS account B:

    a. Create a virtual private cloud (VPC) called, for example, "my-efs-vpc" with CIDR, for example, "172.20.0.0/16" and subnet for the AWS EFS volume.

    b. On the AWS console, go to https://console.aws.amazon.com/efs.

    c. Click **Create new filesystem**:

        i. Create a filesystem named, for example, "my-filesystem".

        ii. Select the VPC created earlier ("my-efs-vpc").

        iii. Accept the default for the remaining settings.

    d. Ensure that the volume and Mount Targets have been created:

        i. Check https://console.aws.amazon.com/efs#/file-systems.

        ii. Click your volume, and on the **Network** tab wait for all Mount Targets to be available (approximately 1-2 minutes).

    e. On the **Network** tab, copy the Security Group ID. You will need it for the next step.

3. Configure networking access to the AWS EFS volume on AWS account B:

    a. Go to https://console.aws.amazon.com/ec2/v2/home#SecurityGroups.

    b. Find the Security Group used by the AWS EFS volume by filtering for the group ID copied earlier.

    c. On the **Inbound rules** tab, click **Edit inbound rules**, and then add a new rule to allow OpenShift Container Platform nodes to access the AWS EFS volumes (that is, use NFS ports from the cluster):

        - **Type**: NFS
        - **Protocol**: TCP
        - **Port range**: 2049
        - **Source**: Custom/IP address range of your OpenShift Container Platform cluster nodes (for example, "10.0.0.0/16")

    d. Save the rule.

    NOTE

    If you encounter mounting issues, re-check the port number, IP address range, and verify that the AWS EFS volume uses the expected security group.

4. Create VPC peering between the OpenShift Container Platform cluster VPC in AWS account A and the AWS EFS VPC in AWS account B:
Ensure the two VPCs are using different network CIDRs, and after creating the VPC peering, add routes in each VPC to connect the two VPC networks.

  a. Create a peering connection called, for example, “my-efs-crossaccount-peering-connection” in account B. For the local VPC ID, use the EFS-located VPC. To peer with the VPC for account A, for the VPC ID use the OpenShift Container Platform cluster VPC ID.

  b. Accept the peer connection in AWS account A.

  c. Modify the route table of each subnet (EFS-volume used subnets) in AWS account B:

     i. On the left pane, under **Virtual private cloud**, click the down arrow to expand the available options.

     ii. Under **Virtual private cloud**, click **Route tables"**.

     iii. Click the **Routes** tab.

     iv. Under **Destination**, enter 10.0.0.0/16.

     v. Under **Target**, use the peer connection type point from the created peer connection.

  d. Modify the route table of each subnet (OpenShift Container Platform cluster nodes used subnets) in AWS account A:

     i. On the left pane, under **Virtual private cloud**, click the down arrow to expand the available options.

     ii. Under **Virtual private cloud**, click **Route tables"**.

     iii. Click the **Routes** tab.

     iv. Under **Destination**, enter the CIDR for the VPC in account B, which for this example is 172.20.0.0/16.

     v. Under **Target**, use the peer connection type point from the created peer connection.

5. Create an IAM role, for example, “my-efs-acrossaccount-role” in AWS account B, which has a trust relationship with AWS account A, and add an inline AWS EFS policy with permissions to call “my-efs-acrossaccount-driver-policy”.
   This role is used by the CSI driver’s controller service running on the OpenShift Container Platform cluster in AWS account A to determine the mount targets for your file system in AWS account B.

```
# Trust relationships trusted entity trusted account A configuration on my-efs-acrossaccount-role in account B

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::301721915996:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

```
# my-cross-account-assume-policy policy attached to my-efs-acrossaccount-role in account B

{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::589722580343:role/my-efs-acrossaccount-role"
  }
}

# my-efs-acrossaccount-driver-policy attached to my-efs-acrossaccount-role in account B

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSubnets"
      ],
      "Resource": "*"
    },
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeMountTargets",
        "elasticfilesystem:DeleteAccessPoint",
        "elasticfilesystem:ClientMount",
        "elasticfilesystem:DescribeAccessPoints",
        "elasticfilesystem:ClientWrite",
        "elasticfilesystem:ClientRootAccess",
        "elasticfilesystem:DescribeFileSystems",
        "elasticfilesystem:CreateAccessPoint"
      ],
      "Resource": [
        "arn:aws:elasticfilesystem:*:589722580343:access-point/*",
        "arn:aws:elasticfilesystem:*:589722580343:file-system/*"
      ]
    }
  ]
}
```

6. In AWS account A, attach an inline policy to the IAM role of the AWS EFS CSI driver’s controller service account with the necessary permissions to perform Security Token Service (STS) assume role on the IAM role created earlier.

```
# my-cross-account-assume-policy policy attached to Openshift cluster efs csi driver user in account A

{
  "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::589722580343:role/my-efs-acrossaccount-role"
    }
}
```

7. In AWS account A, attach the AWS-managed policy “AmazonElasticFileSystemClientFullAccess” to OpenShift Container Platform cluster master role. The role name is in the form **<clusterID>-master-role** (for example, **my-0120ef-czjrl-master-role**).

8. Create a Kubernetes secret with **awsRoleArn** as the key and the role created earlier as the value:

```
$ oc -n openshift-cluster-csi-drivers create secret generic my-efs-cross-account --from-literal=awsRoleArn='arn:aws:iam::589722580343:role/my-efs-acrossaccount-role'
```

   Since the driver controller needs to get the cross account role information from the secret, you need to add the secret role binding to the AWS EFS CSI driver controller ServiceAccount (SA):

```
$ oc -n openshift-cluster-csi-drivers create role access-secrets --verb=get,list,watch --resource=secrets

$ oc -n openshift-cluster-csi-drivers create rolebinding --role=access-secrets default-to-secrets --serviceaccount=openshift-cluster-csi-drivers:aws-efs-csi-driver-controller-sa
```

9. Create a **filesystem** policy for the file system (AWS EFS volume) in account B, which allows AWS account A to perform a mount on it.

> This step is not mandatory, but can be safer for AWS EFS volume usage.

```
# EFS volume filesystem policy in account B
{
    "Version": "2012-10-17",
    "Id": "efs-policy-wizard-8089bf4a-9787-40f0-958e-bc2363012ace",
    "Statement": [
        {
            "Sid": "efs-statement-bd285549-cfa2-4f8b-861e-c372399fd238",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "elasticfilesystem:ClientRootAccess",
                "elasticfilesystem:ClientWrite",
                "elasticfilesystem:ClientMount"
            ],
            "Resource": "arn:aws:elasticfilesystem:us-east-2:589722580343:file-system/fs-091066a9bf9becbd5",
            "Condition": {
                "Bool": {
                    "elasticfilesystem:AccessedViaMountTarget": "true"
                }
            }
        },
        {
            "Sid": "efs-statement-03646e39-d80f-4daf-b396-281be1e43bab",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::589722580343:role/my-efs-acrossaccount-role"
            },
            "Action": [
                "elasticfilesystem:ClientRootAccess",
                "elasticfilesystem:ClientWrite",
                "elasticfilesystem:ClientMount"
            ],
            "Resource": "arn:aws:elasticfilesystem:us-east-2:589722580343:file-system/fs-
091066a9bf9becbd5"
        }
    ]
}
```

10. Create an AWS EFS volume storage class using a similar configuration to the following:

```
# The cross account efs volume storageClass
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-cross-account-mount-sc
provisioner: efs.csi.aws.com
mountOptions:
  - tls
parameters:
  provisioningMode: efs-ap
  fileSystemId: fs-00f6c3ae6f06388bb
  directoryPerms: "700"
  gidRangeStart: "1000"
  gidRangeEnd: "2000"
  basePath: "/account-a-data"
  csi.storage.k8s.io/provisioner-secret-name: my-efs-cross-account
  csi.storage.k8s.io/provisioner-secret-namespace: openshift-cluster-csi-drivers
volumeBindingMode: Immediate
```

5.11.6. Creating and configuring access to EFS volumes in AWS

This procedure explains how to create and configure EFS volumes in AWS so that you can use them in OpenShift Container Platform.

Prerequisites

- AWS account credentials

Procedure

To create and configure access to an EFS volume in AWS:

1. On the AWS console, open https://console.aws.amazon.com/efs.

2. Click **Create file system**:
   - Enter a name for the file system.
   - For **Virtual Private Cloud (VPC)** select your OpenShift Container Platform's' virtual private cloud (VPC).
   - Accept default settings for all other selections.
3. Wait for the volume and mount targets to finish being fully created:
   a. Go to https://console.aws.amazon.com/efs#/file-systems.
   b. Click your volume, and on the **Network** tab wait for all mount targets to become available (~1-2 minutes).
4. On the **Network** tab, copy the Security Group ID (you will need this in the next step).
5. Go to https://console.aws.amazon.com/ec2/v2/home#SecurityGroups, and find the Security Group used by the EFS volume.
6. On the **Inbound rules** tab, click **Edit inbound rules**, and then add a new rule with the following settings to allow OpenShift Container Platform nodes to access EFS volumes :
   - **Type**: NFS
   - **Protocol**: TCP
   - **Port range**: 2049
   - **Source**: Custom/IP address range of your nodes (for example: “10.0.0.0/16”)
     This step allows OpenShift Container Platform to use NFS ports from the cluster.
7. Save the rule.

5.11.7. Dynamic provisioning for Amazon Elastic File Storage

The AWS EFS CSI driver supports a different form of dynamic provisioning than other CSI drivers. It provisions new PVs as subdirectories of a pre-existing EFS volume. The PVs are independent of each other. However, they all share the same EFS volume. When the volume is deleted, all PVs provisioned out of it are deleted too. The EFS CSI driver creates an AWS Access Point for each such subdirectory. Due to AWS AccessPoint limits, you can only dynamically provision 1000 PVs from a single **StorageClass**/EFS volume.

IMPORTANT

Note that **PVC.spec.resources** is not enforced by EFS.

In the example below, you request 5 GiB of space. However, the created PV is limitless and can store any amount of data (like petabytes). A broken application, or even a rogue application, can cause significant expenses when it stores too much data on the volume.

Using monitoring of EFS volume sizes in AWS is strongly recommended.

Prerequisites

- You have created Amazon Elastic File Storage (Amazon EFS) volumes.

- You have created the AWS EFS storage class.

Procedure

To enable dynamic provisioning:

- Create a PVC (or StatefulSet or Template) as usual, referring to the **StorageClass** created previously.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test
spec:
  storageClassName: efs-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

If you have problems setting up dynamic provisioning, see AWS EFS troubleshooting.

Additional resources

- Creating and configuring access to AWS EFS volume(s)
- Creating the AWS EFS storage class

5.11.8. Creating static PVs with Amazon Elastic File Storage

It is possible to use an Amazon Elastic File Storage (Amazon EFS) volume as a single PV without any dynamic provisioning. The whole volume is mounted to pods.

Prerequisites

- You have created Amazon EFS volumes.

Procedure

- Create the PV using the following YAML file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity: 1
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-ae66151a ❷
    volumeAttributes:
      encryptInTransit: "false" ❸
```

❶ **spec.capacity** does not have any meaning and is ignored by the CSI driver. It is used only when binding to a PVC. Applications can store any amount of data to the volume.

❷ **volumeHandle** must be the same ID as the EFS volume you created in AWS. If you are providing your own access point, **volumeHandle** should be **<EFS volume ID>::<access point ID>**. For example: **fs-6e633ada::fsap-081a1d293f0004630**.

❸ If desired, you can disable encryption in transit. Encryption is enabled by default.

If you have problems setting up static PVs, see AWS EFS troubleshooting.

5.11.9. Amazon Elastic File Storage security

The following information is important for Amazon Elastic File Storage (Amazon EFS) security.

When using access points, for example, by using dynamic provisioning as described earlier, Amazon automatically replaces GIDs on files with the GID of the access point. In addition, EFS considers the user ID, group ID, and secondary group IDs of the access point when evaluating file system permissions. EFS ignores the NFS client's IDs. For more information about access points, see https://docs.aws.amazon.com/efs/latest/ug/efs-access-points.html.

As a consequence, EFS volumes silently ignore FSGroup; OpenShift Container Platform is not able to replace the GIDs of files on the volume with FSGroup. Any pod that can access a mounted EFS access point can access any file on it.

Unrelated to this, encryption in transit is enabled by default. For more information, see https://docs.aws.amazon.com/efs/latest/ug/encryption-in-transit.html.

5.11.10. Amazon Elastic File Storage troubleshooting

The following information provides guidance on how to troubleshoot issues with Amazon Elastic File Storage (Amazon EFS):

- The AWS EFS Operator and CSI driver run in namespace **openshift-cluster-csi-drivers**.
- To initiate gathering of logs of the AWS EFS Operator and CSI driver, run the following command:

```
$ oc adm must-gather
[must-gather      ] OUT Using must-gather plugin-in image: quay.io/openshift-release-
dev/ocp-v4.0-art-
dev@sha256:125f183d13601537ff15b3239df95d47f0a604da2847b561151fedd699f5e3a5
[must-gather      ] OUT namespace/openshift-must-gather-xm4wq created
[must-gather      ] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-2bd8x
created
[must-gather      ] OUT pod for plug-in image quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:125f183d13601537ff15b3239df95d47f0a604da2847b561151fedd699f5e3a5
created
```

- To show AWS EFS Operator errors, view the **ClusterCSIDriver** status:

  ```
  $ oc get clustercsidriver efs.csi.aws.com -o yaml
  ```

- If a volume cannot be mounted to a pod (as shown in the output of the following command):

  ```
  $ oc describe pod
  ...
    Type     Reason       Age    From               Message
    ----     ------       ----   ----               -------
    Normal   Scheduled    2m13s  default-scheduler  Successfully assigned default/efs-app to
  ip-10-0-135-94.ec2.internal
    Warning  FailedMount  13s    kubelet            MountVolume.SetUp failed for volume "pvc-
  d7c097e6-67ec-4fae-b968-7e7056796449" : rpc error: code = DeadlineExceeded desc =
  context deadline exceeded 1
    Warning  FailedMount  10s    kubelet            Unable to attach or mount volumes: unmounted
  volumes=[persistent-storage], unattached volumes=[persistent-storage kube-api-access-
  9j477]: timed out waiting for the condition
  ```

  1 Warning message indicating volume not mounted.

  This error is frequently caused by AWS dropping packets between an OpenShift Container Platform node and Amazon EFS.

  Check that the following are correct:

  - AWS firewall and Security Groups
  - Networking: port number and IP addresses

5.11.11. Uninstalling the AWS EFS CSI Driver Operator

All EFS PVs are inaccessible after uninstalling the AWS EFS CSI Driver Operator (a Red Hat operator).

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

To uninstall the AWS EFS CSI Driver Operator from the web console:

1. Log in to the web console.
2. Stop all applications that use AWS EFS PVs.
3. Delete all AWS EFS PVs:
   a. Click **Storage** → **PersistentVolumeClaims**.
   b. Select each PVC that is in use by the AWS EFS CSI Driver Operator, click the drop-down menu on the far right of the PVC, and then click **Delete PersistentVolumeClaims**.
4. Uninstall the AWS EFS CSI driver:

NOTE

Before you can uninstall the Operator, you must remove the CSI driver first.

a. Click **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**.

b. On the **Instances** tab, for **efs.csi.aws.com**, on the far left side, click the drop-down menu, and then click **Delete ClusterCSIDriver**.

c. When prompted, click **Delete**.

5. Uninstall the AWS EFS CSI Operator:

a. Click **Operators** → **Installed Operators**.

b. On the **Installed Operators** page, scroll or type AWS EFS CSI into the **Search by name** box to find the Operator, and then click it.

c. On the upper, right of the **Installed Operators** > **Operator details**page, click **Actions** → **Uninstall Operator**.

d. When prompted on the **Uninstall Operator** window, click the **Uninstall** button to remove the Operator from the namespace. Any applications deployed by the Operator on the cluster need to be cleaned up manually.
After uninstalling, the AWS EFS CSI Driver Operator is no longer listed in the **Installed Operators** section of the web console.

NOTE

Before you can destroy a cluster (**openshift-install destroy cluster**), you must delete the EFS volume in AWS. An OpenShift Container Platform cluster cannot be destroyed when there is an EFS volume that uses the cluster's VPC. Amazon does not allow deletion of such a VPC.

5.11.12. Additional resources

- Configuring CSI volumes

5.12. AZURE DISK CSI DRIVER OPERATOR

5.12.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Microsoft Azure Disk Storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to Azure Disk storage assets, OpenShift Container Platform installs the Azure Disk CSI Driver Operator and the Azure Disk CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *Azure Disk CSI Driver Operator* provides a storage class named **managed-csi** that you can use to create persistent volume claims (PVCs). The Azure Disk CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on-demand, eliminating the need for cluster administrators to pre-provision storage. You can disable this default storage class if desired (see Managing the default storage class).

- The *Azure Disk CSI driver* enables you to create and mount Azure Disk PVs.

5.12.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

> **IMPORTANT**
>
> OpenShift Container Platform defaults to using the CSI plugin to provision Azure Disk storage.

5.12.3. Creating a storage class with storage account type

Storage classes are used to differentiate and delineate storage levels and usages. By defining a storage class, you can obtain dynamically provisioned persistent volumes.

When creating a storage class, you can designate the storage account type. This corresponds to your Azure storage account SKU tier. Valid options are **Standard_LRS**, **Premium_LRS**, **StandardSSD_LRS**, **UltraSSD_LRS**, **Premium_ZRS**, **StandardSSD_ZRS**, and **PremiumV2_LRS**. For information about finding your Azure SKU tier, see SKU Types.

Both ZRS and PremiumV2_LRS have some region limitations. For information about these limitations, see ZRS limitations and Premium_LRS limitations.

Prerequisites

- Access to an OpenShift Container Platform cluster with administrator rights

Procedure

Use the following steps to create a storage class with a storage account type.

1. Create a storage class designating the storage account type using a YAML file similar to the following:

```
$ oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class> 1
provisioner: disk.csi.azure.com
parameters:
  skuName: <storage-class-account-type> 2
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
EOF
```

1 Storage class name.

2 Storage account type. This corresponds to your Azure storage account SKU tier:`Standard_LRS`, **Premium_LRS**, **StandardSSD_LRS**, **UltraSSD_LRS**, **Premium_ZRS**, **StandardSSD_ZRS**, **PremiumV2_LRS**.

NOTE

For PremiumV2_LRS, specify **cachingMode: None** in **storageclass.parameters**.

2. Ensure that the storage class was created by listing the storage classes:

```
$ oc get storageclass
```

Example output

```
$ oc get storageclass
NAME              PROVISIONER        RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
azurefile-csi          file.csi.azure.com   Delete         Immediate             true                    68m
managed-csi (default)   disk.csi.azure.com   Delete         WaitForFirstConsumer   true
68m
sc-prem-zrs            disk.csi.azure.com   Delete         WaitForFirstConsumer   true
4m25s 1
```

1 New storage class with storage account type.

5.12.4. User-managed encryption

The user-managed encryption feature allows you to provide keys during installation that encrypt OpenShift Container Platform node root volumes, and enables all managed storage classes to use these keys to encrypt provisioned storage volumes. You must specify the custom key in the **platform.<cloud_type>.defaultMachinePlatform** field in the install-config YAML file.

This features supports the following storage types:

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk storage
- Google Cloud Platform (GCP) persistent disk (PD) storage

NOTE

If the OS (root) disk is encrypted, and there is no encrypted key defined in the storage class, Azure Disk CSI driver uses the OS disk encryption key by default to encrypt provisioned storage volumes.

For information about installing with user-managed encryption for Azure, see Enabling user-managed encryption for Azure.

5.12.5. Machine sets that deploy machines with ultra disks using PVCs

You can create a machine set running on Azure that deploys machines with ultra disks. Ultra disks are high-performance storage that are intended for use with the most demanding data workloads.

Both the in-tree plugin and CSI driver support using PVCs to enable ultra disks. You can also deploy machines with ultra disks as data disks without creating a PVC.

Additional resources

- Microsoft Azure ultra disks documentation
- Machine sets that deploy machines on ultra disks using in-tree PVCs
- Machine sets that deploy machines on ultra disks as data disks

5.12.5.1. Creating machines with ultra disks by using machine sets

You can deploy machines with ultra disks on Azure by editing your machine set YAML file.

Prerequisites

- Have an existing Microsoft Azure cluster.

Procedure

1. Copy an existing Azure **MachineSet** custom resource (CR) and edit it by running the following command:

   ```
   $ oc edit machineset <machine-set-name>
   ```

   where **<machine-set-name>** is the machine set that you want to provision machines with ultra disks.

2. Add the following lines in the positions indicated:

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   spec:
     template:
       spec:
         metadata:
           labels:
             disk: ultrassd 1
         providerSpec:
           value:
             ultraSSDCapability: Enabled 2
   ```

   1 Specify a label to use to select a node that is created by this machine set. This procedure uses **disk.ultrassd** for this value.

   2 These lines enable the use of ultra disks.

3. Create a machine set using the updated configuration by running the following command:

```
$ oc create -f <machine-set-name>.yaml
```

4. Create a storage class that contains the following YAML definition:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ultra-disk-sc 1
parameters:
  cachingMode: None
  diskIopsReadWrite: "2000" 2
  diskMbpsReadWrite: "320" 3
  kind: managed
  skuname: UltraSSD_LRS
provisioner: disk.csi.azure.com 4
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer 5
```

1. Specify the name of the storage class. This procedure uses **ultra-disk-sc** for this value.
2. Specify the number of IOPS for the storage class.
3. Specify the throughput in MBps for the storage class.
4. For Azure Kubernetes Service (AKS) version 1.21 or later, use **disk.csi.azure.com**. For earlier versions of AKS, use **kubernetes.io/azure-disk**.
5. Optional: Specify this parameter to wait for the creation of the pod that will use the disk.

5. Create a persistent volume claim (PVC) to reference the **ultra-disk-sc** storage class that contains the following YAML definition:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ultra-disk 1
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: ultra-disk-sc 2
  resources:
    requests:
      storage: 4Gi 3
```

1. Specify the name of the PVC. This procedure uses **ultra-disk** for this value.
2. This PVC references the **ultra-disk-sc** storage class.
3. Specify the size for the storage class. The minimum value is **4Gi**.

6. Create a pod that contains the following YAML definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-ultra
spec:
  nodeSelector:
    disk: ultrassd 1
  containers:
  - name: nginx-ultra
    image: alpine:latest
    command:
      - "sleep"
      - "infinity"
    volumeMounts:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: ultra-disk 2
```

1 Specify the label of the machine set that enables the use of ultra disks. This procedure uses **disk.ultrassd** for this value.

2 This pod references the **ultra-disk** PVC.

Verification

1. Validate that the machines are created by running the following command:

   ```
   $ oc get machines
   ```

   The machines should be in the **Running** state.

2. For a machine that is running and has a node attached, validate the partition by running the following command:

   ```
   $ oc debug node/<node-name> -- chroot /host lsblk
   ```

   In this command, **oc debug node/<node-name>** starts a debugging shell on the node **<node-name>** and passes a command with **--**. The passed command **chroot /host** provides access to the underlying host OS binaries, and **lsblk** shows the block devices that are attached to the host OS machine.

Next steps

- To use an ultra disk from within a pod, create a workload that uses the mount point. Create a YAML file similar to the following example:

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: ssd-benchmark1
  ```

```
spec:
  containers:
  - name: ssd-benchmark1
    image: nginx
    ports:
      - containerPort: 80
        name: "http-server"
    volumeMounts:
    - name: lun0p1
      mountPath: "/tmp"
  volumes:
    - name: lun0p1
      hostPath:
        path: /var/lib/lun0p1
        type: DirectoryOrCreate
  nodeSelector:
    disktype: ultrassd
```

5.12.5.2. Troubleshooting resources for machine sets that enable ultra disks

Use the information in this section to understand and recover from issues you might encounter.

5.12.5.2.1. Unable to mount a persistent volume claim backed by an ultra disk

If there is an issue mounting a persistent volume claim backed by an ultra disk, the pod becomes stuck in the **ContainerCreating** state and an alert is triggered.

For example, if the **additionalCapabilities.ultraSSDEnabled** parameter is not set on the machine that backs the node that hosts the pod, the following error message appears:

```
StorageAccountType UltraSSD_LRS can be used only when additionalCapabilities.ultraSSDEnabled
is set.
```

- To resolve this issue, describe the pod by running the following command:

  ```
  $ oc -n <stuck_pod_namespace> describe pod <stuck_pod_name>
  ```

5.12.6. Additional resources

- Persistent storage using Azure Disk
- Configuring CSI volumes

5.13. AZURE FILE CSI DRIVER OPERATOR

5.13.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) by using the Container Storage Interface (CSI) driver for Microsoft Azure File Storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to Azure File storage assets, OpenShift Container Platform installs the Azure File CSI Driver Operator and the Azure File CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *Azure File CSI Driver Operator* provides a storage class that is named **azurefile-csi** that you can use to create persistent volume claims (PVCs). You can disable this default storage class if desired (see Managing the default storage class).
- The *Azure File CSI driver* enables you to create and mount Azure File PVs. The Azure File CSI driver supports dynamic volume provisioning by allowing storage volumes to be created on-demand, eliminating the need for cluster administrators to pre-provision storage.

Azure File CSI Driver Operator does not support:

- Virtual hard disks (VHD)
- Running on nodes with Federal Information Processing Standard (FIPS) mode enabled for Server Message Block (SMB) file share. However, Network File System (NFS) does support FIPS mode.

For more information about supported features, see Supported CSI drivers and features.

5.13.2. NFS support

OpenShift Container Platform 4.14, and later, supports Azure File Container Storage Interface (CSI) Driver Operator with Network File System (NFS) with the following caveats:

- Creating pods with Azure File NFS volumes that are scheduled to the control plane node causes the mount to be denied.
  To work around this issue: If your control plane nodes are schedulable, and the pods can run on worker nodes, use **nodeSelector** or Affinity to schedule the pod in worker nodes.
- FS Group policy behavior:

  IMPORTANT

  Azure File CSI with NFS does not honor the fsGroupChangePolicy requested by pods. Azure File CSI with NFS applies a default OnRootMismatch FS Group policy regardless of the policy requested by the pod.

- The Azure File CSI Operator does not automatically create a storage class for NFS. You must create it manually. Use a file similar to the following:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> ①
provisioner: file.csi.azure.com ②
parameters:
  protocol: nfs ③
  skuName: Premium_LRS  # available values: Premium_LRS, Premium_ZRS
mountOptions:
  - nconnect=4
```

① Storage class name.

2 Specifies the Azure File CSI provider.

3 Specifies NFS as the storage backend protocol.

5.13.3. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

Additional resources

- Persistent storage using Azure File
- Configuring CSI volumes

5.14. AZURE STACK HUB CSI DRIVER OPERATOR

5.14.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Azure Stack Hub Storage. Azure Stack Hub, which is part of the Azure Stack portfolio, allows you to run apps in an on-premises environment and deliver Azure services in your datacenter.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to Azure Stack Hub storage assets, OpenShift Container Platform installs the Azure Stack Hub CSI Driver Operator and the Azure Stack Hub CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *Azure Stack Hub CSI Driver Operator* provides a storage class (**managed-csi**), with "Standard_LRS" as the default storage account type, that you can use to create persistent volume claims (PVCs). The Azure Stack Hub CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on-demand, eliminating the need for cluster administrators to pre-provision storage.
- The *Azure Stack Hub CSI driver* enables you to create and mount Azure Stack Hub PVs.

5.14.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.14.3. Additional resources

- Configuring CSI volumes

5.15. GCP PD CSI DRIVER OPERATOR

5.15.1. Overview

OpenShift Container Platform can provision persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Google Cloud Platform (GCP) persistent disk (PD) storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a Container Storage Interface (CSI) Operator and driver.

To create CSI-provisioned persistent volumes (PVs) that mount to GCP PD storage assets, OpenShift Container Platform installs the GCP PD CSI Driver Operator and the GCP PD CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- **GCP PD CSI Driver Operator**: By default, the Operator provides a storage class that you can use to create PVCs. You can disable this default storage class if desired (see Managing the default storage class). You also have the option to create the GCP PD storage class as described in Persistent storage using GCE Persistent Disk .

- **GCP PD driver**: The driver enables you to create and mount GCP PD PVs.

IMPORTANT

OpenShift Container Platform defaults to using the CSI plugin to provision GCP PD storage.

5.15.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.15.3. GCP PD CSI driver storage class parameters

The Google Cloud Platform (GCP) persistent disk (PD) Container Storage Interface (CSI) driver uses the CSI **external-provisioner** sidecar as a controller. This is a separate helper container that is deployed with the CSI driver. The sidecar manages persistent volumes (PVs) by triggering the **CreateVolume** operation.

The GCP PD CSI driver uses the **csi.storage.k8s.io/fstype** parameter key to support dynamic provisioning. The following table describes all the GCP PD CSI storage class parameters that are supported by OpenShift Container Platform.

Table 5.5. CreateVolume Parameters

Parameter	Values	Default	Description

Parameter	Values	Default	Description
type	**pd-ssd** or **pd-standard**	**pd-standard**	Allows you to choose between standard PVs or solid-state-drive PVs.
replication-type	**none** or **regional-pd**	**none**	Allows you to choose between zonal or regional PVs.
disk-encryption-kms-key	Fully qualified resource identifier for the key to use to encrypt new disks.	Empty string	Uses customer-managed encryption keys (CMEK) to encrypt new disks.

5.15.4. Creating a custom-encrypted persistent volume

When you create a **PersistentVolumeClaim** object, OpenShift Container Platform provisions a new persistent volume (PV) and creates a **PersistentVolume** object. You can add a custom encryption key in Google Cloud Platform (GCP) to protect a PV in your cluster by encrypting the newly created PV.

For encryption, the newly attached PV that you create uses customer-managed encryption keys (CMEK) on a cluster by using a new or existing Google Cloud Key Management Service (KMS) key.

Prerequisites

- You are logged in to a running OpenShift Container Platform cluster.
- You have created a Cloud KMS key ring and key version.

For more information about CMEK and Cloud KMS resources, see Using customer-managed encryption keys (CMEK).

Procedure

To create a custom-encrypted PV, complete the following steps:

1. Create a storage class with the Cloud KMS key. The following example enables dynamic provisioning of encrypted volumes:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-gce-pd-cmek
provisioner: pd.csi.storage.gke.io
volumeBindingMode: "WaitForFirstConsumer"
allowVolumeExpansion: true
parameters:
  type: pd-standard
  disk-encryption-kms-key: projects/<key-project-id>/locations/<location>/keyRings/<key-ring>/cryptoKeys/<key> 1
```

1 This field must be the resource identifier for the key that will be used to encrypt new disks. Values are case-sensitive. For more information about providing key ID values, see Retrieving a resource's ID and Getting a Cloud KMS resource ID .

NOTE

You cannot add the **disk-encryption-kms-key** parameter to an existing storage class. However, you can delete the storage class and recreate it with the same name and a different set of parameters. If you do this, the provisioner of the existing class must be **pd.csi.storage.gke.io**.

2. Deploy the storage class on your OpenShift Container Platform cluster using the **oc** command:

```
$ oc describe storageclass csi-gce-pd-cmek
```

Example output

```
Name:                csi-gce-pd-cmek
IsDefaultClass:        No
Annotations:           None
Provisioner:           pd.csi.storage.gke.io
Parameters:            disk-encryption-kms-key=projects/key-project-
id/locations/location/keyRings/ring-name/cryptoKeys/key-name,type=pd-standard
AllowVolumeExpansion:  true
MountOptions:          none
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                none
```

3. Create a file named **pvc.yaml** that matches the name of your storage class object that you created in the previous step:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: podpvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-gce-pd-cmek
  resources:
    requests:
      storage: 6Gi
```

NOTE

If you marked the new storage class as default, you can omit the **storageClassName** field.

4. Apply the PVC on your cluster:

```
$ oc apply -f pvc.yaml
```

5. Get the status of your PVC and verify that it is created and bound to a newly provisioned PV:

```
$ oc get pvc
```

Example output

```
NAME      STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS     AGE
podpvc    Bound     pvc-e36abf50-84f3-11e8-8538-42010a800002   10Gi       RWO          csi-
gce-pd-cmek  9s
```

NOTE

If your storage class has the **volumeBindingMode** field set to **WaitForFirstConsumer**, you must create a pod to use the PVC before you can verify it.

Your CMEK-protected PV is now ready to use with your OpenShift Container Platform cluster.

5.15.5. User-managed encryption

The user-managed encryption feature allows you to provide keys during installation that encrypt OpenShift Container Platform node root volumes, and enables all managed storage classes to use these keys to encrypt provisioned storage volumes. You must specify the custom key in the **platform.<cloud_type>.defaultMachinePlatform** field in the install-config YAML file.

This features supports the following storage types:

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk storage
- Google Cloud Platform (GCP) persistent disk (PD) storage

For information about installing with user-managed encryption for GCP PD, see Installation configuration parameters.

5.15.6. Additional resources

- Persistent storage using GCE Persistent Disk
- Configuring CSI volumes

5.16. GOOGLE COMPUTE PLATFORM FILESTORE CSI DRIVER OPERATOR

5.16.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Google Compute Platform (GCP) Filestore Storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to GCP Filestore Storage assets, you install the GCP Filestore CSI Driver Operator and the GCP Filestore CSI driver in the **openshift-cluster-csi-drivers** namespace.

- The *GCP Filestore CSI Driver Operator* does not provide a storage class by default, but you can create one if needed. The GCP Filestore CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on demand, eliminating the need for cluster administrators to pre-provision storage.
- The *GCP Filestore CSI driver* enables you to create and mount GCP Filestore PVs.

5.16.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.16.3. Installing the GCP Filestore CSI Driver Operator

The Google Compute Platform (GCP) Filestore Container Storage Interface (CSI) Driver Operator is not installed in OpenShift Container Platform by default. Use the following procedure to install the GCP Filestore CSI Driver Operator in your cluster.

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

To install the GCP Filestore CSI Driver Operator from the web console:

1. Log in to the web console.
2. Enable the Filestore API in the GCE project by running the following command:

   ```
   $ gcloud services enable file.googleapis.com  --project <my_gce_project> 1
   ```

   1 Replace **<my_gce_project>** with your Google Cloud project.

   You can also do this using Google Cloud web console.
3. Install the GCP Filestore CSI Operator:
   a. Click **Operators** → **OperatorHub**.
   b. Locate the GCP Filestore CSI Operator by typing **GCP Filestore** in the filter box.
   c. Click the **GCP Filestore CSI Driver Operator** button.
   d. On the **GCP Filestore CSI Driver Operator** page, click **Install**.
   e. On the **Install Operator** page, ensure that:
      - **All namespaces on the cluster (default)** is selected.
      - **Installed Namespace** is set to **openshift-cluster-csi-drivers**.

f. Click **Install**.
After the installation finishes, the GCP Filestore CSI Operator is listed in the **Installed Operators** section of the web console.

4. Install the GCP Filestore CSI Driver:

    a. Click **administration → CustomResourceDefinitions → ClusterCSIDriver**.

    b. On the **Instances** tab, click **Create ClusterCSIDriver**.
    Use the following YAML file:

    ```
    apiVersion: operator.openshift.io/v1
    kind: ClusterCSIDriver
    metadata:
        name: filestore.csi.storage.gke.io
    spec:
      managementState: Managed
    ```

    c. Click **Create**.

    d. Wait for the following Conditions to change to a "true" status:

    - GCPFilestoreDriverCredentialsRequestControllerAvailable
    - GCPFilestoreDriverNodeServiceControllerAvailable
    - GCPFilestoreDriverControllerServiceControllerAvailable

Additional resources

- Enabling an API in your Google Cloud .
- Enabling an API using the Google Cloud web console .

5.16.4. Creating a storage class for GCP Filestore Storage

After installing the Operator, you should create a storage class for dynamic provisioning of Google Compute Platform (GCP) Filestore volumes.

Prerequisites

- You are logged in to the running OpenShift Container Platform cluster.

Procedure

To create a storage class:

1. Create a storage class using the following example YAML file:

    Example YAML file

    ```
    kind: StorageClass
    apiVersion: storage.k8s.io/v1
    metadata:
      name: filestore-csi
    provisioner: filestore.csi.storage.gke.io
    ```

```
parameters:
  network: network-name 1
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

1 Specify the name of the GCP virtual private cloud (VPC) network where Filestore instances should be created in.

2. Specify the name of the VPC network where Filestore instances should be created in.
   It is recommended to specify the VPC network that the Filestore instances should be created in. If no VPC network is specified, the Container Storage Interface (CSI) driver tries to create the instances in the default VPC network of the project. On IPI installations, the VPC network name is typically the cluster name with the suffix "-network". However, on UPI installations, the VPC network name can be any value chosen by the user.

   You can find out the VPC network name by inspecting the **MachineSets** objects with the following command:

   ```
   $ oc -n openshift-machine-api get machinesets -o yaml | grep "network:"
               - network: gcp-filestore-network
   (...)
   ```

   In this example, the VPC network name in this cluster is "gcp-filestore-network".

5.16.5. Destroying clusters and GCP Filestore

Typically, if you destroy a cluster, the OpenShift Container Platform installer deletes all of the cloud resources that belong to that cluster. However, when a cluster is destroyed, Google Compute Platform (GCP) Filestore instances are not automatically deleted, so you must manually delete all persistent volume claims (PVCs) that use the Filestore storage class before destroying the cluster.

Procedure

To delete all GCP Filestore PVCs:

1. List all PVCs that were created using the storage class **filestore-csi**:

   ```
   $ oc get pvc -o json -A | jq -r '.items[] | select(.spec.storageClassName == "filestore-csi")
   ```

2. Delete all of the PVCs listed by the previous command:

   ```
   $ oc delete <pvc-name> 1
   ```

   1 Replace <pvc-name> with the name of any PVC that you need to delete.

5.16.6. Additional resources

- Configuring CSI volumes

5.17. IBM VPC BLOCK CSI DRIVER OPERATOR

5.17.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for IBM® Virtual Private Cloud (VPC) Block Storage.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to IBM® VPC Block storage assets, OpenShift Container Platform installs the IBM® VPC Block CSI Driver Operator and the IBM® VPC Block CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *IBM® VPC Block CSI Driver Operator* provides three storage classes named **ibmc-vpc-block-10iops-tier** (default), **ibmc-vpc-block-5iops-tier**, and **ibmc-vpc-block-custom** for different tiers that you can use to create persistent volume claims (PVCs). The IBM® VPC Block CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on demand, eliminating the need for cluster administrators to pre-provision storage. You can disable this default storage class if desired (see Managing the default storage class).

- The *IBM® VPC Block CSI driver* enables you to create and mount IBM® VPC Block PVs.

5.17.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

Additional resources

- Configuring CSI volumes

5.18. IBM POWER VIRTUAL SERVER BLOCK CSI DRIVER OPERATOR

5.18.1. Introduction

The IBM Power® Virtual Server Block CSI Driver will be installed through IBM Power® Virtual Server Block CSI Driver Operator and the operator is based on libarary-go. The OpenShift library-go is a collection of functions that allow us to build OpenShift operators easily. Most of the functionality of a CSI driver operator is already available there. The IBM Power® Virtual Server Block CSI Driver Operator is installed by the cluster-storage-operator. The Cluster-storage-operator installs the IBM Power® Virtual Server Block CSI Driver Operator if the Platform type is Power Virtual Servers.

5.18.2. Overview

OpenShift Container Platform can provision persistent volumes (PVs) by using the Container Storage Interface (CSI) driver for IBM Power® Virtual Server Block Storage.

IMPORTANT

IBM Power Virtual Server Block CSI Driver Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Familiarity with persistent storage and configuring CSI volumes is helpful when working with a CSI Operator and driver.

To create CSI-provisioned PVs that mount to IBM Power® Virtual Server Block storage assets, OpenShift Container Platform installs the IBM Power® Virtual Server Block CSI Driver Operator and the IBM Power® Virtual Server Block CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- The *IBM Power® Virtual Server Block CSI Driver Operator* provides two storage classes named **ibm-powervs-tier1** (default), and **ibm-powervs-tier3** for different tiers that you can use to create persistent volume claims (PVCs). The IBM Power® Virtual Server Block CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on demand, eliminating the need for cluster administrators to pre-provision storage.

- The *IBM Power® Virtual Server Block CSI driver* allows you to create and mount IBM Power® Virtual Server Block PVs.

5.18.3. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

Additional resources

- Configuring CSI volumes

5.19. OPENSTACK CINDER CSI DRIVER OPERATOR

5.19.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for OpenStack Cinder.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a Container Storage Interface (CSI) Operator and driver.

To create CSI-provisioned PVs that mount to OpenStack Cinder storage assets, OpenShift Container Platform installs the OpenStack Cinder CSI Driver Operator and the OpenStack Cinder CSI driver in the **openshift-cluster-csi-drivers** namespace.

- The *OpenStack Cinder CSI Driver Operator* provides a CSI storage class that you can use to create PVCs. You can disable this default storage class if desired (see Managing the default storage class).
- The *OpenStack Cinder CSI driver* enables you to create and mount OpenStack Cinder PVs.

For OpenShift Container Platform, automatic migration from OpenStack Cinder in-tree to the CSI driver is available as a Technology Preview (TP) feature. With migration enabled, volumes provisioned using the existing in-tree plugin are automatically migrated to use the OpenStack Cinder CSI driver. For more information, see CSI automatic migration feature .

5.19.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

> **IMPORTANT**
>
> OpenShift Container Platform defaults to using the CSI plugin to provision Cinder storage.

5.19.3. Making OpenStack Cinder CSI the default storage class

The OpenStack Cinder CSI driver uses the **cinder.csi.openstack.org** parameter key to support dynamic provisioning.

To enable OpenStack Cinder CSI provisioning in OpenShift Container Platform, it is recommended that you overwrite the default in-tree storage class with **standard-csi**. Alternatively, you can create the persistent volume claim (PVC) and specify the storage class as "standard-csi".

In OpenShift Container Platform, the default storage class references the in-tree Cinder driver. However, with CSI automatic migration enabled, volumes created using the default storage class actually use the CSI driver.

Procedure

Use the following steps to apply the **standard-csi** storage class by overwriting the default in-tree storage class.

1. List the storage class:

```
$ oc get storageclass
```

Example output

```
NAME                PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
standard(default)     cinder.csi.openstack.org   Delete          WaitForFirstConsumer   true
46h
standard-csi          kubernetes.io/cinder      Delete          WaitForFirstConsumer   true
46h
```

2. Change the value of the annotation **storageclass.kubernetes.io/is-default-class** to **false** for the default storage class, as shown in the following example:

   ```
   $ oc patch storageclass standard -p '{"metadata": {"annotations":
   {"storageclass.kubernetes.io/is-default-class": "false"}}}'
   ```

3. Make another storage class the default by adding or modifying the annotation as **storageclass.kubernetes.io/is-default-class=true**.

   ```
   $ oc patch storageclass standard-csi -p '{"metadata": {"annotations":
   {"storageclass.kubernetes.io/is-default-class": "true"}}}'
   ```

4. Verify that the PVC is now referencing the CSI storage class by default:

   ```
   $ oc get storageclass
   ```

   Example output

   ```
   NAME                PROVISIONER              RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   standard            kubernetes.io/cinder      Delete          WaitForFirstConsumer   true
   46h
   standard-csi(default)  cinder.csi.openstack.org   Delete          WaitForFirstConsumer   true
   46h
   ```

5. Optional: You can define a new PVC without having to specify the storage class:

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: cinder-claim
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 1Gi
   ```

   A PVC that does not specify a specific storage class is automatically provisioned by using the default storage class.

6. Optional: After the new file has been configured, create it in your cluster:

   ```
   $ oc create -f cinder-claim.yaml
   ```

Additional resources

- Configuring CSI volumes

5.20. OPENSTACK MANILA CSI DRIVER OPERATOR

5.20.1. Overview

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for the OpenStack Manila shared file system service.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a Container Storage Interface (CSI) Operator and driver.

To create CSI-provisioned PVs that mount to Manila storage assets, OpenShift Container Platform installs the Manila CSI Driver Operator and the Manila CSI driver by default on any OpenStack cluster that has the Manila service enabled.

- The *Manila CSI Driver Operator* creates the required storage class that is needed to create PVCs for all available Manila share types. The Operator is installed in the **openshift-cluster-csi-drivers** namespace.

- The *Manila CSI driver* enables you to create and mount Manila PVs. The driver is installed in the **openshift-manila-csi-driver** namespace.

5.20.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.20.3. Manila CSI Driver Operator limitations

The following limitations apply to the Manila Container Storage Interface (CSI) Driver Operator:

Only NFS is supported

OpenStack Manila supports many network-attached storage protocols, such as NFS, CIFS, and CEPHFS, and these can be selectively enabled in the OpenStack cloud. The Manila CSI Driver Operator in OpenShift Container Platform only supports using the NFS protocol. If NFS is not available and enabled in the underlying OpenStack cloud, you cannot use the Manila CSI Driver Operator to provision storage for OpenShift Container Platform.

Snapshots are not supported if the back end is CephFS-NFS

To take snapshots of persistent volumes (PVs) and revert volumes to snapshots, you must ensure that the Manila share type that you are using supports these features. A Red Hat OpenStack administrator must enable support for snapshots (**share type extra-spec snapshot_support**) and for creating shares from snapshots (**share type extra-spec create_share_from_snapshot_support**) in the share type associated with the storage class you intend to use.

FSGroups are not supported

Since Manila CSI provides shared file systems for access by multiple readers and multiple writers, it does not support the use of FSGroups. This is true even for persistent volumes created with the ReadWriteOnce access mode. It is therefore important not to specify the **fsType** attribute in any storage class that you manually create for use with Manila CSI Driver.

IMPORTANT

In Red Hat OpenStack Platform 16.x and 17.x, the Shared File Systems service (Manila) with CephFS through NFS fully supports serving shares to OpenShift Container Platform through the Manila CSI. However, this solution is not intended for massive scale. Be sure to review important recommendations in CephFS NFS Manila-CSI Workload Recommendations for Red Hat OpenStack Platform.

5.20.4. Dynamically provisioning Manila CSI volumes

OpenShift Container Platform installs a storage class for each available Manila share type.

The YAML files that are created are completely decoupled from Manila and from its Container Storage Interface (CSI) plugin. As an application developer, you can dynamically provision ReadWriteMany (RWX) storage and deploy pods with applications that safely consume the storage using YAML manifests.

You can use the same pod and persistent volume claim (PVC) definitions on-premise that you use with OpenShift Container Platform on AWS, GCP, Azure, and other platforms, with the exception of the storage class reference in the PVC definition.

NOTE

Manila service is optional. If the service is not enabled in Red Hat OpenStack Platform (RHOSP), the Manila CSI driver is not installed and the storage classes for Manila are not created.

Prerequisites

- RHOSP is deployed with appropriate Manila share infrastructure so that it can be used to dynamically provision and mount volumes in OpenShift Container Platform.

Procedure (UI)

To dynamically create a Manila CSI volume using the web console:

1. In the OpenShift Container Platform console, click **Storage** → **Persistent Volume Claims**.
2. In the persistent volume claims overview, click **Create Persistent Volume Claim**.
3. Define the required options on the resulting page.
   a. Select the appropriate storage class.
   b. Enter a unique name for the storage claim.
   c. Select the access mode to specify read and write access for the PVC you are creating.

      IMPORTANT

      Use RWX if you want the persistent volume (PV) that fulfills this PVC to be mounted to multiple pods on multiple nodes in the cluster.
4. Define the size of the storage claim.
5. Click **Create** to create the persistent volume claim and generate a persistent volume.

Procedure (CLI)

To dynamically create a Manila CSI volume using the command-line interface (CLI):

1. Create and save a file with the **PersistentVolumeClaim** object described by the following YAML:

   pvc-manila.yaml

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: pvc-manila
   spec:
     accessModes: 1
       - ReadWriteMany
     resources:
       requests:
         storage: 10Gi
     storageClassName: csi-manila-gold 2
   ```

   1 Use RWX if you want the persistent volume (PV) that fulfills this PVC to be mounted to multiple pods on multiple nodes in the cluster.

   2 The name of the storage class that provisions the storage back end. Manila storage classes are provisioned by the Operator and have the **csi-manila-** prefix.

2. Create the object you saved in the previous step by running the following command:

   ```
   $ oc create -f pvc-manila.yaml
   ```

   A new PVC is created.

3. To verify that the volume was created and is ready, run the following command:

   ```
   $ oc get pvc pvc-manila
   ```

   The **pvc-manila** shows that it is **Bound**.

You can now use the new PVC to configure a pod.

Additional resources

- Configuring CSI volumes

5.21. SECRETS STORE CSI DRIVER

5.21.1. Overview

Kubernetes secrets are stored with Base64 encoding. etcd provides encryption at rest for these secrets, but when secrets are retrieved, they are decrypted and presented to the user. If role-based access control is not configured properly on your cluster, anyone with API or etcd access can retrieve or modify a secret. Additionally, anyone who is authorized to create a pod in a namespace can use that access to read any secret in that namespace.

To store and manage your secrets securely, you can configure the OpenShift Container Platform Secrets Store Container Storage Interface (CSI) Driver Operator to mount secrets from an external secret management system, such as Azure Key Vault, by using a provider plugin. Applications can then use the secret, but the secret does not persist on the system after the application pod is destroyed.

The Secrets Store CSI Driver Operator, **secrets-store.csi.k8s.io**, enables OpenShift Container Platform to mount multiple secrets, keys, and certificates stored in enterprise-grade external secrets stores into pods as a volume. The Secrets Store CSI Driver Operator communicates with the provider using gRPC to fetch the mount contents from the specified external secrets store. After the volume is attached, the data in it is mounted into the container's file system. Secrets store volumes are mounted in-line.

For more information about CSI inline volumes, see CSI inline ephemeral volumes .

IMPORTANT

The Secrets Store CSI Driver Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI driver.

5.21.1.1. Secrets store providers

The following secrets store providers are available for use with the Secrets Store CSI Driver Operator:

- AWS Secrets Manager
- AWS Systems Manager Parameter Store
- Azure Key Vault

5.21.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.21.3. Installing the Secrets Store CSI driver

Prerequisites

- Access to the OpenShift Container Platform web console.
- Administrator access to the cluster.

Procedure

To install the Secrets Store CSI driver:

1. Install the Secrets Store CSI Driver Operator:
   a. Log in to the web console.
   b. Click **Operators → OperatorHub**.
   c. Locate the Secrets Store CSI Driver Operator by typing "Secrets Store CSI" in the filter box.
   d. Click the **Secrets Store CSI Driver Operator** button.
   e. On the **Secrets Store CSI Driver Operator** page, click **Install**.
   f. On the **Install Operator** page, ensure that:
      - **All namespaces on the cluster (default)** is selected.
      - **Installed Namespace** is set to **openshift-cluster-csi-drivers**.
   g. Click **Install**.
      After the installation finishes, the Secrets Store CSI Driver Operator is listed in the **Installed Operators** section of the web console.
2. Create the **ClusterCSIDriver** instance for the driver (**secrets-store.csi.k8s.io**):
   a. Click **Administration → CustomResourceDefinitions → ClusterCSIDriver**.
   b. On the **Instances** tab, click **Create ClusterCSIDriver**.
      Use the following YAML file:

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
  name: secrets-store.csi.k8s.io
spec:
 managementState: Managed
```

   c. Click **Create**.

Next steps

- Mounting secrets from an external secrets store to a CSI volume

5.21.4. Uninstalling the Secrets Store CSI Driver Operator

Prerequisites

- Access to the OpenShift Container Platform web console.

- Administrator access to the cluster.

Procedure

To uninstall the Secrets Store CSI Driver Operator:

1. Stop all application pods that use the **secrets-store.csi.k8s.io** provider.
2. Remove any third-party provider plug-in for your chosen secret store.
3. Remove the Container Storage Interface (CSI) driver and associated manifests:
   a. Click **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**.
   b. On the **Instances** tab, for **secrets-store.csi.k8s.io**, on the far left side, click the drop-down menu, and then click **Delete ClusterCSIDriver**.
   c. When prompted, click **Delete**.
4. Verify that the CSI driver pods are no longer running.
5. Uninstall the Secrets Store CSI Driver Operator:

   NOTE

   Before you can uninstall the Operator, you must remove the CSI driver first.

   a. Click **Operators** → **Installed Operators**.
   b. On the **Installed Operators** page, scroll or type "Secrets Store CSI" into the **Search by name** box to find the Operator, and then click it.
   c. On the upper, right of the **Installed Operators** > **Operator details** page, click **Actions** → **Uninstall Operator**.
   d. When prompted on the **Uninstall Operator** window, click the **Uninstall** button to remove the Operator from the namespace. Any applications deployed by the Operator on the cluster need to be cleaned up manually.
      After uninstalling, the Secrets Store CSI Driver Operator is no longer listed in the **Installed Operators** section of the web console.

5.21.5. Additional resources

- Configuring CSI volumes

5.22. VMWARE VSPHERE CSI DRIVER OPERATOR

5.22.1. Overview

OpenShift Container Platform can provision persistent volumes (PVs) using the Container Storage Interface (CSI) VMware vSphere driver for Virtual Machine Disk (VMDK) volumes.

Familiarity with persistent storage and configuring CSI volumes is recommended when working with a CSI Operator and driver.

To create CSI-provisioned persistent volumes (PVs) that mount to vSphere storage assets, OpenShift Container Platform installs the vSphere CSI Driver Operator and the vSphere CSI driver by default in the **openshift-cluster-csi-drivers** namespace.

- **vSphere CSI Driver Operator**: The Operator provides a storage class, called **thin-csi**, that you can use to create persistent volumes claims (PVCs). The vSphere CSI Driver Operator supports dynamic volume provisioning by allowing storage volumes to be created on-demand, eliminating the need for cluster administrators to pre-provision storage. You can disable this default storage class if desired (see Managing the default storage class).

- **vSphere CSI driver**: The driver enables you to create and mount vSphere PVs. In OpenShift Container Platform 4.14, the driver version is 3.0.2. The vSphere CSI driver supports all of the file systems supported by the underlying Red Hat Core OS release, including XFS and Ext4. For more information about supported file systems, see Overview of available file systems.

IMPORTANT

OpenShift Container Platform defaults to using the CSI plugin to provision vSphere storage.

NOTE

The vSphere CSI Driver supports dynamic and static provisioning. When using static provisioning in the PV specifications, do not use the key **storage.kubernetes.io/csiProvisionerIdentity** in **csi.volumeAttributes** because this key indicates dynamically provisioned PVs.

5.22.2. About CSI

Storage vendors have traditionally provided storage drivers as part of Kubernetes. With the implementation of the Container Storage Interface (CSI), third-party providers can instead deliver storage plugins using a standard interface without ever having to change the core Kubernetes code.

CSI Operators give OpenShift Container Platform users storage options, such as volume snapshots, that are not possible with in-tree volume plugins.

5.22.3. vSphere storage policy

The vSphere CSI Driver Operator storage class uses vSphere's storage policy. OpenShift Container Platform automatically creates a storage policy that targets datastore configured in cloud configuration:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: thin-csi
provisioner: csi.vsphere.vmware.com
parameters:
  StoragePolicyName: "$openshift-storage-policy-xxxx"
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: false
reclaimPolicy: Delete
```

5.22.4. ReadWriteMany vSphere volume support

If the underlying vSphere environment supports the vSAN file service, then vSphere Container Storage Interface (CSI) Driver Operator installed by OpenShift Container Platform supports provisioning of ReadWriteMany (RWX) volumes. If vSAN file service is not configured, then ReadWriteOnce (RWO) is the only access mode available. If you do not have vSAN file service configured, and you request RWX, the volume fails to get created and an error is logged.

For more information about configuring the vSAN file service in your environment, see vSAN File Service.

You can request RWX volumes by making the following persistent volume claim (PVC):

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
     - ReadWriteMany
  storageClassName: thin-csi
```

Requesting a PVC of the RWX volume type should result in provisioning of persistent volumes (PVs) backed by the vSAN file service.

5.22.5. VMware vSphere CSI Driver Operator requirements

To install the vSphere CSI Driver Operator, the following requirements must be met:

- VMware vSphere version 7.0 Update 2 or later
- vCenter 7.0 Update 2 or later
- Virtual machines of hardware version 15 or later
- No third-party vSphere CSI driver already installed in the cluster

If a third-party vSphere CSI driver is present in the cluster, OpenShift Container Platform does not overwrite it. The presence of a third-party vSphere CSI driver prevents OpenShift Container Platform from updating to OpenShift Container Platform 4.13 or later.

NOTE

The VMware vSphere CSI Driver Operator is supported only on clusters deployed with **platform: vsphere** in the installation manifest.

To remove a third-party CSI driver, see Removing a third-party vSphere CSI Driver .

5.22.6. Removing a third-party vSphere CSI Driver Operator

OpenShift Container Platform 4.10, and later, includes a built-in version of the vSphere Container Storage Interface (CSI) Operator Driver that is supported by Red Hat. If you have installed a vSphere CSI driver provided by the community or another vendor, updates to the next major version of OpenShift Container Platform, such as 4.13, or later, might be disabled for your cluster.

OpenShift Container Platform 4.12, and later, clusters are still fully supported, and updates to z-stream releases of 4.12, such as 4.12.z, are not blocked, but you must correct this state by removing the third-party vSphere CSI Driver before updates to next major version of OpenShift Container Platform can occur. Removing the third-party vSphere CSI driver does not require deletion of associated persistent volume (PV) objects, and no data loss should occur.

NOTE

These instructions may not be complete, so consult the vendor or community provider uninstall guide to ensure removal of the driver and components.

To uninstall the third-party vSphere CSI Driver:

1. Delete the third-party vSphere CSI Driver (VMware vSphere Container Storage Plugin) Deployment and Daemonset objects.

2. Delete the configmap and secret objects that were installed previously with the third-party vSphere CSI Driver.

3. Delete the third-party vSphere CSI driver **CSIDriver** object:

```
~ $ oc delete CSIDriver csi.vsphere.vmware.com
```

```
csidriver.storage.k8s.io "csi.vsphere.vmware.com" deleted
```

After you have removed the third-party vSphere CSI Driver from the OpenShift Container Platform cluster, installation of Red Hat's vSphere CSI Driver Operator automatically resumes, and any conditions that could block upgrades to OpenShift Container Platform 4.11, or later, are automatically removed. If you had existing vSphere CSI PV objects, their lifecycle is now managed by Red Hat's vSphere CSI Driver Operator.

5.22.7. vSphere persistent disks encryption

You can encrypt virtual machines (VMs) and dynamically provisioned persistent volumes (PVs) on OpenShift Container Platform running on top of vSphere.

NOTE

OpenShift Container Platform does not support RWX-encrypted PVs. You cannot request RWX PVs out of a storage class that uses an encrypted storage policy.

You must encrypt VMs before you can encrypt PVs, which you can do during installation or postinstallation.

For information about encrypting VMs, see:

- Requirements for encrypting virtual machines
- During installation: Step 7 of Installing RHCOS and starting the OpenShift Container Platform bootstrap process
- Post-installation enabling encryption on a vSphere cluster

After encrypting VMs, you can configure a storage class that supports dynamic encryption volume provisioning using the vSphere Container Storage Interface (CSI) driver. This can be accomplished in one of two ways using:

- Datastore URL: This approach is not very flexible, and forces you to use a single datastore. It also does not support topology-aware provisioning.
- Tag-based placement: Encrypts the provisioned volumes and uses tag-based placement to target specific datastores.

5.22.7.1. Using datastore URL

Procedure

To encrypt using the datastore URL:

1. Find out the name of the default storage policy in your datastore that supports encryption. This is same policy that was used for encrypting your VMs.
2. Create a storage class that uses this storage policy:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: encryption
provisioner: csi.vsphere.vmware.com
parameters:
 storagePolicyName: <storage-policy-name> 1
 datastoreurl: "ds:///vmfs/volumes/vsan:522e875627d-b090c96b526bb79c/"
```

1 Name of default storage policy in your datastore that supports encryption

5.22.7.2. Using tag-based placement

Procedure

To encrypt using tag-based placement:

1. In vCenter create a category for tagging datastores that will be made available to this storage class. Also, ensure that **StoragePod(Datastore clusters)**, **Datastore**, and **Folder** are selected as Associable Entities for the created category.
2. In vCenter, create a tag that uses the category created earlier.
3. Assign the previously created tag to each datastore that will be made available to the storage class. Make sure that datastores are shared with hosts participating in the OpenShift Container Platform cluster.
4. In vCenter, from the main menu, click **Policies and Profiles**.
5. On the **Policies and Profiles** page, in the navigation pane, click **VM Storage Policies**.
6. Click **CREATE**.
7. Type a name for the storage policy.

8. Select **Enable host based rules** and **Enable tag based placement rules**

9. In the **Next** tab:

    a. Select **Encryption** and **Default Encryption Properties**.

    b. Select the tag category created earlier, and select tag selected. Verify that the policy is selecting matching datastores.

10. Create the storage policy.

11. Create a storage class that uses the storage policy:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name:  csi-encrypted
provisioner: csi.vsphere.vmware.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
parameters:
 storagePolicyName: <storage-policy-name> 1
```

1 Name of the storage policy that you created for encryption

5.22.8. vSphere CSI topology overview

OpenShift Container Platform provides the ability to deploy OpenShift Container Platform for vSphere on different zones and regions, which allows you to deploy over multiple compute clusters and datacenters, thus helping to avoid a single point of failure.

This is accomplished by defining zone and region categories in vCenter, and then assigning these categories to different failure domains, such as a compute cluster, by creating tags for these zone and region categories. After you have created the appropriate categories, and assigned tags to vCenter objects, you can create additional machinesets that create virtual machines (VMs) that are responsible for scheduling pods in those failure domains.

The following example defines two failure domains with one region and two zones:

Table 5.6. vSphere storage topology with one region and two zones

Compute cluster	Failure domain	Description
Compute cluster: ocp1, Datacenter: Atlanta	openshift-region: us-east-1 (tag), openshift-zone: us-east-1a (tag)	This defines a failure domain in region us-east-1 with zone us-east-1a.
Computer cluster: ocp2, Datacenter: Atlanta	openshift-region: us-east-1 (tag), openshift-zone: us-east-1b (tag)	This defines a different failure domain within the same region called us-east-1b.

5.22.8.1. Creating vSphere storage topology during installation

5.22.8.1.1. Procedure

- Specify the topology during installation. See the *Configuring regions and zones for a VMware vCenter* section.

No additional action is necessary and the default storage class that is created by OpenShift Container Platform is topology aware and should allow provisioning of volumes in different failure domains.

Additional resources

- Configuring regions and zones for a VMware vCenter

5.22.8.2. Creating vSphere storage topology postinstallation

5.22.8.2.1. Procedure

1. In the VMware vCenter vSphere client GUI, define appropriate zone and region catagories and tags.
   While vSphere allows you to create categories with any arbitrary name, OpenShift Container Platform strongly recommends use of **openshift-region** and **openshift-zone** names for defining topology categories.

   For more information about vSphere categories and tags, see the VMware vSphere documentation.

2. In OpenShift Container Platform, create failure domains. See the *Specifying multiple regions and zones for your cluster on vSphere* section.

3. Create a tag to assign to datastores across failure domains:
   When an OpenShift Container Platform spans more than one failure domain, the datastore might not be shared across those failure domains, which is where topology-aware provisioning of persistent volumes (PVs) is useful.

   a. In vCenter, create a category for tagging the datastores. For example, **openshift-zonal-datastore-cat**. You can use any other category name, provided the category uniquely is used for tagging datastores participating in OpenShift Container Platform cluster. Also, ensure that **StoragePod**, **Datastore**, and **Folder** are selected as Associable Entities for the created category.

   b. In vCenter, create a tag that uses the previously created category. This example uses the tag name **openshift-zonal-datastore**.

   c. Assign the previously created tag (in this example **openshift-zonal-datastore**) to each datastore in a failure domain that would be considered for dynamic provisioning.

   NOTE

   You can use any names you like for datastore categories and tags. The names used in this example are provided as recommendations. Ensure that the tags and categories that you define uniquely identify only datastores that are shared with all hosts in the OpenShift Container Platform cluster.

4. As needed, create a storage policy that targets the tag-based datastores in each failure domain:

   a. In vCenter, from the main menu, click **Policies and Profiles**.

b. On the **Policies and Profiles** page, in the navigation pane, click **VM Storage Policies**.

c. Click **CREATE**.

d. Type a name for the storage policy.

e. For the rules, choose Tag Placement rules and select the tag and category that targets the desired datastores (in this example, the **openshift-zonal-datastore** tag).
The datastores are listed in the storage compatibility table.

5. Create a new storage class that uses the new zoned storage policy:

a. Click **Storage** > **StorageClasses**.

b. On the **StorageClasses** page, click **Create StorageClass**.

c. Type a name for the new storage class in **Name**.

d. Under **Provisioner**, select **csi.vsphere.vmware.com**.

e. Under **Additional parameters**, for the StoragePolicyName parameter, set **Value** to the name of the new zoned storage policy that you created earlier.

f. Click **Create**.

Example output

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: zoned-sc 1
provisioner: csi.vsphere.vmware.com
parameters:
  StoragePolicyName: zoned-storage-policy 2
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

1 New topology aware storage class name.

2 Specify zoned storage policy.

NOTE

You can also create the storage class by editing the preceding YAML file and running the command **oc create -f $FILE**.

Additional resources

- Specifying multiple regions and zones for your cluster on vSphere
- VMware vSphere tag documentation

5.22.8.3. Creating vSphere storage topology without an infra topology

NOTE

OpenShift Container Platform recommends using the infrastructure object for specifying failure domains in a topology aware setup. Specifying failure domains in the infrastructure object and specify topology-categories in the **ClusterCSIDriver** object at the same time is an unsupported operation.

5.22.8.3.1. Procedure

1. In the VMware vCenter vSphere client GUI, define appropriate zone and region catagories and tags.
   While vSphere allows you to create categories with any arbitrary name, OpenShift Container Platform strongly recommends use of **openshift-region** and **openshift-zone** names for defining topology.

   For more information about vSphere categories and tags, see the VMware vSphere documentation.

2. To allow the container storage interface (CSI) driver to detect this topology, edit the **clusterCSIDriver** object YAML file **driverConfig** section:

   - Specify the **openshift-zone** and **openshift-region** categories that you created earlier.

   - Set **driverType** to **vSphere**.

     ```
     ~ $ oc edit clustercsidriver csi.vsphere.vmware.com -o yaml
     ```

     Example output

     ```
     apiVersion: operator.openshift.io/v1
     kind: ClusterCSIDriver
     metadata:
       name: csi.vsphere.vmware.com
     spec:
       logLevel: Normal
       managementState: Managed
       observedConfig: null
       operatorLogLevel: Normal
       unsupportedConfigOverrides: null
       driverConfig:
         driverType: vSphere ①
           vSphere:
             topologyCategories: ②
             - openshift-zone
             - openshift-region
     ```

     ① Ensure that **driverType** is set to **vSphere**.

     ② **openshift-zone** and **openshift-region** categories created earlier in vCenter.

3. Verify that **CSINode** object has topology keys by running the following commands:

   ```
   ~ $ oc get csinode
   ```

Example output

```
NAME DRIVERS AGE
co8-4s88d-infra-2m5vd 1 27m
co8-4s88d-master-0 1 70m
co8-4s88d-master-1 1 70m
co8-4s88d-master-2 1 70m
co8-4s88d-worker-j2hmg 1 47m
co8-4s88d-worker-mbb46 1 47m
co8-4s88d-worker-zlk7d 1 47m
```

```
~ $ oc get csinode co8-4s88d-worker-j2hmg -o yaml
```

Example output

```
...
spec:
  drivers:
  - allocatable:
      count: 59
  name: csi-vsphere.vmware.com
  nodeID: co8-4s88d-worker-j2hmg
  topologyKeys: 1
  - topology.csi.vmware.com/openshift-zone
  - topology.csi.vmware.com/openshift-region
```

1 Topology keys from vSphere **openshift-zone** and **openshift-region** catagories.

NOTE

CSINode objects might take some time to receive updated topology information. After the driver is updated, **CSINode** objects should have topology keys in them.

4. Create a tag to assign to datastores across failure domains:
   When an OpenShift Container Platform spans more than one failure domain, the datastore might not be shared across those failure domains, which is where topology-aware provisioning of persistent volumes (PVs) is useful.

   a. In vCenter, create a category for tagging the datastores. For example, **openshift-zonal-datastore-cat**. You can use any other category name, provided the category uniquely is used for tagging datastores participating in OpenShift Container Platform cluster. Also, ensure that **StoragePod**, **Datastore**, and **Folder** are selected as Associable Entities for the created category.

   b. In vCenter, create a tag that uses the previously created category. This example uses the tag name **openshift-zonal-datastore**.

   c. Assign the previously created tag (in this example **openshift-zonal-datastore**) to each datastore in a failure domain that would be considered for dynamic provisioning.

NOTE

You can use any names you like for categories and tags. The names used in this example are provided as recommendations. Ensure that the tags and categories that you define uniquely identify only datastores that are shared with all hosts in the OpenShift Container Platform cluster.

5. Create a storage policy that targets the tag-based datastores in each failure domain:

   a. In vCenter, from the main menu, click **Policies and Profiles**.

   b. On the **Policies and Profiles** page, in the navigation pane, click **VM Storage Policies**.

   c. Click **CREATE**.

   d. Type a name for the storage policy.

   e. For the rules, choose Tag Placement rules and select the tag and category that targets the desired datastores (in this example, the **openshift-zonal-datastore** tag).
   The datastores are listed in the storage compatibility table.

6. Create a new storage class that uses the new zoned storage policy:

   a. Click **Storage** > **StorageClasses**.

   b. On the **StorageClasses** page, click **Create StorageClass**.

   c. Type a name for the new storage class in **Name**.

   d. Under **Provisioner**, select **csi.vsphere.vmware.com**.

   e. Under **Additional parameters**, for the StoragePolicyName parameter, set **Value** to the name of the new zoned storage policy that you created earlier.

   f. Click **Create**.

   Example output

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: zoned-sc 1
provisioner: csi.vsphere.vmware.com
parameters:
  StoragePolicyName: zoned-storage-policy 2
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

1 New topology aware storage class name.

2 Specify zoned storage policy.

NOTE

You can also create the storage class by editing the preceding YAML file and running the command **oc create -f $FILE**.

Additional resources

- VMware vSphere tag documentation

5.22.8.4. Results

Creating persistent volume claims (PVCs) and PVs from the topology aware storage class are truly zonal, and should use the datastore in their respective zone depending on how pods are scheduled:

```
~ $ oc get pv <pv-name> -o yaml
```

Example output

```
...
nodeAffinity:
  required:
    nodeSelectorTerms:
    - matchExpressions:
      - key: topology.csi.vmware.com/openshift-zone 1
        operator: In
        values:
        - <openshift-zone>
      -key: topology.csi.vmware.com/openshift-region 2
        operator: In
        values:
        - <openshift-region>
...
peristentVolumeclaimPolicy: Delete
storageClassName: <zoned-storage-class-name> 3
volumeMode: Filesystem
...
```

1 2 PV has zoned keys.

3 PV is using the zoned storage class.

5.22.9. Additional resources

- Configuring CSI volumes

CHAPTER 6. GENERIC EPHEMERAL VOLUMES

6.1. OVERVIEW

Generic ephemeral volumes are a type of ephemeral volume that can be provided by all storage drivers that support persistent volumes and dynamic provisioning. Generic ephemeral volumes are similar to **emptyDir** volumes in that they provide a per-pod directory for scratch data, which is usually empty after provisioning.

Generic ephemeral volumes are specified inline in the pod spec and follow the pod's lifecycle. They are created and deleted along with the pod.

Generic ephemeral volumes have the following features:

- Storage can be local or network-attached.
- Volumes can have a fixed size that pods are not able to exceed.
- Volumes might have some initial data, depending on the driver and parameters.
- Typical operations on volumes are supported, assuming that the driver supports them, including snapshotting, cloning, resizing, and storage capacity tracking.

NOTE

Generic ephemeral volumes do not support offline snapshots and resize.

Due to this limitation, the following Container Storage Interface (CSI) drivers do not support the following features for generic ephemeral volumes:

- Azure Disk CSI driver does not support resize.
- Cinder CSI driver does not support snapshot.

6.2. LIFECYCLE AND PERSISTENT VOLUME CLAIMS

The parameters for a volume claim are allowed inside a volume source of a pod. Labels, annotations, and the whole set of fields for persistent volume claims (PVCs) are supported. When such a pod is created, the ephemeral volume controller then creates an actual PVC object (from the template shown in the *Creating generic ephemeral volumes* procedure) in the same namespace as the pod, and ensures that the PVC is deleted when the pod is deleted. This triggers volume binding and provisioning in one of two ways:

- Either immediately, if the storage class uses immediate volume binding.
  With immediate binding, the scheduler is forced to select a node that has access to the volume after it is available.
- When the pod is tentatively scheduled onto a node (**WaitForFirstConsumervolume** binding mode).
  This volume binding option is recommended for generic ephemeral volumes because then the scheduler can choose a suitable node for the pod.

In terms of resource ownership, a pod that has generic ephemeral storage is the owner of the PVCs that provide that ephemeral storage. When the pod is deleted, the Kubernetes garbage collector deletes the PVC, which then usually triggers deletion of the volume because the default reclaim policy of storage classes is to delete volumes. You can create quasi-ephemeral local storage by using a storage class with a reclaim policy of retain: the storage outlives the pod, and in this case, you must ensure that volume clean-up happens separately. While these PVCs exist, they can be used like any other PVC. In particular, they can be referenced as data source in volume cloning or snapshotting. The PVC object also holds the current status of the volume.

Additional resources

- Creating generic ephemeral volumes

6.3. SECURITY

You can enable the generic ephemeral volume feature to allows users who can create pods to also create persistent volume claims (PVCs) indirectly. This feature works even if these users do not have permission to create PVCs directly. Cluster administrators must be aware of this. If this does not fit your security model, use an admission webhook that rejects objects such as pods that have a generic ephemeral volume.

The normal namespace quota for PVCs still applies, so even if users are allowed to use this new mechanism, they cannot use it to circumvent other policies.

6.4. PERSISTENT VOLUME CLAIM NAMING

Automatically created persistent volume claims (PVCs) are named by a combination of the pod name and the volume name, with a hyphen (-) in the middle. This naming convention also introduces a potential conflict between different pods, and between pods and manually created PVCs.

For example, **pod-a** with volume **scratch** and **pod** with volume **a-scratch** both end up with the same PVC name, **pod-a-scratch**.

Such conflicts are detected, and a PVC is only used for an ephemeral volume if it was created for the pod. This check is based on the ownership relationship. An existing PVC is not overwritten or modified, but this does not resolve the conflict. Without the right PVC, a pod cannot start.

IMPORTANT

Be careful when naming pods and volumes inside the same namespace so that naming conflicts do not occur.

6.5. CREATING GENERIC EPHEMERAL VOLUMES

Procedure

1. Create the **pod** object definition and save it to a file.

2. Include the generic ephemeral volume information in the file.

   my-example-pod-with-generic-vols.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-app
```

```
spec:
  containers:
    - name: my-frontend
      image: busybox:1.28
      volumeMounts:
      - mountPath: "/mnt/storage"
        name: data
      command: [ "sleep", "1000000" ]
  volumes:
    - name: data 1
      ephemeral:
        volumeClaimTemplate:
          metadata:
            labels:
              type: my-app-ephvol
          spec:
            accessModes: [ "ReadWriteOnce" ]
            storageClassName: "gp2-csi"
            resources:
              requests:
                storage: 1Gi
```

1 Generic ephemeral volume claim.

CHAPTER 7. EXPANDING PERSISTENT VOLUMES

7.1. ENABLING VOLUME EXPANSION SUPPORT

Before you can expand persistent volumes, the **StorageClass** object must have the **allowVolumeExpansion** field set to **true**.

Procedure

- Edit the **StorageClass** object and add the **allowVolumeExpansion** attribute by running the following command:

  ```
  $ oc edit storageclass <storage_class_name> 1
  ```

  1 Specifies the name of the storage class.

  The following example demonstrates adding this line at the bottom of the storage class configuration.

  ```
  apiVersion: storage.k8s.io/v1
  kind: StorageClass
  ...
  parameters:
    type: gp2
  reclaimPolicy: Delete
  allowVolumeExpansion: true 1
  ```

  1 Setting this attribute to **true** allows PVCs to be expanded after creation.

7.2. EXPANDING CSI VOLUMES

You can use the Container Storage Interface (CSI) to expand storage volumes after they have already been created.

CSI volume expansion does not support the following:

- Recovering from failure when expanding volumes
- Shrinking

Prerequisites

- The underlying CSI driver supports resize.
- Dynamic provisioning is used.
- The controlling **StorageClass** object has **allowVolumeExpansion** set to **true**. For more information, see "Enabling volume expansion support."

Procedure

1. For the persistent volume claim (PVC), set **.spec.resources.requests.storage** to the desired new size.

2. Watch the **status.conditions** field of the PVC to see if the resize has completed. OpenShift Container Platform adds the **Resizing** condition to the PVC during expansion, which is removed after expansion completes.

7.3. EXPANDING FLEXVOLUME WITH A SUPPORTED DRIVER

When using FlexVolume to connect to your back-end storage system, you can expand persistent storage volumes after they have already been created. This is done by manually updating the persistent volume claim (PVC) in OpenShift Container Platform.

FlexVolume allows expansion if the driver is set with **RequiresFSResize** to **true**. The FlexVolume can be expanded on pod restart.

Similar to other volume types, FlexVolume volumes can also be expanded when in use by a pod.

Prerequisites

- The underlying volume driver supports resize.

- The driver is set with the **RequiresFSResize** capability to **true**.

- Dynamic provisioning is used.

- The controlling **StorageClass** object has **allowVolumeExpansion** set to **true**.

Procedure

- To use resizing in the FlexVolume plugin, you must implement the **ExpandableVolumePlugin** interface using these methods:

  RequiresFSResize

  If **true**, updates the capacity directly. If **false**, calls the **ExpandFS** method to finish the filesystem resize.

  ExpandFS

  If **true**, calls **ExpandFS** to resize filesystem after physical volume expansion is done. The volume driver can also perform physical volume resize together with filesystem resize.

> IMPORTANT
>
> Because OpenShift Container Platform does not support installation of FlexVolume plugins on control plane nodes, it does not support control-plane expansion of FlexVolume.

7.4. EXPANDING LOCAL VOLUMES

You can manually expand persistent volumes (PVs) and persistent volume claims (PVCs) created by using the local storage operator (LSO).

Procedure

1. Expand the underlying devices. Ensure that appropriate capacity is available on these devices.

2. Update the corresponding PV objects to match the new device sizes by editing the **.spec.capacity** field of the PV.

3. For the storage class that is used for binding the PVC to PVet, set **allowVolumeExpansion:true**.

4. For the PVC, set **.spec.resources.requests.storage** to match the new size.

Kubelet should automatically expand the underlying file system on the volume, if necessary, and update the status field of the PVC to reflect the new size.

7.5. EXPANDING PERSISTENT VOLUME CLAIMS (PVCS) WITH A FILE SYSTEM

Expanding PVCs based on volume types that need file system resizing, such as GCE, EBS, and Cinder, is a two-step process. First, expand the volume objects in the cloud provider. Second, expand the file system on the node.

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

Prerequisites

- The controlling **StorageClass** object must have **allowVolumeExpansion** set to **true**.

Procedure

1. Edit the PVC and request a new size by editing **spec.resources.requests**. For example, the following expands the **ebs** PVC to 8 Gi:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ebs
spec:
  storageClass: "storageClassWithFlagSet"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi 1
```

1 Updating **spec.resources.requests** to a larger amount expands the PVC.

2. After the cloud provider object has finished resizing, the PVC is set to **FileSystemResizePending**. Check the condition by entering the following command:

```
$ oc describe pvc <pvc_name>
```

3. When the cloud provider object has finished resizing, the **PersistentVolume** object reflects the newly requested size in **PersistentVolume.Spec.Capacity**. At this point, you can create or recreate a new pod from the PVC to finish the file system resizing. Once the pod is running, the newly requested size is available and the **FileSystemResizePending** condition is removed from the PVC.

7.6. RECOVERING FROM FAILURE WHEN EXPANDING VOLUMES

If expanding underlying storage fails, the OpenShift Container Platform administrator can manually recover the persistent volume claim (PVC) state and cancel the resize requests. Otherwise, the resize requests are continuously retried by the controller.

Procedure

1. Mark the persistent volume (PV) that is bound to the PVC with the **Retain** reclaim policy. This can be done by editing the PV and changing **persistentVolumeReclaimPolicy** to **Retain**.

2. Delete the PVC.

3. Manually edit the PV and delete the **claimRef** entry from the PV specs to ensure that the newly created PVC can bind to the PV marked **Retain**. This marks the PV as **Available**.

4. Re-create the PVC in a smaller size, or a size that can be allocated by the underlying storage provider.

5. Set the **volumeName** field of the PVC to the name of the PV. This binds the PVC to the provisioned PV only.

6. Restore the reclaim policy on the PV.

Additional resources

- The controlling **StorageClass** object has **allowVolumeExpansion** set to **true** (see Enabling volume expansion support).

CHAPTER 8. DYNAMIC PROVISIONING

8.1. ABOUT DYNAMIC PROVISIONING

The **StorageClass** resource object describes and classifies storage that can be requested, as well as provides a means for passing parameters for dynamically provisioned storage on demand. **StorageClass** objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. Cluster Administrators (**cluster-admin**) or Storage Administrators (**storage-admin**) define and create the **StorageClass** objects that users can request without needing any detailed knowledge about the underlying storage volume sources.

The OpenShift Container Platform persistent volume framework enables this functionality and allows administrators to provision a cluster with persistent storage. The framework also gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Many storage types are available for use as persistent volumes in OpenShift Container Platform. While all of them can be statically provisioned by an administrator, some types of storage are created dynamically using the built-in provider and plugin APIs.

8.2. AVAILABLE DYNAMIC PROVISIONING PLUGINS

OpenShift Container Platform provides the following provisioner plugins, which have generic implementations for dynamic provisioning that use the cluster’s configured provider’s API to create new storage resources:

Storage type	Provisioner plugin name	Notes
Red Hat OpenStack Platform (RHOSP) Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface (CSI)	**manila.csi.openstack.org**	Once installed, the OpenStack Manila CSI Driver Operator and ManilaDriver automatically create the required storage classes for all available Manila share types needed for dynamic provisioning.
Amazon Elastic Block Store (Amazon EBS)	**kubernetes.io/aws-ebs**	For dynamic provisioning when using multiple clusters in different zones, tag each node with **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>** where **<cluster_name>** and **<cluster_id>** are unique per cluster.
Azure Disk	**kubernetes.io/azure-disk**	

Storage type	Provisioner plugin name	Notes
Azure File	**kubernetes.io/azure-file**	The **persistent-volume-binder** service account requires permissions to create and get secrets to store the Azure storage account and keys.
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	In multi-zone configurations, it is advisable to run one OpenShift Container Platform cluster per GCE project to avoid PVs from being created in zones where no node in the current cluster exists.
IBM Power® Virtual Server Block	**powervs.csi.ibm.com**	After installation, the IBM Power® Virtual Server Block CSI Driver Operator and IBM Power® Virtual Server Block CSI Driver automatically create the required storage classes for dynamic provisioning.
VMware vSphere	**kubernetes.io/vsphere-volume**	

IMPORTANT

Any chosen provisioner plugin also requires configuration for the relevant cloud, host, or third-party provider as per the relevant documentation.

8.3. DEFINING A STORAGE CLASS

StorageClass objects are currently a globally scoped object and must be created by **cluster-admin** or **storage-admin** users.

IMPORTANT

The Cluster Storage Operator might install a default storage class depending on the platform in use. This storage class 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 storage class.

The following sections describe the basic definition for a **StorageClass** object and specific examples for each of the supported plugin types.

8.3.1. Basic StorageClass object definition

The following resource shows the parameters and default values that you use to configure a storage class. This example uses the AWS ElasticBlockStore (EBS) object definition.

Sample StorageClass definition

```
kind: StorageClass ❶
apiVersion: storage.k8s.io/v1 ❷
metadata:
  name: <storage-class-name> ❸
  annotations: ❹
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs ❺
parameters: ❻
  type: gp3
...
```

❶ (required) The API object type.

❷ (required) The current apiVersion.

❸ (required) The name of the storage class.

❹ (optional) Annotations for the storage class.

❺ (required) The type of provisioner associated with this storage class.

❻ (optional) The parameters required for the specific provisioner, this will change from plugin to plug-iin.

8.3.2. Storage class annotations

To set a storage class as the cluster-wide default, add the following annotation to your storage class metadata:

```
storageclass.kubernetes.io/is-default-class: "true"
```

For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

This enables any persistent volume claim (PVC) that does not specify a specific storage class to automatically be provisioned through the default storage class. However, your cluster can have more than one storage class, but only one of them can be the default storage class.

NOTE

The beta annotation **storageclass.beta.kubernetes.io/is-default-class** is still working; however, it will be removed in a future release.

To set a storage class description, add the following annotation to your storage class metadata:

```
kubernetes.io/description: My Storage Class Description
```

For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

8.3.3. RHOSP Cinder object definition

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> ❶
provisioner: kubernetes.io/cinder
parameters:
  type: fast  ❷
  availability: nova ❸
  fsType: ext4 ❹
```

❶ Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

❷ Volume type created in Cinder. Default is empty.

❸ Availability Zone. If not specified, volumes are generally round-robined across all active zones where the OpenShift Container Platform cluster has a node.

❹ File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

8.3.4. RHOSP Manila Container Storage Interface (CSI) object definition

Once installed, the OpenStack Manila CSI Driver Operator and ManilaDriver automatically create the required storage classes for all available Manila share types needed for dynamic provisioning.

8.3.5. AWS Elastic Block Store (EBS) object definition

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 2
  iopsPerGB: "10" 3
  encrypted: "true" 4
  kmsKeyId: keyvalue 5
  fsType: ext4 6
```

1. (required) Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2. (required) Select from **io1**, **gp3**, **sc1**, **st1**. The default is **gp3**. See the AWS documentation for valid Amazon Resource Name (ARN) values.

3. Optional: Only for **io1** volumes. I/O operations per second per GiB. The AWS volume plugin multiplies this with the size of the requested volume to compute IOPS of the volume. The value cap is 20,000 IOPS, which is the maximum supported by AWS. See the AWS documentation for further details.

4. Optional: Denotes whether to encrypt the EBS volume. Valid values are **true** or **false**.

5. Optional: The full ARN of the key to use when encrypting the volume. If none is supplied, but **encypted** is set to **true**, then AWS generates a key. See the AWS documentation for a valid ARN value.

6. Optional: File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

8.3.6. Azure Disk object definition

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer 2
allowVolumeExpansion: true
parameters:
  kind: Managed 3
  storageaccounttype: Premium_LRS 4
reclaimPolicy: Delete
```

1. Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2. Using **WaitForFirstConsumer** is strongly recommended. This provisions the volume while allowing enough storage to schedule the pod on a free worker node from an available zone.

3. Possible values are **Shared** (default), **Managed**, and **Dedicated**.

IMPORTANT

Red Hat only supports the use of **kind: Managed** in the storage class.

With **Shared** and **Dedicated**, Azure creates unmanaged disks, while OpenShift Container Platform creates a managed disk for machine OS (root) disks. But because Azure Disk does not allow the use of both managed and unmanaged disks on a node, unmanaged disks created with **Shared** or **Dedicated** cannot be attached to OpenShift Container Platform nodes.

4 Azure storage account SKU tier. Default is empty. Note that Premium VMs can attach both **Standard_LRS** and **Premium_LRS** disks, Standard VMs can only attach **Standard_LRS** disks, Managed VMs can only attach managed disks, and unmanaged VMs can only attach unmanaged disks.

a. If **kind** is set to **Shared**, Azure creates all unmanaged disks in a few shared storage accounts in the same resource group as the cluster.

b. If **kind** is set to **Managed**, Azure creates new managed disks.

c. If **kind** is set to **Dedicated** and a **storageAccount** is specified, Azure uses the specified storage account for the new unmanaged disk in the same resource group as the cluster. For this to work:

- The specified storage account must be in the same region.
- Azure Cloud Provider must have write access to the storage account.

d. If **kind** is set to **Dedicated** and a **storageAccount** is not specified, Azure creates a new dedicated storage account for the new unmanaged disk in the same resource group as the cluster.

8.3.7. Azure File object definition

The Azure File storage class uses secrets to store the Azure storage account name and the storage account key that are required to create an Azure Files share. These permissions are created as part of the following procedure.

Procedure

1. Define a **ClusterRole** object that allows access to create and view secrets:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
#  name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: ['']
  resources: ['secrets']
  verbs:     ['get','create']
```

1 The name of the cluster role to view and create secrets.

2. Add the cluster role to the service account:

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
system:serviceaccount:kube-system:persistent-volume-binder
```

3. Create the Azure File **StorageClass** object:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> 1
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus 2
  skuName: Standard_LRS 3
  storageAccount: <storage-account> 4
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 Location of the Azure storage account, such as **eastus**. Default is empty, meaning that a new Azure storage account will be created in the OpenShift Container Platform cluster's location.

3 SKU tier of the Azure storage account, such as **Standard_LRS**. Default is empty, meaning that a new Azure storage account will be created with the **Standard_LRS** SKU.

4 Name of the Azure storage account. If a storage account is provided, then **skuName** and **location** are ignored. If no storage account is provided, then the storage class searches for any storage account that is associated with the resource group for any accounts that match the defined **skuName** and **location**.

8.3.7.1. Considerations when using Azure File

The following file system features are not supported by the default Azure File storage class:

- Symlinks
- Hard links
- Extended attributes
- Sparse files
- Named pipes

Additionally, the owner user identifier (UID) of the Azure File mounted directory is different from the process UID of the container. The **uid** mount option can be specified in the **StorageClass** object to define a specific user identifier to use for the mounted directory.

The following **StorageClass** object demonstrates modifying the user and group identifier, along with enabling symlinks for the mounted directory.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 Specifies the user identifier to use for the mounted directory.

2 Specifies the group identifier to use for the mounted directory.

3 Enables symlinks.

8.3.8. GCE PersistentDisk (gcePD) object definition

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 2
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 Select either **pd-standard** or **pd-ssd**. The default is **pd-standard**.

8.3.9. VMware vSphere object definition

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: csi.vsphere.vmware.com 2
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 For more information about using VMware vSphere CSI with OpenShift Container Platform, see the Kubernetes documentation.

8.4. CHANGING THE DEFAULT STORAGE CLASS

Use the following procedure to change the default storage class.

For example, if you have two defined storage classes, **gp3** and **standard**, and you want to change the default storage class from **gp3** to **standard**.

Prerequisites

- Access to the cluster with cluster-admin privileges.

Procedure

To change the default storage class:

1. List the storage classes:

```
$ oc get storageclass
```

Example output

```
NAME            TYPE
gp3 (default)    kubernetes.io/aws-ebs 1
standard        kubernetes.io/aws-ebs
```

1 **(default)** indicates the default storage class.

2. Make the desired storage class the default.
For the desired storage class, set the **storageclass.kubernetes.io/is-default-class** annotation to **true** by running the following command:

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

NOTE

You can have multiple default storage classes for a short time. However, you should ensure that only one default storage class exists eventually.

With multiple default storage classes present, any persistent volume claim (PVC) requesting the default storage class (**pvc.spec.storageClassName**=nil) gets the most recently created default storage class, regardless of the default status of that storage class, and the administrator receives an alert in the alerts dashboard that there are multiple default storage classes, **MultipleDefaultStorageClasses**.

3. Remove the default storage class setting from the old default storage class.

For the old default storage class, change the value of the **storageclass.kubernetes.io/is-default-class** annotation to **false** by running the following command:

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

4. Verify the changes:

```
$ oc get storageclass
```

Example output

```
NAME                 TYPE
gp3                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```