OpenShift Container Platform 4.15

存储

在 OpenShift Container Platform 中配置和管理存储

Last Updated: 2024-07-26

OpenShift Container Platform 4.15 存储

在 OpenShift Container Platform 中配置和管理存储

法律通告

摘要

本文档提供了使用不同存储后端配置持久性卷以及通过 pod 管理动态分配存储的信息。

目录

第 1 章 OPENSHIFT CONTAINER PLATFORM 存储概述

OpenShift Container Platform 支持多种存储类型，包括内部存储和云供应商。您可以在 OpenShift Container Platform 集群中管理持久性和非持久性数据的容器存储。

1.1. OPENSHIFT CONTAINER PLATFORM 存储的常见术语表

该术语表定义了存储内容中使用的常用术语。

访问模式

卷访问模式描述了卷功能。您可以使用访问模式匹配持久性卷声明 (PVC) 和持久性卷 (PV)。以下是访问模式的示例：

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

Cinder

Red Hat OpenStack Platform (RHOSP) 的块存储服务，用于管理所有卷的管理、安全性和调度。

配置映射

配置映射提供将配置数据注入 pod 的方法。您可以在类型为 **ConfigMap** 的卷中引用存储在配置映射中的数据。在 pod 中运行的应用程序可以使用这个数据。

Container Storage Interface (CSI)

在不同容器编配 (CO) 系统之间管理容器存储的 API 规格。

动态置备

该框架允许您按需创建存储卷，使集群管理员无需预置备持久性存储。

临时存储

Pod 和容器可能需要临时或过渡的本地存储才能进行操作。此临时存储的生命周期不会超过每个 pod 的生命周期，且此临时存储无法在 pod 间共享。

Fiber 频道

用于在数据中心、计算机服务器、交换机和存储之间传输数据的联网技术。

FlexVolume

FlexVolume 是一个树外插件接口，它使用基于 exec 的模型与存储驱动程序进行接口。您必须在每个节点上在预定义的卷插件路径中安装 FlexVolume 驱动程序二进制文件，并在某些情况下是 control plane 节点。

fsGroup

fsGroup 定义 pod 的文件系统组 ID。

iSCSI

互联网小型计算机系统接口 (iSCSI) 是基于互联网协议的存储网络标准，用于连接数据存储设施。iSCSI 卷允许将现有 iSCSI （通过 IP 的 SCSI）卷挂载到您的 Pod 中。

hostPath

OpenShift Container Platform 集群中的 hostPath 卷将主机节点的文件系统中的文件或目录挂载到 pod 中。

KMS 密钥

Key Management Service (KMS) 可帮助您在不同服务间实现所需的数据加密级别。您可以使用 KMS 密钥加密、解密和重新加密数据。

本地卷

本地卷代表挂载的本地存储设备，如磁盘、分区或目录。

NFS

网络文件系统(NFS)允许远程主机通过网络挂载文件系统，并像它们挂载在本地那样与这些文件系统进行交互。这使系统管理员能够将资源整合到网络上的集中式服务器上。

OpenShift Data Foundation

OpenShift Container Platform 支持的文件、块存储和对象存储的、内部或混合云的持久性存储供应商

持久性存储

Pod 和容器可能需要永久存储才能正常工作。OpenShift Container Platform 使用 Kubernetes 持久性卷 (PV) 框架来允许集群管理员为集群提供持久性存储。开发人员可以在不了解底层存储基础架构的情况下使用 PVC 来请求 PV 资源。

持久性卷 (PV)

OpenShift Container Platform 使用 Kubernetes 持久性卷 (PV) 框架来允许集群管理员为集群提供持久性存储。开发人员可以在不了解底层存储基础架构的情况下使用 PVC 来请求 PV 资源。

持久性卷声明 (PVC)

您可以使用 PVC 将 PersistentVolume 挂载到 Pod 中。您可以在不了解云环境的详情的情况下访问存储。

Pod

一个或多个带有共享资源（如卷和 IP 地址）的容器，在 OpenShift Container Platform 集群中运行。pod 是定义、部署和管理的最小计算单元。

重新声明策略

告知集群在卷被释放后使用什么操作。卷重新声明政策包括 **Retain**、**Recycle** 或 **Delete**。

基于角色的访问控制 (RBAC)

基于角色的访问控制 (RBAC) 是一种根据您机构中个别用户的角色监管计算机或网络资源的访问方法。

无状态应用程序

无状态应用是一种应用程序，它不会为与客户端进行下一个会话而保持在当前会话中生成的客户端数据。

有状态应用程序

有状态应用是一种应用程序，它会将数据保存到持久磁盘存储中。服务器、客户端和应用程序可以使用持久磁盘存储。您可以使用 OpenShift Container Platform 中的 **Statefulset** 对象来管理一组 Pod 的部署和扩展，并保证这些 Pod 的排序和唯一性。

静态置备

集群管理员创建多个 PV。PV 包含存储详情。PV 存在于 Kubernetes API 中，可供使用。

存储

OpenShift Container Platform 支持许多类型的存储，包括内部存储和云供应商。您可以在 OpenShift Container Platform 集群中管理持久性和非持久性数据的容器存储。

Storage class

存储类为管理员提供了描述它们所提供的存储类的方法。不同的类可能会映射到服务质量级别、备份策略，以及由集群管理员决定的任意策略。

VMware vSphere 的虚拟机磁盘 (VMDK) 卷

虚拟机磁盘 (VMDK) 是一种文件格式，用于描述虚拟机中使用的虚拟硬盘的容器。

1.2. 存储类型

OpenShift Container Platform 存储广泛分为两类，即临时存储和持久性存储。

1.2.1. 临时存储

Pod 和容器具有临时或临时性，面向无状态应用。临时存储可让管理员和开发人员更好地管理其某些操作的本地存储。如需有关临时存储概述、类型和管理的更多信息，请参阅了解临时存储。

1.2.2. 持久性存储

容器中部署的有状态应用需要持久存储。OpenShift Container Platform 使用名为持久性卷(PV)的预置备存储框架来允许集群管理员置备持久性存储。这些卷中的数据可能超过单个 pod 的生命周期。开发人员可以使用持久性卷声明(PVC)来请求存储要求。如需有关持久性存储概述、配置和生命周期的更多信息，请参阅了解持久性存储。

1.3. CONTAINER STORAGE INTERFACE (CSI)

CSI 是在不同容器编配(CO)系统中管理容器存储的 API 规范。您可以在容器原生环境中管理存储卷，而无需具体了解底层存储基础架构。使用 CSI 时，存储可在不同的容器编配系统中有效，无论您使用的存储供应商是什么。如需有关 CSI 的更多信息，请参阅使用容器存储接口(CSI)。

1.4. 动态置备

通过动态置备，您可以按需创建存储卷，使集群管理员无需预置备存储。有关动态置备的更多信息，请参阅动态置备。

第 2 章 了解临时存储

2.1. 概述

除了持久性存储外，Pod 和容器还需要临时或短暂的本地存储才能进行操作。此临时存储的生命周期不会超过每个 pod 的生命周期，且此临时存储无法在 pod 间共享。

Pod 使用临时本地存储进行涂销空间、缓存和日志。与缺少本地存储相关的问题包括：

- Pod 无法检测到有多少可用的本地存储。
- Pod 无法请求保证的本地存储。
- 本地存储是一个最佳资源。
- pod 可能会因为其他 pod 填充本地存储而被驱除。只有在足够的存储被重新声明前，不会接受这些新 pod。

与持久性卷不同，临时存储没有特定结构，它会被节点上运行的所有 pod 共享，并同时会被系统、容器运行时和 OpenShift Container Platform 使用。临时存储框架允许 Pod 指定其临时本地存储需求。它还允许 OpenShift Container Platform 在适当的时候调度 pod，并保护节点不受过度使用本地存储的影响。

虽然临时存储框架允许管理员和开发人员更好地管理本地存储，但 I/O 吞吐量和延迟不会直接生效。

2.2. 临时存储的类型

主分区中始终提供临时本地存储。创建主分区的基本方法有两种： root 和 runtime。

root
默认情况下，该分区包含 kubelet 根目录、**/var/lib/kubelet/** 和 **/var/log/** 目录。此分区可以在用户 Pod、OS 和 Kubernetes 系统守护进程间共享。Pod 可以通过 **EmptyDir** 卷、容器日志、镜像层和容器可写层来消耗这个分区。kubelet 管理这个分区的共享访问和隔离。这个分区是临时的，应用程序无法预期这个分区中的任何性能 SLA（如磁盘 IOPS）。

Runtime
这是一个可选分区，可用于 overlay 文件系统。OpenShift Container Platform 会尝试识别并提供共享访问以及这个分区的隔离。容器镜像层和可写入层存储在此处。如果 runtime 分区存在，则 **root** 分区不包含任何镜像层或者其它可写入的存储。

2.3. 临时存储管理

集群管理员可以通过设置配额在非终端状态的所有 Pod 中定义临时存储的限制范围，及临时存储请求数量，来管理项目中的临时存储。开发人员也可以在 Pod 和容器级别设置这个计算资源的请求和限值。

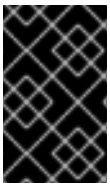
您可以通过指定请求和限值来管理本地临时存储。pod 中的每个容器可以指定以下内容：

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

2.3.1. 临时存储限制和请求单元

临时存储的限制和请求以字节数量来衡量。您可以使用以下后缀之一将存储表示为普通整数或固定点号：E、P、T、G、M、k。您还可以使用两的指数：Ei、Pi、Ti、Gi、Mi Ki。

例如，以下数量全部代表大约相同的值：128974848、129e6、129M 和 123Mi。



重要

每个字节数量的后缀都是区分大小写的。务必使用正确的问题单。使用区分大小写的 "M"，如在 "400M" 中使用的，将请求设置为 400MB。使用区分大小写的 "400Mi" 来请求 400 mebibytes。如果您为临时存储指定 "400m"，则存储请求仅为 0.4 字节。

2.3.2. 临时存储请求和限值示例

以下示例配置文件显示了一个具有两个容器的 pod：

- 每个容器请求 2GiB 本地临时存储。
- 每个容器限制为 4GiB 本地临时存储。
- 在 pod 级别，kubelet 通过添加该 pod 中所有容器的限制来达到总体 pod 存储限制。
  - 在本例中，pod 级别的总存储使用量是所有容器的磁盘用量总和，以及 pod 的 **emptyDir** 卷。
  - 因此，pod 的请求为 4GiB 本地临时存储，限值为 8GiB 本地临时存储。

使用配额和限值的临时存储配置示例

```
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"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

1. 容器请求本地临时存储。
2. 本地临时存储的容器限制。

2.3.3. 临时存储配置会影响 pod 调度和驱除

pod 规格中的设置会影响调度程序如何决定调度 pod 以及 kubelet 驱除 pod。

- 首先，调度程序确保调度容器的资源请求总和小于节点的容量。在这种情况下，只有在可用临时存储（可分配资源）超过 4GiB 时，pod 才能分配给节点。
- 其次，在容器级别上，因为第一个容器设置了资源限值，kubelet 驱除管理器会测量此容器的磁盘用量，并在此容器的存储使用量超过其限制时驱除 pod (4GiB)。如果总用量超过总体 pod 存储限制(8GiB)，kubelet 驱除管理器也会标记驱除 pod。

有关为项目定义配额的详情，请参考每个项目的配额设置。

2.4. 监控临时存储

您可以使用 **/bin/df** 作为监控临时容器数据所在卷的临时存储使用情况的工具，即 **/var/lib/kubelet** 和 **/var/lib/containers**。如果集群管理员将 **/var/lib/containers** 放置在单独的磁盘上，则可以使用 **df** 命令来显示 **/var/lib/kubelet** 的可用空间。

要在 **/var/lib** 中显示已用和可用空间的信息，请输入以下命令：

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

输出显示 **/var/lib** 中的临时存储使用情况：

输出示例

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

第 3 章 了解持久性存储

3.1. 持久性存储概述

管理存储与管理计算资源不同。OpenShift Container Platform 使用 Kubernetes 持久性卷 (PV) 框架来允许集群管理员为集群提供持久性存储。开发者可以使用持久性卷声明 (PVC) 来请求 PV 资源而无需具体了解底层存储基础架构。

PVC 是特定于一个项目的，开发人员可创建并使用它作为使用 PV 的方法。PV 资源本身并不特定于某一个项目; 它们可以在整个 OpenShift Container Platform 集群间共享，并可以被任何项目使用。在 PV 绑定到 PVC 后，就不会将 PV 绑定到额外的 PVC。这意味着这个绑定 PV 被限制在一个命名空间（绑定的项目）中。

PV 由 **PersistentVolume** API 对象定义，它代表了集群中现有存储的片段，这些存储可以由集群管理员静态置备，也可以使用 **StorageClass** 对象动态置备。它与一个节点一样，是一个集群资源。

PV 是卷插件，与 **Volumes** 资源类似，但PV 的生命周期独立于任何使用它的 pod。PV 对象获取具体存储（NFS、iSCSI 或者特定 cloud-provider 的存储系统）的实现详情。

重要

存储的高可用性功能由底层的存储架构提供。

PVC 由 **PersistentVolumeClaim** API 项定义，它代表了开发人员对存储的一个请求。它与一个 pod 类似，pod 会消耗节点资源， PVC 消耗 PV 资源。例如：pod 可以请求特定级别的资源，比如 CPU 和内存，而 PVC 可以请求特定的存储容量和访问模式。例如：它们可以被加载为“只允许加载一次，可读写”，或“可以加载多次，只读”。

3.2. 卷和声明的生命周期

PV 是集群中的资源。PVC 是对这些资源的请求，也是对该资源的声明检查。PV 和 PVC 之间的交互有以下生命周期。

3.2.1. 置备存储

根据 PVC 中定义的开发人员的请求，集群管理员配置一个或者多个动态置备程序用来置备存储及一个匹配的 PV。

另外，集群管理员也可以预先创建多个 PV，它们包含了可用存储的详情。PV 存在于 API 中，且可以被使用。

3.2.2. 绑定声明

当您创建 PVC 时，您会要求特定的存储量，指定所需的访问模式，并创建一个存储类来描述和分类存储。master 中的控制循环会随时检查是否有新的 PVC，并把新的 PVC 与一个适当的 PV 进行绑定。如果没有适当的 PV，则存储类的置备程序会创建一个适当的 PV。

所有 PV 的大小可能会超过 PVC 的大小。这在手动置备 PV 时尤为如此。要最小化超额，OpenShift Container Platform 将会把 PVC 绑定到匹配所有其他标准的最小 PV。

如果匹配的卷不存在，或者相关的置备程序无法创建所需的存储，则请求将会处于未绑定的状态。当出现了匹配的卷时，相应的声明就会与其绑定。例如：在一个集群中有多个手动置备的 50Gi 卷。它们无法和一个请求 100Gi 的 PVC 相匹配。当在这个集群中添加了一个 100Gi PV 时， PVC 就可以和这个 PV 绑

定。

3.2.3. 使用 pod 和声明的 PV

pod 使用声明（claim）作为卷。集群通过检查声明来找到绑定的卷，并为 pod 挂载相应的卷。对于那些支持多个访问模式的卷，您必须指定作为 pod 中的卷需要使用哪种模式。

一旦您的声明被绑定后，被绑定的 PV 就会专属于您，直到您不再需要它。您可以通过在 pod 的 volumes 定义中包括 **persistentVolumeClaim** 来调度 pod 并访问声明的 PV。

注意

如果将具有高文件数的持久性卷附加到 pod，则这些 pod 可能会失败，或者可能需要很长时间才能启动。如需更多信息，请参阅在 OpenShift 中使用具有高文件计数的持久性卷时，为什么 pod 无法启动或占用大量时间来实现"Ready"状态？

3.2.4. 使用中的存储对象保护

使用中的存储对象保护功能确保了被 pod 使用的活跃的 PVC 以及与其绑定的 PV 不会从系统中移除，如果删除它们可能会导致数据丢失。

使用中的存储对象保护功能被默认启用。

注意

当使用 PVC 的 **Pod** 对象存在时，这个 PVC 被认为是被 Pod 使用的活跃的 PVC。

如果用户删除一个被 pod 使用的活跃的 PVC，这个 PVC 不会被立刻删除。这个删除过程会延迟到 PVC 不再被 pod 使用时才进行。另外，如果集群管理员删除了绑定到 PVC 的 PV，这个 PV 不会被立即删除。这个删除过程会延迟到 PV 不再绑定到 PVC 时才进行。

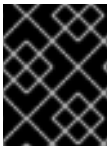
3.2.5. 释放持久性卷

当不再需要使用一个卷时，您可以从 API 中删除 PVC 对象，这样相应的资源就可以被重新声明。当声明被删除后，这个卷就被认为是已被释放，但它还不可以被另一个声明使用。这是因为之前声明者的数据仍然还保留在卷中，这些数据必须根据相关政策进行处理。

3.2.6. 持久性卷的重新声明策略

持久性卷重新声明（reclaim）策略指定了在卷被释放后集群可以如何使用它。卷重新声明政策包括 **Retain**、**Recycle** 或 **Delete**。

- **Retain** 策略可为那些支持它的卷插件手动重新声明资源。
- **Recycle** 策略在从其请求中释放后，将卷重新放入到未绑定的持久性卷池中。



重要

在 OpenShift Container Platform 4 中， **Recycle** 重新声明策略已被弃用。我们推荐使用动态置备功能。

- **Delete** 策略删除 OpenShift Container Platform 以及外部基础架构(如 Amazon EBS 或者 VMware vSphere)中相关的存储资源中的 **PersistentVolume**。

注意

动态置备的卷总是被删除。

3.2.7. 手动重新声明持久性卷

删除持久性卷生命（PVC）后，持久性卷（PV）仍然存在，并被视为"released（释放）"。但是，由于之前声明的数据保留在卷中，所以无法再使用 PV。

流程

要以集群管理员的身份手动重新声明 PV:

1. 删除 PV。

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

外部基础架构（如 AWS EBS、GCE PD、Azure Disk 或 Cinder 卷）中的关联的存储资产在 PV 被删除后仍然存在。

2. 清理相关存储资产中的数据。

3. 删除关联的存储资产。另外，若要重复使用同一存储资产，请使用存储资产定义创建新 PV。

重新声明的 PV 现在可供另一个 PVC 使用。

3.2.8. 更改持久性卷的重新声明策略

更改持久性卷的重新声明策略：

1. 列出集群中的持久性卷：

```
$ oc get pv
```

输**出示例**

```
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. 选择一个持久性卷并更改其重新声明策略：

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

3. 验证您选择的持久性卷是否具有正确的策略：

```
$ oc get pv
```

输**出示例**

```
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
```

在前面的输出中，绑定到声明 **default/claim3** 的卷现在具有 **Retain** 重新声明策略。当用户删除声明 **default/claim3** 时，这个卷不会被自动删除。

3.3. 持久性卷（PV）

每个 PV 都会包括一个 **spec** 和 **status**，它们分别代表卷的规格和状态，例如：

PersistentVolume 对象定义示例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 5Gi (2)
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain (4)
  ...
status:
  ...
```

1. 持久性卷的名称。
2. 卷可以使用的存储容量。
3. 访问模式，用来指定读写权限及挂载权限。
4. 重新声明策略，指定在资源被释放后如何处理它。

3.3.1. PV 类型

OpenShift Container Platform 支持以下持久性卷插件：

- AliCloud Disk
- AWS Elastic Block Store (EBS)
- AWS Elastic File Store (EFS)
- Azure Disk
- Azure File

- Cinder
- Fibre Channel
- GCP 持久性磁盘
- GCP Filestore
- IBM Power Virtual Server Block
- IBM® VPC Block
- HostPath
- iSCSI
- 本地卷
- NFS
- OpenStack Manila
- Red Hat OpenShift Data Foundation
- VMware vSphere

3.3.2. 容量

一般情况下，一个持久性卷（PV）有特定的存储容量。这可以通过使用 PV 的 **capacity** 属性来设置。

目前，存储容量是唯一可以设置或请求的资源。以后可能会包括 IOPS 、 throughput 等属性。

3.3.3. 访问模式

一个持久性卷可以以资源供应商支持的任何方式挂载到一个主机上。不同的供应商具有不同的功能，每个 PV 的访问模式可以被设置为特定卷支持的特定模式。例如：NFS 可以支持多个读写客户端，但一个特定的 NFS PV 可能会以只读方式导出。每个 PV 都有自己一组访问模式来描述指定的 PV 功能。

声明会与有类似访问模式的卷匹配。用来进行匹配的标准只包括访问模式和大小。声明的访问模式代表一个请求。比声明要求的条件更多的资源可能会匹配，而比要求的条件更少的资源则不会被匹配。例如：如果一个声明请求 RWO，但唯一可用卷是一个 NFS PV（RWO+ROX+RWX），则该声明与这个 NFS 相匹配，因为它支持 RWO。

系统会首先尝试直接匹配。卷的模式必须与您的请求匹配，或包含更多模式。大小必须大于或等于预期值。如果两个卷类型（如 NFS 和 iSCSI）有相同的访问模式，则一个要求这个模式的声明可能会与其中任何一个进行匹配。不同的卷类型之间没有匹配顺序，在同时匹配时也无法选择特定的一个卷类型。

所有有相同模式的卷都被分组，然后按大小（由小到大）进行排序。绑定程序会获取具有匹配模式的组群，并按容量顺序进行查找，直到找到一个大小匹配的项。。

重要

卷访问模式描述了卷功能。它们不会被强制限制。存储供应商会最终负责处理由于资源使用无效导致的运行时错误。供应商中的错误会在运行时作为挂载错误显示。

例如，NFS 提供 **ReadWriteOnce** 访问模式。如果要使用卷的 ROX 功能，请将声明标记为 **ReadOnlyMany**。

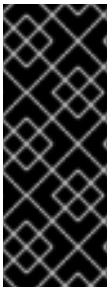
iSCSI 和 Fibre Channel（光纤通道）卷目前没有隔离机制。您必须保证在同一时间点上只在一个节点使用这些卷。在某些情况下，比如对节点进行 drain 操作时，卷可以被两个节点同时使用。在排空节点前，删除使用卷的 pod。

下表列出了访问模式：

表 3.1. 访问模式

访问模式	CLI 缩写	描述
ReadWriteOnce	**RWO**	卷只可以被一个节点以读写模式挂载。
ReadWriteOncePod [1]	**RWOP**	卷可以被一个单独的节点上的单个 pod 以读写模式挂载。
ReadOnlyMany	**ROX**	卷可以被多个节点以只读形式挂载。
ReadWriteMany	**RWX**	卷可以被多个节点以读写模式挂载。

1. 持久性卷的 ReadWriteOncePod 访问模式是一个技术预览功能。



重要

持久性卷的 ReadWriteOncePod 访问模式只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

表 3.2. 支持的持久性卷访问模式

卷插件	ReadWriteOnce [1]	ReadWriteOncePod [2]	ReadOnlyMany	ReadWriteMany
AliCloud Disk	▮	▮	-	-
AWS EBS [3]	▮	▮	-	-
AWS EFS	▮	▮	▮	▮

卷插件	ReadWriteOnce [1]	ReadWriteOncePod [2]	ReadOnlyMany	ReadWriteMany
Azure File	▮	▮	▮	▮
Azure Disk	▮	▮	-	-
Cinder	▮	▮	-	-
Fibre Channel	▮	▮	▮	▮[4]
GCP 持久性磁盘	▮	▮	-	-
GCP Filestore	▮	▮	▮	▮
HostPath	▮	▮	-	-
IBM Power Virtual Server 磁盘	▮	▮	▮	▮
IBM® VPC Disk	▮	▮	-	-
iSCSI	▮	▮	▮	▮[4]
本地卷	▮	▮	-	-
LVM 存储	▮	▮	-	-
NFS	▮	▮	▮	▮
OpenStack Manila	-	▮	-	▮
Red Hat OpenShift Data Foundation	▮	▮	-	▮
VMware vSphere	▮	▮	-	▮[5]

1. ReadWriteOnce（RWO）卷不能挂载到多个节点上。如果节点失败，系统不允许将附加的 RWO 卷挂载到新节点上，因为它已经分配给了故障节点。如果因此遇到多附件错误消息，请强制在关闭或崩溃的节点上删除 pod，以避免关键工作负载中的数据丢失，例如在附加动态持久性卷时。

2. ReadWriteOncePod 是一个技术预览功能。

3. 为依赖 AWS EBS 的 pod 使用重新创建的部署策略。

4. 只有原始块卷支持 ReadWriteMany (RWX) 访问模式用于光纤通道和 iSCSI。如需更多信息，请参阅 "Block volume support"。

5. 如果底层 vSphere 环境支持 vSAN 文件服务，则 OpenShift Container Platform 安装的 vSphere Container Storage Interface (CSI) Driver Operator 支持 provisioning of ReadWriteMany (RWX) 卷。如果您没有配置 vSAN 文件服务，且您请求 RWX，则卷将无法被创建，并记录错误。如需更多信息，请参阅"使用 Container Storage Interface" → "VMware vSphere CSI Driver Operator"。

3.3.4. 阶段

卷可以处于以下几个阶段：

表 3.3. 卷阶段

阶段	描述
Available	可用资源，还未绑定到任何声明。
Bound	卷已绑定到一个声明。
Released	以前使用这个卷的声明已被删除，但该资源还没有被集群重新声明。
Failed	卷的自动重新声明失败。

您可以运行以下命令来查看绑定到 PV 的 PVC 名称：

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

3.3.4.1. 挂载选项

您可以使用属性 **mountOptions** 在挂载 PV 时指定挂载选项。

例如：

挂载选项示例

```
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 在将 PV 挂载到磁盘时使用指定的挂载选项。

以下 PV 类型支持挂载选项：

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- GCE Persistent Disk
- iSCSI
- 本地卷
- NFS
- Red Hat OpenShift Data Foundation（仅限 Ceph RBD）
- VMware vSphere

注意

Fibre Channel 和 HostPath PV 不支持挂载选项。

其他资源

- ReadWriteMany vSphere 卷支持

3.4. 持久性卷声明 (PVC)

每个 **PersistentVolumeClaim** 对象都会包括一个 **spec** 和 **status**，它们分别代表了声明的规格和状态。例如：

PersistentVolumeClaim 对象定义示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim 1
```

```
spec:
  accessModes:
    - ReadWriteOnce (2)
  resources:
    requests:
      storage: 8Gi (3)
  storageClassName: gold (4)
status:
  ...
```

1. PVC 的名称。
2. 访问模式，用来指定读写权限及挂载权限。
3. PVC 可用的存储量。
4. 声明所需的 **StorageClass** 的名称。

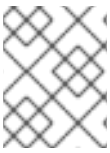
3.4.1. 存储类

另外，通过在 **storageClassName** 属性中指定存储类的名称，声明可以请求一个特定的存储类。只有具有请求的类的 PV（**storageClassName**的值与 PVC 中的值相同）才会与 PVC 绑定。集群管理员可配置动态置备程序为一个或多个存储类提供服务。集群管理员可根据需要创建与 PVC 的规格匹配的 PV。

重要

根据使用的平台，Cluster Storage Operator 可能会安装一个默认的存储类。此存储类由 Operator 拥有和控制。不能在定义注解和标签之外将其删除或修改。如果需要实现不同的行为，则必须定义自定义存储类。

集群管理员也可以为所有 PVC 设置默认存储类。当配置了默认存储类时， PVC 必须明确要求将存储类 **StorageClass** 或 **storageClassName**设为 **""**， 以便绑定到没有存储类的 PV。



注意

如果一个以上的存储类被标记为默认，则只能在 **storageClassName** 被显式指定时才能创建 PVC。因此，应只有一个存储类被设置为默认值。

3.4.2. 访问模式

声明在请求带有特定访问权限的存储时，使用与卷相同的格式。

3.4.3. Resources

象 pod 一样，声明可以请求具体数量的资源。在这种情况下，请求用于存储。同样的资源模型适用于卷和声明。

3.4.4. 声明作为卷

pod 通过将声明作为卷来访问存储。在使用声明时，声明需要和 pod 位于同一个命名空间。集群在 pod 的命名空间中找到声明，并使用它来使用这个声明后台的**PersistentVolume**。卷被挂载到主机和 pod 中，例如：

挂载卷到主机和 pod 示例

```
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 在 pod 中挂载卷的路径。

2 要挂载的卷的名称。不要挂载到容器 root、/ 或主机和容器中相同的任何路径。如果容器有足够权限，可能会损坏您的主机系统（如主机的 **/dev/pts** 文件）。使用 **/host** 挂载主机是安全的。

3 要使用的 PVC 名称（存在于同一命名空间中）。

3.5. 块卷支持

OpenShift Container Platform 可以静态置备原始块卷。这些卷没有文件系统。对于可以直接写入磁盘或者实现其自己的存储服务的应用程序来说，使用它可以获得性能优势。

原始块卷可以通过在 PV 和 PVC 规格中指定 **volumeMode: Block**来置备。

重要

使用原始块卷的 pod 需要配置为允许特权容器。

下表显示了哪些卷插件支持块卷。

表 3.4. 块卷支持

卷插件	手动置备	动态置备	完全支持
Amazon Elastic Block Store (Amazon EBS)	▮	▮	▮
Amazon Elastic File Storage (Amazon EFS)			
AliCloud Disk	▮	▮	▮
Azure Disk	▮	▮	▮

卷插件	手动置备	动态置备	完全支持
Azure File			
Cinder	▮	▮	▮
Fibre Channel	▮		▮
GCP	▮	▮	▮
HostPath			
IBM VPC 磁盘	▮	▮	▮
iSCSI	▮		▮
本地卷	▮		▮
LVM 存储	▮	▮	▮
NFS			
Red Hat OpenShift Data Foundation	▮	▮	▮
VMware vSphere	▮	▮	▮

重要

可手动置备但未提供完全支持的块卷作为技术预览功能提供。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

3.5.1. 块卷示例

PV 示例

```
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 需要把 **volumeMode** 设置为 **Block** 来代表这个 PV 是一个原始块卷。

PVC 示例

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

1 需要把 **volumeMode** 设置为 **Block** 来代表请求一个原始块 PVC。

Pod 规格示例

```
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 对于块设备，使用 **VolumeDevices**而不是 **volumeMounts**。只有 **persistentVolumeClaim** 源可以和原始块卷一起使用。

2 使用 **devicePath** 而不是 **mountPath** 来代表到原始块映射到系统的物理设备的路径。

3 卷源必须是 **persistentVolumeClaim** 类型，且必须与期望的 PVC 的名称匹配。

表 3.5. **volumeMode** 接受的值

值	默认
Filesystem	是
Block	否

表 3.6. 块卷的绑定方案

PV volumeMode	PVC volumeMode	绑定结果
Filesystem	Filesystem	绑定
Unspecified	Unspecified	绑定
Filesystem	Unspecified	绑定
Unspecified	Filesystem	绑定
Block	Block	绑定
Unspecified	Block	无绑定
Block	Unspecified	无绑定
Filesystem	Block	无绑定
Block	Filesystem	无绑定

重要

未指定值时将使用默认值 **Filesystem**。

3.6. 使用 FSGROUP 减少 POD 超时

如果存储卷包含很多文件（1,000,000 或更多），您可能会遇到 pod 超时问题。

这是因为，在默认情况下，OpenShift Container Platform 会递归更改每个卷内容的所有权和权限，以便在挂载卷时与 pod 的 **securityContext** 中指定的 **fsGroup** 匹配。对于大型卷，检查和更改所有权和权限可能会非常耗时，从而会减慢 pod 启动的速度。您可以使用 **securityContext** 中的 **fsGroupChangePolicy** 字段来控制 OpenShift Container Platform 检查和管理卷的所有权和权限的方式。

fsGroupChangePolicy 定义在 pod 中公开卷之前更改卷的所有权和权限的行为。此字段仅适用于支持 **fsGroup**- 控制的所有权和权限。此字段有两个可能的值：

- **OnRootMismatch** ：仅当 root 目录的权限和所有权与卷的预期权限不匹配时才会更改权限和所有权。这有助于缩短更改卷的所有权和权限所需的时间，以减少 pod 超时。

- **Always** ：当卷被挂载时，始终更改卷的权限和所有权。

fsGroupChangePolicy 示例

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

1 **OnRootMismatch** 指定跳过递归权限更改，这有助于避免 pod 超时问题。

注意

fsGroupChangePolicyfield 对临时卷类型没有影响，如 secret、configMap 和 emptydir。

第 4 章 配置持久性存储

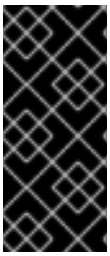
4.1. 使用 AWS ELASTIC BLOCK STORE 的持久性存储

OpenShift Container Platform 支持 Amazon Elastic Block Store (EBS) 卷。您可以使用 Amazon EC2 为 OpenShift Container Platform 集群置备持久性存储。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。您可以动态置备 Amazon EBS 卷。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。您可以定义 KMS 密钥来加密 AWS 上的 container-persistent 卷。默认情况下，新创建的使用 OpenShift Container Platform 版本 4.10 和更高版本的集群使用 gp3 存储和 AWS EBS CSI 驱动程序。

重要

存储的高可用性功能由底层存储供应商实现。



重要

OpenShift Container Platform 4.12 及更新的版本为 AWS Block in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。有关迁移的更多信息，请参阅 CSI 自动迁移。

4.1.1. 创建 EBS 存储类

存储类用于区分和划分存储级别和使用。通过定义存储类，用户可以获得动态置备的持久性卷。

4.1.2. 创建持久性卷声明

先决条件

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程

1. 在 OpenShift Container Platform 控制台中，点击 **Storage → Persistent Volume Claims**。
2. 在持久性卷声明概述页中，点 **Create Persistent Volume Claim**。
3. 在出现的页面中定义所需选项。
   a. 从下拉菜单中选择之前创建的存储类。
   b. 输入存储声明的唯一名称。
   c. 选择访问模式。此选择决定了存储声明的读写访问权限。
   d. 定义存储声明的大小。
4. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

4.1.3. 卷格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查卷是否包含由持久性卷定义中的 **fsType** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

此验证可让您将未格式化的 AWS 卷用作持久性卷，因为 OpenShift Container Platform 在首次使用前会进行格式化。

4.1.4. 一个节点上的 EBS 卷的最大数目

默认情况下，OpenShift Container Platform 最多支持把 39 个 EBS 卷附加到一个节点。这个限制与 AWS 卷限制一致。卷限制取决于实例类型。

重要

作为集群管理员，您必须使用树内或 Container Storage Interface（CSI）卷及其相应的存储类，但不得同时使用这两个卷类型。对于 in-tree 和 CSI 卷，最大附加的 EBS 卷数量会单独计算，因此每种类型您都最多可以有 39 个 EBS 卷。

有关访问额外存储选项（如卷快照）的详情，请参考 AWS Elastic Block Store CSI Driver Operator 。

4.1.5. 使用 KMS 密钥在 AWS 上加密容器持久性卷

在部署到 AWS 时，定义在 AWS 上加密容器持久性卷的 KMS 密钥很有用。

先决条件

- 底层基础架构必须包含存储。
- 您必须在 AWS 上创建客户 KMS 密钥。

流程

1. 创建存储类：

```
$ 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. 指定存储类的名称。
2. 在置备的卷中创建的文件系统。

3 指定加密 container-persistent 卷时要使用的密钥的完整 Amazon 资源名称 (ARN)。如果没有提供任何密钥，但 **encrypted** 字段被设置为 **true**，则使用默认的 KMS 密钥。请参阅

2. 使用指定 KMS 密钥的存储类创建 PVC：

```
$ 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. 创建工作负载容器以使用 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. 其他资源

- 有关访问额外存储选项的信息，如卷快照，请参阅 AWS Elastic Block Store CSI Driver Operator，这些内容无法在树状卷插件中使用。

4.2. 使用 AZURE 持久性存储

OpenShift Container Platform 支持 Microsoft Azure Disk 卷。您可以使用 Azure 为 OpenShift Container Platform 集群置备永久性存储。我们假设您对 Kubernetes 和 Azure 有一定的了解。Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。Azure 磁盘卷可以动态部署。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。

重要

OpenShift Container Platform 4.11 及之后的版本为 Azure Disk in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。有关迁移的更多信息，请参阅 CSI 自动迁移。

重要

存储的高可用性功能由底层的存储架构提供。

其他资源

- Microsoft Azure Disk

4.2.1. 创建 Azure 存储类

存储类用于区分和划分存储级别和使用。通过定义存储类，用户可以获得动态置备的持久性卷。

流程

1. 在 OpenShift Container Platform 控制台中点击 **Storage→ Storage Classes**。
2. 在存储类概述中，点击 **Create Storage Class**。
3. 在出现的页面中定义所需选项。
   a. 输入一个名称来指代存储类。
   b. 输入描述信息（可选）。
   c. 选择 reclaim 策略。
   d. 从下拉列表中选择 **kubernetes.io/azure-disk** 。
      i. 输入存储帐户类型。这与您的 Azure 存储帐户 SKU 层对应。有效选项包括 **Premium_LRS**、**PremiumV2_LRS**、**Standard_LRS**、**StandardSSD_LRS** 和 **UltraSSD_LRS**。

重要

所有区域都不支持 skuname **PremiumV2_LRS**，在一些支持的区域中也不支持所有可用区。如需更多信息，请参阅 Azure 文档。

      ii. 输入帐户类型。有效选项为 **shared**、**dedicated** 和 **managed**.

重要

红帽仅在存储类中支持使用 **kind: Managed**。

使用 **Shared** 和 **Dedicated** 时，Azure 会创建非受管磁盘，而 OpenShift Container Platform 为机器 OS（root）磁盘创建一个受管磁盘。但是，因为 Azure Disk 不允许在节点上同时使用受管和非受管磁盘，所以使用 **Shared** 或 **Dedicated** 创建的非受管磁盘无法附加到 OpenShift Container Platform 节点。

e. 根据需要为存储类输入附加参数。

4. 点 **Create** 创建存储类。

其他资源

- Azure Disk Storage Class

4.2.2. 创建持久性卷声明

先决条件

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程

1. 在 OpenShift Container Platform 控制台中，点击 **Storage → Persistent Volume Claims**。
2. 在持久性卷声明概述页中，点 **Create Persistent Volume Claim**。
3. 在出现的页面中定义所需选项。
   a. 从下拉菜单中选择之前创建的存储类。
   b. 输入存储声明的唯一名称。
   c. 选择访问模式。此选择决定了存储声明的读写访问权限。
   d. 定义存储声明的大小。
4. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

4.2.3. 卷格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查它是否包含由 **fstype** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

这将可以使用未格式化的 Azure 卷作为持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

4.2.4. 用于部署带有使用 PVC 的巨型磁盘的机器的机器集

您可以创建在 Azure 上运行的机器集，该机器集用来部署带有巨型磁盘的机器。ultra 磁盘是高性能存储，用于要求最苛刻的数据工作负载。

in-tree 插件和 CSI 驱动程序都支持使用 PVC 启用巨型 磁盘。您还可以在不创建 PVC 的情况下将巨型磁盘部署为数据磁盘。

其他资源

- Microsoft Azure ultra 磁盘文档
- 使用 CSI PVC 在强制磁盘上部署机器的机器集
- 在计算磁盘上部署机器的机器集作为数据磁盘

4.2.4.1. 使用机器集创建带有巨型磁盘的机器

您可以通过编辑机器集 YAML 文件在 Azure 上部署带有巨型磁盘的机器。

先决条件

- 已有 Microsoft Azure 集群。

流程

1. 运行以下命令，复制现有的 Azure **MachineSet** 自定义资源(CR)并编辑它：

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

   其中 **<machine-set-name>** 是您要使用巨型磁盘置备机器的机器集。

2. 在指示的位置中添加以下行：

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

   1 指定标签，用于选择此机器集创建的节点。此流程使用 **disk.ulssd** 用于这个值。

   2 这些行支持使用 ultra 磁盘。

3. 运行以下命令，使用更新的配置创建机器集：

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

4. 创建一个包含以下 YAML 定义的存储类：

```
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) 指定存储类的名称。此流程使用 **ultra-disk-sc** 作为这个值。

(2) 指定存储类的 IOPS 数量。

(3) 指定存储类的吞吐量，单位为 MBps。

(4) 对于 Azure Kubernetes Service(AKS)版本 1.21 或更高版本，请使用 **disk.csi.azure.com**。对于 AKS 的早期版本，请使用 **kubernetes.io/azure-disk**。

(5) 可选：指定此参数以等待创建使用磁盘的 pod。

5. 创建一个持久性卷声明(PVC)来引用包含以下 YAML 定义的 **ultra-disk-sc** 存储类：

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

(1) 指定 PVC 的名称。此流程使用 **ultra-disk** 作为这个值。

(2) 此 PVC 引用了 **ultra-disk-sc** 存储类。

(3) 指定存储类的大小。最小值为 **4Gi**。

6. 创建包含以下 YAML 定义的 pod：

```
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 指定启用巨型磁盘的机器集标签。此流程使用 **disk.ulssd** 用于这个值。

2 这个 pod 引用了 **ultra-disk** PVC。

验证

1. 运行以下命令验证机器是否已创建：

```
$ oc get machines
```

机器应处于 **Running** 状态。

2. 对于正在运行并附加节点的机器，请运行以下命令验证分区：

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

在这个命令中，**oc debug node/<node-name>** 会在节点 **<node-name>** 上启动一个 debugging shell，并传递一个带有 **--** 的命令。传递的命令 **chroot /host** 提供对底层主机操作系统二进制文件的访问，**lsblk** 显示连接至主机操作系统计算机的块设备。

后续步骤

- 要在 pod 中使用大量磁盘，请创建使用挂载点的工作负载。创建一个类似以下示例的 YAML 文件：

```
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. 启用 ultra 磁盘的机器集的故障排除资源

使用本节中的信息从您可能会遇到的问题了解和恢复。

4.2.4.2.1. 无法挂载由巨型磁盘支持的持久性卷声明

如果挂载了被巨型磁盘支持的持久性卷声明的问题，pod 会一直处于 **ContainerCreating** 状态，并触发警报。

例如，如果没有在支持托管 pod 的节点的机器上设置 **additionalCapabilities.ultraSSDEnabled** 参数，则会出现以下出错信息：

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

- 要解决这个问题，请运行以下命令来描述 pod：

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

4.3. 使用 AZURE FILE 的持久性存储

OpenShift Container Platform 支持 Microsoft Azure File 卷。您可以使用 Azure 为 OpenShift Container Platform 集群置备永久性存储。我们假设您对 Kubernetes 和 Azure 有一定的了解。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。您可以动态置备 Azure File 卷。

持久性卷不绑定到单个项目或命名空间，您可以在 OpenShift Container Platform 集群中共享它们。持久性卷声明是针对某个项目或命名空间的，应用程序中可以请求它的用户。

重要

存储的高可用性功能由底层的存储架构提供。

重要

Azure File 卷使用服务器消息块。

重要

OpenShift Container Platform 4.13 及更新的版本为 Azure File in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。有关迁移的更多信息，请参阅 CSI 自动迁移。

其他资源

- Azure File

4.3.1. 创建 Azure File 共享持久性卷声明

要创建持久性卷声明，您必须首先定义一个包含 Azure 帐户和密钥的 **Secret** 对象。此 secret 在 **PersistentVolume** 定义中使用，应用程序中使用的持久性卷声明将引用该 secret。

先决条件

- 已存在 Azure File 共享。
- 有访问此共享所需的凭证，特别是存储帐户和密钥。

流程

1. 创建包含 Azure File 凭证的 **Secret** 对象：

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

1. Azure File 存储帐户名称。
2. Azure File 存储帐户密钥。

2. 创建引用您创建的 **Secret** 对象的 **PersistentVolume** 对象：

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  storageClassName: azure-file-sc
  azureFile:
    secretName: <secret-name> 3
    shareName: share-1 4
    readOnly: false
```

1. 持久性卷的名称。
2. 此持久性卷的大小。
3. 包含 Azure File 共享凭证的 secret 名称。
4. Azure File 共享的名称。

3. 创建映射到您创建的持久性卷的 **PersistentVolumeClaim** 对象：

```
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "claim1" (1)
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi" (2)
  storageClassName: azure-file-sc (3)
  volumeName: "pv0001" (4)
```

1. 持久性卷声明的名称。
2. 持久性卷声明的大小。
3. 用于置备持久性卷的存储类的名称。指定 **PersistentVolume** 定义中使用的存储类。
4. 引用 Azure File 共享的现有 **PersistentVolume** 对象的名称。

4.3.2. 在 pod 中挂载 Azure File 共享

创建持久性卷声明后，应用程序就可以使用它。以下示例演示了在 pod 中挂载此共享。

先决条件

- 已存在一个映射到底层 Azure File 共享的持久性卷声明。

流程

- 创建可挂载现有持久性卷声明的 pod:

```
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)
```

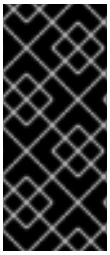
1. pod 的名称。
2. 在 pod 中挂载 Azure File 共享的路径。不要挂载到容器 root、/ 或主机和容器中相同的任何路径。如果容器有足够权限，可能会损坏您的主机系统（如主机的 **/dev/pts** 文件）。使用 **/host** 挂载主机是安全的。

3 之前创建的 **PersistentVolumeClaim** 对象的名称。

4.4. 使用 CINDER 的持久性存储

OpenShift Container Platform 支持 OpenStack Cinder。我们假设您对 Kubernetes 和 OpenStack 有一定的了解。

Cinder 卷可以动态置备。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。



重要

OpenShift Container Platform 4.11 及之后的版本为 Cinder in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。有关迁移的更多信息，请参阅 CSI 自动迁移。

其他资源

- 如需了解有关 OpenStack Block Storage 如何为虚拟硬盘提供持久块存储管理的信息，请参阅 OpenStack Cinder。

4.4.1. 使用 Cinder 手动置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

先决条件

- 为 Red Hat OpenStack Platform（RHOSP）配置 OpenShift Container Platform
- Cinder 卷 ID

4.4.1.1. 创建持久性卷

您必须在对象定义中定义持久性卷 (PV)，然后才能在 OpenShift Container Platform 中创建它：

流程

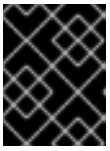
1. 将对象定义保存到文件中。

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. 持久性卷声明或 Pod 使用的卷名称。
2. 为这个卷分配的存储量。
3. 为 Red Hat OpenStack Platform（RHOSP）Cinder 卷指定 **cinder**。
4. 当这个卷被第一次挂载时，文件系统会被创建。
5. 要使用的 Cinder 卷。



重要

在卷被格式化并置备后，不要更改 **fstype** 参数的值。更改此值可能会导致数据丢失和 pod 失败。

2. 创建在上一步中保存的对象定义文件。

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

4.4.1.2. 持久性卷格式化

因为 OpenShift Container Platform 在首次使用卷前会进行格式化，所以可以使用未格式化的 Cinder 卷作为 PV。

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查在 PV 定义中是否包含由 **fsType** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

4.4.1.3. Cinder 卷安全

如果在应用程序中使用 Cinder PV，请在其部署配置中配置安全性。

先决条件

- 必须创建一个使用适当 **fsGroup** 策略的 SCC。

流程

1. 创建一个服务帐户并将其添加到 SCC：

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

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

2. 在应用程序的部署配置中，提供服务帐户名称和 **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> (6)
      securityContext:
        fsGroup: 7777 (7)
```

1. 要运行的 pod 的副本数。
2. 要运行的 pod 的标签选择器。
3. 控制器创建的 pod 模板。
4. pod 上的标签。它们必须包含标签选择器中的标签。
5. 扩展任何参数后的最大名称长度为 63 个字符。
6. 指定您创建的服务帐户。
7. 为 pod 指定 **fsGroup**。

4.5. 使用 FIBRE CHANNEL 持久性存储

OpenShift Container Platform 支持 Fibre Channel，它允许您使用 Fibre Channel 卷为 OpenShift Container Platform 集群提供持久性存储。我们假设您对 Kubernetes 和 Fibre Channel 有一定的了解。

重要

基于 ARM 架构的基础架构不支持使用 Fibre Channel 的持久性存储。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。

重要

存储的高可用性功能由底层的存储架构提供。

其他资源

- 使用光纤通道设备

4.5.1. 置备

要使用 **PersistentVolume** API 置备 Fibre Channel 卷，必须提供以下内容：

- **targetWWNs** (Fibre Channel 阵列目标的 World Wide Names）。
- 一个有效的 LUN 号码。
- 文件系统类型。

持久性卷和 LUN 之间有一个一对一的映射。

先决条件

- Fibre Channel LUN 必须存在于底层系统中。

PersistentVolume 对象定义

```
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 全局广泛的标识符(WWID)。FC **wwids** 或 FC 目标 **WWN** 和 **lun** 的组合必须设置，但不能同时设置。建议在 WWN 目标中使用 FC WWID 标识符，因为它可以保证每个存储设备独有，并且独立于用于访问该设备的路径。通过发出 SCSI Indentification Vital Product Data（**page 0x83** ）或单元 Serial Number（**page 0x80**）来获得 WWID 标识符。FC WWID 被标识为 **/dev/disk/by-id/** 来引用磁盘上的数据，即使设备的路径发生了变化，即使从不同系统访问该设备也是如此。

2 3 Fibre Channel WWN 由 **/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>** 代表，但您不需要提供 **WWN**之前（包括**0x**）和以后（包括 **-**）的部分。

重要

在卷被格式化并置备后，修改 **fstype** 参数的值会导致数据丢失和 pod 失败。

4.5.1.1. 强制磁盘配额

使用 LUN 分区强制磁盘配额和大小限制。每个 LUN 都被映射到一个持久性卷，持久性卷必须使用唯一的名称。

采用这种方法强制配额可让最终用户以特定数量（如 10Gi）请求持久性存储，并可与相等或更大容量的卷进行匹配。

4.5.1.2. Fibre Channel 卷安全

用户使用持久性卷声明来请求存储。这个声明只在用户的命名空间中有效，且只能被同一命名空间中的 pod 使用。任何尝试访问命名空间中的持久性卷都会导致 pod 失败。

每个 Fibre Channel LUN 必须可以被集群中的所有节点访问。

4.6. 使用 FLEXVOLUME 的持久性存储

重要

FlexVolume 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

虚拟机监控程序 Container Storage Interface(CSI)驱动程序是在 OpenShift Container Platform 中写入卷驱动程序的建议方法。FlexVolume 驱动程序的维护者应该实施 CSI 驱动程序，并将 FlexVolume 用户移到 CSI。FlexVolume 的用户应该将其工作负载移到 CSI 驱动程序。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

OpenShift Container Platform 支持 FlexVolume，这是一个树外插件，使用可执行模型与驱动程序进行接口。

要从没有内置插件的后端使用存储，您可以通过 FlexVolume 驱动程序来扩展 OpenShift Container Platform，并为应用程序提供持久性存储。

Pod 通过 **flexvolume** 树内插件与 FlexVolume 驱动程序交互。

其他资源

- 扩展持久性卷

4.6.1. 关于 FlexVolume 驱动程序

FlexVolume 驱动程序是一个可执行文件，它位于集群中所有节点的一个明确定义的目录中。OpenShift Container Platform 会在需要挂载或卸载由带有 **flexVolume** 的 **PersistentVolume** 对象代表的卷时调用 FlexVolume 驱动程序。

重要

OpenShift Container Platform 不支持 FlexVolume 的 attach 和 detach 操作。

4.6.2. FlexVolume 驱动程序示例

FlexVolume 驱动程序的第一个命令行参数始终是一个操作名称。其他参数都针对于每个操作。大多数操作都使用 JSON 字符串作为参数。这个参数是一个完整的 JSON 字符串，而不是包括 JSON 数据的文件名称。

FlexVolume 驱动程序包含：

- 所有 **flexVolume.options**。
- **flexVolume** 的一些选项带有 **kubernetes.io/**前缀，如 **fsType** 和 **readwrite**。
- 如果使用 secret，secret 的内容带有 **kubernetes.io/secret/** 前缀。

FlexVolume 驱动程序 JSON 输入示例

```
{
 "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. **flexVolume.options** 中的所有选项。
2. **flexVolume.fsType** 的值。
3. 基于 **flexVolume.readOnly** 的 **ro**/**rw**。
4. 由 **flexVolume.secretRef**引用的 secret 的所有键及其值。

OpenShift Container Platform 需要有关驱动程序标准输出的 JSON 数据。如果没有指定，输出会描述操作的结果。

FlexVolume 驱动程序默认输出示例

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

驱动程序的退出代码应该为 **0**（成功），或 **1**（失败）。

操作应该是"幂等"的，这意味着挂载一个已被挂载的卷的结果是一个成功的操作。

4.6.3. 安装 FlexVolume 驱动程序

用于扩展 OpenShift Container Platform 的 FlexVolume 驱动程序仅在节点上执行。要实现 FlexVolume，需要调用的操作列表和安装路径都是必需的。

先决条件

- FlexVolume 驱动程序必须实现以下操作：

init

初始化驱动程序。它会在初始化所有节点的过程中被调用。

- 参数: 无
- 执行于：节点
- 预期输出：默认 JSON

mount

挂载一个卷到目录。这可包括挂载该卷所需的任何内容，包括查找该设备，然后挂载该设备。

- 参数: **<mount-dir> <json>**
- 执行于：节点
- 预期输出：默认 JSON

unmount

从目录中卸载卷。这可以包括在卸载后清除卷所必需的任何内容。

- 参数: **<mount-dir>**
- 执行于：节点
- 预期输出：默认 JSON

mountdevice

将卷的设备挂载到一个目录，然后 pod 可以从这个目录绑定挂载。

这个 call-out 不会传递 FlexVolume spec 中指定的 "secrets"。如果您的驱动需要 secret，不要实现这个 call-out。

- 参数: **<mount-dir> <json>**
- 执行于：节点
- 预期输出：默认 JSON

unmountdevice

从目录中卸载卷的设备。

- 参数: **<mount-dir>**
- 执行于：节点
- 预期输出：默认 JSON
  - 所有其他操作都应该返回带有 **{"status"："Not supported"}** 及退出代码 **1** 的 JSON。

流程

安装 FlexVolume 驱动程序：

1. 确保可执行文件存在于集群中的所有节点上。

2. 将可执行文件放在卷插件路径: **/etc/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>**。

例如，要为存储 **foo** 安装 FlexVolume 驱动程序，请将可执行文件放在: **/etc/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo**。

4.6.4. 使用 FlexVolume 驱动程序消耗存储

OpenShift Container Platform 中的每个 **PersistentVolume** 都代表存储后端中的一个存储资产，例如一个卷。

流程

- 使用 **PersistentVolume** 对象来引用已安装的存储。

使用 FlexVolume 驱动程序示例定义持久性卷对象

```
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. 卷的名称。这是如何通过持久性卷声明或从 pod 识别它。这个名称可以与后端存储中的卷的名称不同。
2. 为这个卷分配的存储量。
3. 驱动程序的名称。这个字段是必须的。
4. 卷中的文件系统。这个字段是可选的。
5. 对 secret 的引用。此 secret 中的键和值在调用时会提供给 FlexVolume 驱动程序。这个字段是可选的。
6. read-only 标记。这个字段是可选的。
7. FlexVolume 驱动程序的额外选项。除了用户在 **options** 字段中指定的标记外，以下标记还会传递给可执行文件：

```
"fsType":"<FS type>",
"readwrite":"<rw>",
```

```
"secret/key1":"<secret1>"
...
"secret/keyN":"<secretN>"
```

注意

secret 只会传递到 mount 或 unmount call-outs。

4.7. 使用 GCE PERSISTENT DISK 的持久性存储

OpenShift Container Platform 支持 GCE Persistent Disk 卷 (gcePD)。您可以使用 GCE 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 GCE 有一定的了解。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。

GCE Persistent Disk 卷可以动态部署。

持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。

重要

OpenShift Container Platform 4.12 及更新的版本为 GCE Persist Disk in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。

有关迁移的更多信息，请参阅 CSI 自动迁移。

重要

存储的高可用性功能由底层的存储架构提供。

其他资源

- GCE Persistent Disk

4.7.1. 创建 GCE 存储类

存储类用于区分和划分存储级别和使用。通过定义存储类，用户可以获得动态置备的持久性卷。

4.7.2. 创建持久性卷声明

先决条件

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程

1. 在 OpenShift Container Platform 控制台中，点击 **Storage → Persistent Volume Claims**。

2. 在持久性卷声明概述页中，点 **Create Persistent Volume Claim**。
3. 在出现的页面中定义所需选项。
   a. 从下拉菜单中选择之前创建的存储类。
   b. 输入存储声明的唯一名称。
   c. 选择访问模式。此选择决定了存储声明的读写访问权限。
   d. 定义存储声明的大小。
4. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

4.7.3. 卷格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查卷是否包含由持久性卷定义中的 **fsType** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

此验证可让您将未格式化的 GCE 卷用作持久性卷，因为 OpenShift Container Platform 在首次使用前会进行格式化。

4.8. 使用 ISCSI 的持久性存储

您可以使用 iSCSI为 OpenShift Container Platform 集群提供持久性存储。我们假设您对 Kubernetes 和 iSCSI 有一定的了解。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。

重要

存储的高可用性功能由底层存储供应商实现。

重要

当您在 Amazon Web Services 上使用 iSCSI 时，必须更新默认的安全策略，使其包含 iSCSI 端口中节点间的 TCP 流量。默认情况下，它们是端口 **860** 和 **3260**。

重要

用户必须通过安装 **iscsi-initiator-utils** 软件包并在 **/etc/iscsi/initiatorname.iscsi** 中配置启动器名称，确保所有 OpenShift Container Platform 节点上已配置了 iSCSI 启动器。**iscsi-initiator-utils** 软件包已在使用 Red Hat Enterprise Linux CoreOS (RHCOS) 的部署中安装。

如需更多信息，请参阅管理存储设备。

4.8.1. 置备

在将存储作为卷挂载到 OpenShift Container Platform 之前，请确认它已存在于底层的基础架构中。iSCSI 需要的是 iSCSI 目标门户，一个有效的 iSCSI 限定名称 (IQN)，一个有效的 LUN 号码，文件系统类型，以及 **persistenceVolume** API。

PersistentVolume 对象定义

```
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. 强制磁盘配额

使用 LUN 分区强制磁盘配额和大小限制。每个 LUN 都是一个持久性卷。kubernetes 为持久性卷强制使用唯一的名称。

以这种方式强制配额可让最终用户以特定数量（如 **10Gi**）请求持久性存储，并与相等或更大容量的对应卷匹配。

4.8.3. iSCSI 卷安全

用户使用 **PersistentVolumeClaim** 对象请求存储。这个声明只在用户的命名空间中有效，且只能被在同一命名空间中的 pod 调用。尝试使用其他命名空间中的持久性卷声明会导致 pod 失败。

每个 iSCSI LUN 都需要可以被集群中的所有节点访问。

4.8.3.1. Challenge Handshake Authentication Protocol (CHAP) 配置

另外，OpenShift Container Platform 可以使用 CHAP 在 iSCSI 目标中验证自己：

```
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. 启用 iSCSI 发现的 CHAP 验证。
2. 启用 iSCSI 会话的 CHAP 验证。
3. 使用用户名 + 密码指定 Secrets 对象的名称。该 **Secret** 对象必须在所有可使用引用卷的命名空间中可用。

4.8.4. iSCSI 多路径

对于基于 iSCSI 的存储，您可以使用相同的 IQN 为多个目标入口 IP 地址配置多路径。通过多路径，当路径中的一个或者多个组件失败时，仍可保证对持久性卷的访问。

要在 pod 规格中指定多路径，请使用 **portals** 字段。例如：

```
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. 使用 **portals** 字段添加额外的目标门户。

4.8.5. iSCSI 自定义 initiator IQN

如果 iSCSI 目标仅限于特定的 IQN，则配置自定义 initiator iSCSI 限定名称 (IQN)，但不会保证 iSCSI PV 附加到的节点具有这些 IQN。

使用 **initiatorName** 字段指定一个自定义 initiator IQN。

```
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) 指定 initiator 的名称。

4.9. 使用 NFS 的持久性存储

OpenShift Container Platform 集群可以使用 NFS 来置备持久性存储。持久性卷 (PV) 和持久性卷声明 (PVC) 提供了在项目间共享卷的方法。虽然 PV 定义中包含的与 NFS 相关的信息也可以直接在 **Pod** 中定义，但是这样做不会使创建的卷作为一个特定的集群资源，从而可能会导致卷冲突。

其他资源

- 挂载 NFS 共享

4.9.1. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。要置备 NFS 卷，则需要一个 NFS 服务器和导出路径列表。

流程

1. 为 PV 创建对象定义：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 5Gi (2)
  accessModes:
  - ReadWriteOnce (3)
  nfs: (4)
    path: /tmp (5)
    server: 172.17.0.2 (6)
  persistentVolumeReclaimPolicy: Retain (7)
```

(1) 卷的名称。这是各个 **oc <command> pod** 命令中的 PV 标识。

(2) 为这个卷分配的存储量。

(3) 虽然这看上去象是设置对卷的访问控制，但它实际上被用作标签并用来将 PVC 与 PV 匹配。当前，还不能基于 **accessModes** 强制访问规则。

(4) 使用的卷类型，在这个示例里是 **nfs** 插件。

(5) NFS 服务器导出的路径。

6 NFS 服务器的主机名或 IP 地址。

7 PV 的 reclaim 策略。它决定了在卷被释放后会发生什么。

注意

每个 NFS 卷都必须由集群中的所有可调度节点挂载。

2. 确定创建了 PV：

```
$ oc get pv
```

输出示例

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

3. 创建绑定至新 PV 的持久性卷声明：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce 1
  resources:
    requests:
      storage: 5Gi 2
  volumeName: pv0001
  storageClassName: ""
```

1 访问模式不强制实施安全性，而是作为标签来将 PV 与 PVC 匹配。

2 此声明会寻找提供 **5Gi** 或更高容量的 PV。

4. 确认创建了持久卷声明：

```
$ oc get pvc
```

输出示例

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

4.9.2. 强制磁盘配额

使用磁盘分区强制磁盘配额和大小限制。每个分区都可以有自己的导出。每个导出都是一个 PV。OpenShift Container Platform 会保证每个 PV 都使用不同的名称，但 NFS 卷服务器和路径的唯一性是由管理员实现的。

采用这种方法强制配额可让软件开发人员以特定数量（如 10Gi）请求持久性存储，并可与相等或更大容量的卷进行匹配。

4.9.3. NFS 卷安全

这部分论述了 NFS 卷安全性，其中包括匹配的权限和 SELinux 考虑。用户需要了解 POSIX 权限、进程 UID 、supplemental 组和 SELinux 的基本知识。

软件开发人员可以使用 PVC 名称，或直接在 **Pod** 定义中 **volumes** 部分使用 NFS 插件来请求 NFS 存储。

NFS 服务器中的 **/etc/exports** 文件包含可访问的 NFS 目录。目标 NFS 目录有 POSIX 拥有者和组群 ID。OpenShift Container Platform NFS 插件使用相同的 POSIX 所有者权限及在导出的 NFS 目录中找到的权限挂载容器的 NFS 目录。然而，容器实际运行时所使用的 UID 与 NFS 挂载的所有者的 UID 不同。这是所需的行为。

例如，目标 NFS 目录在 NFS 服务器中，如下所示：

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

输**出示例**

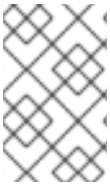
```
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

```
$ id nfsnobody
```

输**出示例**

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

为了可以访问目录，容器必须匹配 SELinux 标签，并使用 UID**65534**、**nfsnobody**的所有者，或其 supplemental 组的 **5555** 运行。



注意

所有者 ID **65534** 只是一个示例。虽然 NFS 的 **root_squash** 把 **root**，uid **0** 映射到 **nfsnobody**，uid **65534**，但 NFS 导出的所有者 ID 可能是任意值。NFS 导出的所有者不需要是 **65534** 。

4.9.3.1. 组 ID

用来控制 NFS 访问（假设不能在 NFS 导出中修改权限）的建议方法是使用附加组（supplemental group）。OpenShift Container Platform 中的附件组的功能是用于共享存储（NFS 是一个共享存储）。相对块存储（如 iSCSI），使用 **fsGroup** SCC 策略和在 Pod 的 **securityContext** 中的**fsGroup** 值。

注意

在访问持久性存储时，一般情况下最好使用 supplemental 组 ID 而不是使用用户 ID。

示例中目标 NFS 目录上的组 ID 是 **5555**， Pod 可以使用 Pod 的 **securityContext**定义中的 **supplementalGroups** 来设置组 ID。例如：

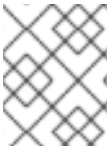
```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1 **securityContext** 必须在 pod 一级定义，而不是在某个特定容器中定义。

2 为 pod 定义的 GID 数组。在这种情况下，是阵列中的一个元素。使用逗号将不同 GID 分开。

假设没有可能满足 pod 要求的自定义 SCC，pod 可能与受限 SCC 匹配。这个 SCC 把 **supplementalGroups** 策略设置为 **RunAsAny**。这代表提供的任何组群 ID 都被接受，且不进行范围检查。

因此，上面的 pod 可以通过，并被启动。但是，如果需要进行组 ID 范围检查，使用自定义 SCC 就是首选的解决方案。可创建一个定义了最小和最大组群 ID 的自定义 SCC，这样就会强制进行组 ID 范围检查，组 ID **5555** 将被允许。



注意

要使用自定义 SCC，需要首先将其添加到适当的服务帐户（service account）中。例如，在一个特定项目中使用 **default** 服务账户（除非在 **Pod** 规格中指定了另外一个账户）。

4.9.3.2. 用户 ID

用户 ID 可以在容器镜像或者 **Pod** 定义中定义。

注意

通常情况下，最好使用附件组群 ID 而不是用户 ID 来获得对持久性存储的访问。

在上面显示的目标 NFS 目录示例中，容器需要将其 UID 设定为 **65534**，忽略组 ID。因此可以把以下内容添加到 **Pod** 定义中：

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

1 Pods 包括一个特定于每一个容器的 **securityContext** 定义，以及一个适用于 Pod 定义中所有容器的 pod 的 **securityContext**。

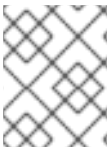
2 **65534** 是 **nfsnobody** 用户。

假设项目为 **default** 且 SCC 为 **restricted**，则不允许 pod 请求的用户 ID **65534**。因此，pod 会因以下原因失败：

- 它要求 **65534** 作为其用户 ID。

- Pod 可用的所有 SCC 被检查以决定哪些 SCC 允许 ID 为 **65534** 的用户。虽然检查了 SCC 的所有策略，但这里的焦点是用户 ID。
- 因为所有可用的 SCC 都使用 **MustRunAsRange** 作为其 **runAsUser** 策略，所以需要进行 UID 范围检查。
- **65534** 不包含在 SCC 或项目的用户 ID 范围内。

一般情况下，作为一个最佳实践方案，最好不要修改预定义的 SCC。解决这个问题的的首选方法是，创建一个自定义 SCC，在其中定义最小和最大用户 ID。 UID 范围仍然会被强制检查，UID **65534** 会被允许。



注意

要使用自定义 SCC，需要首先将其添加到适当的服务帐户（service account）中。例如，在一个特定项目中使用 **default** 服务账户（除非在 **Pod** 规格中指定了另外一个账户）。

4.9.3.3. SELinux

Red Hat Enterprise Linux（RHEL）和 Red Hat Enterprise Linux CoreOS（RHCOS）系统被配置为默认在远程 NFS 服务器中使用 SELinux。

对于非 RHEL 和非 RHCOS 系统，SELinux 不允许从 pod 写入远程 NFS 服务器。NFS 卷正确挂载，但只读。您将需要按照以下步骤启用正确的 SELinux 权限。

先决条件

- 必须安装 **container-selinux** 软件包。这个软件包提供 **virt_use_nfs** SELinux 布尔值。

流程

- 使用以下命令启用 **virt_use_nfs** 布尔值。使用 **-P** 选项可以使这个布尔值在系统重启后仍然有效。

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

4.9.3.4. 导出设置

为了使任意容器用户都可以读取和写入卷， NFS 服务器中的每个导出的卷都应该满足以下条件：

- 每个导出必须使用以下格式导出：

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

- 必须将防火墙配置为允许到挂载点的流量。
  - 对于 NFSv4，配置默认端口 **2049** （**nfs**）。

    NFSv4

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

  - 对于 NFSv3，需要配置三个端口： **2049** （**nfs**）、 **20048** （**mountd**）和 **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
```

- 必须设置 NFS 导出和目录，以便目标 pod 可以对其进行访问。将导出设定为由容器的主 UID 拥有，或使用 **supplementalGroups** 来允许 pod 组进行访问（如上面的与组 ID 相关的章节所示）。

4.9.4. 重新声明资源

NFS 实现了 OpenShift Container Platform **Recyclable** 插件接口。自动进程根据在每个持久性卷上设定的策略处理重新声明的任务。

默认情况下，PV 被设置为 **Retain**。

当一个 PVC 被删除后，PV 被释放，这个 PV 对象不能被重复使用。反之，应该创建一个新的 PV，其基本的卷详情与原始卷相同。

例如：管理员创建一个名为 **nfs1**的 PV：

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

用户创建 **PVC1**，它绑定到 **nfs1**。然后用户删除了 **PVC1**，对**nfs1**的声明会被释放。这将会使 **nfs1** 的状态变为 **Released**。如果管理员想要使这个 NFS 共享变为可用，则应该创建一个具有相同 NFS 服务器详情的新 PV，但使用一个不同的 PV 名称：

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

删除原来的 PV。不建议使用相同名称重新创建。尝试手工把一个 PV 的状态从 **Released** 改为 **Available** 会导致错误并可能造成数据丢失。

4.9.5. 其他配置和故障排除

根据所使用的 NFS 版本以及配置，可能还需要额外的配置步骤来进行正确的导出和安全映射。以下是一些可能适用的信息：

NFSv4 挂载错误地显示所有文件的所有者为 **nobody:nobody**	• 可归因于 NFS 中的 **/etc/idmapd.conf** 中的 ID 映射设置。 • 请参考红帽解决方案。
在 NFSv4 上禁用 ID 映射	• 在 NFS 客户端和服务器中运行： `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

4.10. RED HAT OPENSHIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation 是 OpenShift Container Platform 支持的文件、块和对象存储的持久性存储供应商，可以在内部或混合云环境中使用。作为红帽存储解决方案，Red Hat OpenShift Data Foundation 与 OpenShift Container Platform 完全集成，用于部署、管理和监控。如需更多信息，请参阅 Red Hat OpenShift Data Foundation 文档 。

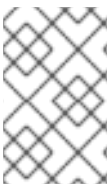
重要

虚拟化的 Red Hat Hyperconverged Infrastructure(RHHI)上的 OpenShift Data Foundation 不被支持。它使用超融合节点来托管 OpenShift Container Platform 安装的虚拟机。有关支持的平台的更多信息，请参阅 Red Hat OpenShift Data Foundation 支持性和互操作性指南。

4.11. 使用 VMWARE VSPHERE 卷的持久性存储

OpenShift Container Platform 允许使用 VMware vSphere 的虚拟机磁盘 (VMDK) 卷。您可以使用 VMware vSphere 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 VMware vSphere 已有一定了解。

VMware vSphere 卷可以动态置备。OpenShift Container Platform 在 vSphere 中创建磁盘，并将此磁盘附加到正确的镜像。

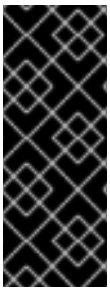


注意

OpenShift Container Platform 将新卷置备为独立持久性卷，它们可以在集群中的任何节点上自由附加和分离卷。因此，您无法备份使用快照的卷，或者从快照中恢复卷。如需更多信息，请参阅快照限制。

Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。

持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间的，相应的用户可请求它。



重要

对于新安装，OpenShift Container Platform 4.13 及更新的版本为 vSphere in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。更新至 OpenShift Container Platform 4.15 及更新的版本还提供自动迁移。有关更新和迁移的更多信息，请参阅 CSI 自动迁移。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。

其他资源

- VMware vSphere

4.11.1. 动态置备 VMware vSphere 卷

动态置备 VMware vSphere 卷是推荐的方法。

4.11.2. 先决条件

- 在一个满足您使用的组件要求的 VMware vSphere 版本上安装了 OpenShift Container Platform 集群。有关 vSphere 版本支持的信息，请参阅在 vSphere 上安装集群。

您可以通过以下任一流程使用默认的存储类动态置备这些卷。

4.11.2.1. 使用 UI 动态置备 VMware vSphere 卷

OpenShift Container Platform 安装了一个默认的存储类，其名为 **thin**，使用 **thin** 磁盘格式置备卷。

先决条件

- 当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程

1. 在 OpenShift Container Platform 控制台中，点击 **Storage → Persistent Volume Claims**。
2. 在持久性卷声明概述页中，点 **Create Persistent Volume Claim**。
3. 在接下来的页面中定义所需选项。
   a. 选择 **thin** 存储类。
   b. 输入存储声明的唯一名称。
   c. 选择访问模式来决定所创建存储声明的读写访问权限。
   d. 定义存储声明的大小。
4. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

4.11.2.2. 使用 CLI 动态置备 VMware vSphere 卷

OpenShift Container Platform 安装了一个默认的 StorageClass，其名为 **thin**，使用 **thin** 磁盘格式置备卷。

先决条件

- 当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程 (CLI)

1. 您可以通过创建一个包含以下内容的 **pvc.yaml** 文件来定义 VMware vSphere PersistentVolumeClaim：

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

1. 代表持久性卷声明的唯一名称。
2. 持久性卷声明的访问模式。使用 **ReadWriteOnce** 时，单个节点可以通过读写权限挂载该卷。
3. 持久性卷声明的大小。

2. 输入以下命令从文件创建 **PersistentVolumeClaim** 对象：

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

4.11.3. 静态置备 VMware vSphere 卷

要静态置备 VMware vSphere 卷，您必须创建虚拟机磁盘供持久性卷框架引用。

先决条件

- 当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。

流程

1. 创建虚拟机磁盘。在静态置备 VMware vSphere 卷前，必须手动创建虚拟机磁盘 (VMDK)。可使用以下任一方法：
   - 使用 **vmkfstools** 创建。通过 Secure Shel (SSH) 访问 ESX，然后使用以下命令创建 VMDK 卷：
     -

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

- 使用 **vmware-diskmanager** 创建：

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

2. 创建引用 VMDK 的持久性卷。创建包含 **PersistentVolume** 对象定义的 **pv1.yaml** 文件：

```
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) 卷的名称。持久性卷声明或 pod 识别它的名称。

(2) 为这个卷分配的存储量。

(3) 使用的卷类型，**vsphereVolume** 表示 vSphere 卷。此标签用于将 vSphere VMDK 卷挂载到 Pod 中。卸载卷时会保留卷内容。卷类型支持 VMFS 和 VSAN 数据存储。

(4) 要使用的现有 VMDK 卷。如果使用 **vmkfstools**，在卷定义中数据存储名称必须放在方括号 **[]** 内，如前面所示。

(5) 要挂载的文件系统类型。例如： ext4、xfs 或者其他文件系统。

重要

在格式化并置备卷后更改 fsType 参数的值可能会导致数据丢失和 pod 故障。

3. 从文件创建 **PersistentVolume** 对象：

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

4. 创建一个映射到您在上一步中创建的持久性卷声明。创建包含 **PersistentVolumeClaim** 对象定义的 **pvc1.yaml** 文件：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1 (1)
spec:
  accessModes:
    - ReadWriteOnce (2)
```

```
  resources:
    requests:
      storage: "1Gi" (3)
  volumeName: pv1 (4)
```

1. 代表持久性卷声明的唯一名称。
2. 持久性卷声明的访问模式。使用 ReadWriteOnce 时，单个节点可以通过读写权限挂载这个卷。
3. 持久性卷声明的大小。
4. 现有持久性卷的名称。

5. 从文件创建 **PersistentVolumeClaim** 对象：

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

4.11.3.1. 格式化 VMware vSphere 卷

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查卷是否包含由 **PersistentVolume**（PV）定义中 **fsType** 参数值指定的文件系统。如果没有使用文件系统格式化设备，该设备中的所有数据都会被清除，设备也会自动格式化为指定的文件系统。

因为 OpenShift Container Platform 在首次使用卷前会进行格式化，所以您可以使用未格式化的 vSphere 卷作为 PV。

4.12. 使用本地存储的持久性存储

4.12.1. 本地存储概述

您可以使用以下解决方案置备本地存储：

- HostPath Provisioner (HPP)
- Local Storage Operator (LSO)
- Logical Volume Manager (LVM) Storage

警告

这些解决方案只支持置备节点本地存储。工作负载绑定到提供存储的节点。如果节点不可用，工作负载也会变得不可用。要保持工作负载可用性（尽管节点失败），您必须确保通过主动或被动复制机制进行存储数据。

4.12.1.1. HostPath Provisioner 功能概述

您可以使用 HostPath Provisioner (HPP) 执行以下操作：

- 将主机文件系统路径映射到存储类以进行本地存储。
- 静态创建存储类，在节点上配置文件系统路径以进行存储消耗。
- 根据存储类静态置备持久性卷 (PV)。
- 在了解底层存储拓扑时，创建工作负载和 PersistentVolumeClaims (PVC)。

注意

HPP 在上游 Kubernetes 中提供。但是，不建议使用来自上游 Kubernetes 的 HPP。

4.12.1.2. Local Storage Operator 功能概述

您可以使用 Local Storage Operator (LSO) 执行以下操作：

- 在不修改设备配置的情况下，将存储设备（磁盘或分区）分配给存储类。
- 通过配置 **LocalVolume** 自定义资源 (CR) 以静态置备 PV 和存储类。
- 在了解底层存储拓扑时创建工作负载和 PVC。

注意

LSO 由红帽开发和交付。

4.12.1.3. LVM 存储功能概述

您可以使用逻辑卷管理器(LVM)存储执行以下操作：

- 将存储设备（磁盘或分区）配置为 lvm2 卷组，并将卷组公开为存储类。
- 在不考虑节点拓扑的情况下，使用 PVC 创建工作负载和请求存储。

LVM 存储使用 TopoLVM CSI 驱动程序，将存储空间动态分配给拓扑中的节点，并置备 PV。

注意

LVM 存储由红帽开发和维护。LVM 存储提供的 CSI 驱动程序是上游项目 "topolvm"。

4.12.1.4. LVM 存储、LSO 和 HPP 的比较

以下小节比较 LVM Storage、Local Storage Operator (LSO)和 HostPath Provisioner (HPP) 提供的功能，以置备本地存储。

4.12.1.4.1. 支持存储类型和文件系统的比较

下表比较了对 LVM Storage、Local Storage Operator (LSO)和 HostPath Provisioner (HPP)提供的存储类型和文件系统的支持，以置备本地存储：

表 4.1. 支持存储类型和文件系统的比较

功能	LVM 存储	LSO	HPP
支持块存储	是	是	否
支持文件存储	是	是	是
支持对象存储[1]	否	否	否
可用文件系统	**ext4**,**xfs**	**ext4**,**xfs**	支持节点上任何挂载的系统。

1. 任何解决方案(LVM 存储、LSO 和 HPP)都不支持对象存储。因此，如果要使用对象存储，则需要 S3 对象存储解决方案，如 Red Hat OpenShift Data Foundation 中的 **MultiClusterGateway**。所有解决方案都可以充当 S3 对象存储解决方案的底层存储供应商。

4.12.1.4.2. 对核心功能的支持比较

下表比较了 LVM Storage、Local Storage Operator (LSO)和 HostPath Provisioner (HPP)支持置备本地存储的核心功能：

表 4.2. 对核心功能的支持比较

功能	LVM 存储	LSO	HPP
支持自动文件系统格式	是	是	N/A
支持动态置备	是	否	否
支持使用软件独立磁盘阵列(RAID)阵列	是 支持 4.15 及更新的版本。	是	是
支持透明磁盘加密	是 在 4.16 及更新的版本中支持。	是	是
支持基于卷的磁盘加密	否	否	否
支持断开连接的安装	是	是	是
支持 PVC 扩展	是	否	否
支持卷快照和卷克隆	是	否	否

功能	LVM 存储	LSO	HPP
支持精简配置	是 默认情况下，设备是精简置备的。	是 您可以将设备配置为指向精简配置的卷	是 您可以配置指向精简配置的卷的路径。
支持自动磁盘发现和设置	是 在安装过程中和运行时提供自动磁盘发现。您还可以在 **LVMCluster** 自定义资源(CR)中动态添加磁盘，以增加现有存储类的存储容量。	技术预览 在安装过程中提供自动磁盘发现。	否

4.12.1.4.3. 性能和隔离功能的比较

下表比较了置备本地存储中的 LVM 存储、Local Storage Operator (LSO)和 HostPath Provisioner (HPP)的性能和隔离功能。

表 4.3. 性能和隔离功能的比较

功能	LVM 存储	LSO	HPP
性能	对使用相同存储类的所有工作负载，I/O 速度都是共享的。 块存储允许直接 I/O 操作。 精简配置可能会影响性能。	I/O 依赖于 LSO 配置。 块存储允许直接 I/O 操作。	对使用相同存储类的所有工作负载，I/O 速度都是共享的。 底层文件系统实施的限制可能会影响 I/O 速度。
隔离边界 [1]	LVM 逻辑卷(LV) 与 HPP 相比，它提供了更高级别的隔离。	LVM 逻辑卷(LV) 与 HPP 相比，它提供了更高级别的隔离	文件系统路径 与 LSO 和 LVM 存储相比，它提供了较低级别的隔离。

1. 隔离边界指的是使用本地存储资源的不同工作负载或应用程序之间的隔离程度。

4.12.1.4.4. 支持额外功能的比较

下表比较了 LVM Storage、Local Storage Operator (LSO)和 HostPath Provisioner (HPP)提供的额外功能，以置备本地存储：

表 4.4. 支持额外功能的比较

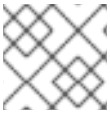
功能	LVM 存储	LSO	HPP
支持通用临时卷	是	否	否
支持 CSI 内联临时卷	否	否	否
支持存储拓扑	是 支持 CSI 节点拓扑	是 LSO 通过节点容限提供对存储拓扑的部分支持。	否
支持 **ReadWriteMany** (RWX) 访问模式 [1]	否	否	否

1. 所有解决方案 (LVM 存储、LSO 和 HPP) 都有 **ReadWriteOnce** (RWO) 访问模式。RWO 访问模式允许从同一节点上的多个 pod 访问。

4.12.2. 使用本地卷的持久性存储

OpenShift Container Platform 可以使用本地卷来置备持久性存储。本地持久性卷允许您使用标准持久性卷声明接口访问本地存储设备，如磁盘或分区。

无需手动将 pod 调度到节点即可使用本地卷，因为系统了解卷节点的约束。但是，本地卷仍会受到底层节点可用性的影响，而且并不适用于所有应用程序。



注意

本地卷只能用作静态创建的持久性卷。

4.12.2.1. 安装 Local Storage Operator

默认情况下，OpenShift Container Platform 中不会安装 Local Storage Operator。使用以下流程来安装和配置这个 Operator，从而在集群中启用本地卷。

先决条件

- 访问 OpenShift Container Platform web 控制台或命令行 (CLI)。

流程

1. 创建 **openshift-local-storage** 项目：

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

2. 可选：允许在基础架构节点上创建本地存储。
   您可能希望使用 Local Storage Operator 在基础架构节点上创建卷来支持一些组件，如日志记录和监控。

   您必须调整默认节点选择器，以便 Local Storage Operator 包含基础架构节点，而不只是 worker 节点。

要阻止 Local Storage Operator 继承集群范围的默认选择器，请输入以下命令：

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

3. 可选：允许在单节点部署中的 CPU 管理池中运行本地存储。
在单节点部署中使用 Local Storage Operator，并允许使用属于 **management** 池的 CPU。在使用管理工作负载分区的单节点安装上执行这个步骤。

   要允许 Local Storage Operator 在管理 CPU 池上运行，请运行以下命令：

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

使用 UI

按照以下步骤，通过 web 控制台安装 Local Storage Operator：

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 导航至 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Local Storage** 以查找 Local Storage Operator。
4. 点击 **Install**。
5. 在 **Install Operator** 页面中，选择 **A specific namespace on the cluster**。从下拉菜单中选择 **openshift-local-storage**。
6. 将 **Update Channel** 和 **Approval Strategy** 的值调整为所需的值。
7. 点击 **Install**。

完成后，Web 控制台的 **Installed Operators** 部分中会列出 Local Storage Operator。

使用 CLI

1. 通过 CLI 安装 Local Storage Operator。
   a. 创建对象 YAML 文件，以定义 Local Storage Operator 的 Operator 组和订阅，如 **openshift-local-storage.yaml**：

openshift-local-storage.yaml 示例

```
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) 安装计划的用户批准策略。

2. 输入以下命令来创建 Local Storage Operator 对象：

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

在此阶段，Operator Lifecycle Manager (OLM) 已可以了解 Local Storage Operator。Operator 的 ClusterServiceVersion (CSV) 应出现在目标命名空间中，由 Operator 提供的 API 应可用于创建。

3. 通过检查是否创建了所有 pod 和 Local Storage Operator 来验证本地存储安装：

a. 检查是否已创建所有必需的 pod:

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

输**出示例**

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

b. 检查 ClusterServiceVersion (CSV) YAML 清单，查看 **openshift-local-storage** 项目中是否有 Local Storage Operator:

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

输**出示例**

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

如果通过了所有检查，则代表 Local Storage Operator 已被成功安装。

4.12.2.2. 使用 Local Storage Operator 置备本地卷

无法通过动态置备来创建本地卷。相反，持久性卷可由 Local Storage Operator 创建。本地卷置备程序会在定义的资源中指定的路径上查找任意文件系统或块设备。

先决条件

- 安装了 Local Storage Operator。
- 您有一个满足以下条件的本地磁盘：

- 它附加到一个节点。
- 它尚未挂载。
- 它不包含分区。

流程

1. 创建本地卷资源。此资源必须定义本地卷的节点和路径。

注意

不要在同一设备中使用不同的存储类名称。这样做可创建多个持久性卷 (PV)。

例如：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)
      forceWipeDevicesAndDestroyAllData: false (4)
      volumeMode: Filesystem (5)
      fsType: xfs (6)
      devicePaths: (7)
        - /path/to/device (8)
```

(1) 安装了 Local Storage Operator 的命名空间。

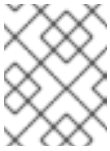
(2) 可选：包含附加了本地存储卷的节点列表的节点选择器。本例使用从 **oc get node** 获取的节点主机名。如果没有定义值，则 Local Storage Operator 会尝试在所有可用节点上查找匹配的磁盘。

(3) 创建持久性卷对象时使用的存储类的名称。如果不存在，Local Storage Operator 会自动创建存储类。确保使用唯一标识此本地卷的存储类。

(4) 此设置定义是否调用 **wipefs**，它会删除分区表签名 (magic strings)，使磁盘准备好用于 Local Storage Operator (LSO) 置备。除了签名外，没有其它数据会被清除。默认为 "false"（不调用 **wipefs** ）。当在需要重新使用的磁盘中，将 **forceWipeDevicesAndDestroyAllData** 设置为 "true" 很有用。在这些情况下，将此字段设置为 true 可消除管理员手动擦除磁盘的需要。此类情况可以包括单节点 OpenShift (SNO) 集群环境，其中节点可以多次重新部署，或使用 OpenShift Data Foundation (ODF)，其中

之前的数据可以保留在计划作为对象存储设备 (OSD) 消耗的磁盘上。

5 定义本地卷类型的卷模式，可以是 **Filesystem** 或 **Block**。



注意

原始块卷 (**volumeMode: Block**) 不会被格式化为文件系统。仅在 pod 上运行的任何应用程序都可以使用原始块设备时使用此模式。

6 第一次挂载本地卷时所创建的文件系统。

7 包含要从中选择的本地存储设备列表的路径。

8 使用到 **LocalVolume** 资源 **by-id** 的实际本地磁盘文件路径（如 **/dev/disk/by-id/wwn**）替换这个值。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。

注意

如果使用 RHEL KVM 运行 OpenShift Container Platform，则必须为虚拟机磁盘分配序列号。否则，重启后无法识别虚拟机磁盘。您可以使用 **virsh edit <VM>** 命令添加 **<serial>mydisk</serial>** 定义。

例如：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: "local-sc" 3
      forceWipeDevicesAndDestroyAllData: false 4
      volumeMode: Block 5
      devicePaths: 6
        - /path/to/device 7
```

1 安装了 Local Storage Operator 的命名空间。

2 可选：包含附加了本地存储卷的节点列表的节点选择器。本例使用从 **oc get node** 获取的节点主机名。如果没有定义值，则 Local Storage Operator 会尝试在所有可用节点上查找匹配的磁盘。

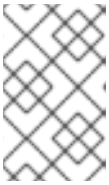
3 创建持久性卷对象时使用的存储类的名称。

4 此设置定义是否调用 **wipefs**，它会删除分区表签名 (magic strings)，使磁盘准备好用于 Local Storage Operator (LSO) 置备。除了签名外，没有其它数据会被清除。默认为 "false"（不调用 **wipefs** ）。当在需要重新使用的磁盘中，将 **forceWipeDevicesAndDestroyAllData** 设置为 "true" 很有用。在这些情况下，将此字段设置为 true 可消除管理员手动擦除磁盘的需要。此类情况可以包括单节点 OpenShift (SNO) 集群环境，其中节点可以多次重新部署，或使用 OpenShift Data Foundation (ODF)，其中之前的数据可以保留在计划作为对象存储设备 (OSD) 消耗的磁盘上。

5 定义本地卷类型的卷模式，可以是 **Filesystem** 或 **Block**。

6 包含要从中选择的本地存储设备列表的路径。

7 使用到 **LocalVolume** 资源 **by-id** 的实际本地磁盘文件路径（如 **dev/disk/by-id/wwn**）替换这个值。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。



注意

如果使用 RHEL KVM 运行 OpenShift Container Platform，则必须为虚拟机磁盘分配序列号。否则，重启后无法识别虚拟机磁盘。您可以使用 **virsh edit <VM>** 命令添加 **<serial>mydisk</serial>** 定义。

2. 在 OpenShift Container Platform 集群中创建本地卷资源。指定您刚才创建的文件：

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

3. 验证置备程序是否已创建并创建了相应的守护进程集：

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

输**出示例**

```
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
```

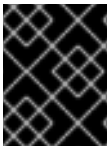
注意所需和当前的守护进程设定进程数。所需的数量为 **0** 表示标签选择器无效。

4. 验证持久性卷是否已创建：

```
$ oc get pv
```

输**出示例**

```
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
```



重要

编辑 **LocalVolume** 对象不会更改现有持久性卷的 **fsType** 或 **volumeMode**，因为这样做可能会导致破坏性操作。

4.12.2.3. 在没有 Local Storage Operator 的情况下置备本地卷

无法通过动态置备来创建本地卷。反之，可以通过在对象定义中定义持久性卷（PV）来创建持久性卷。本地卷置备程序会在定义的资源中指定的路径上查找任意文件系统或块设备。

重要

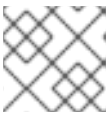
手动置备 PV 的风险包括在删除 PVC 时，在 PV 间可能会出现数据泄漏的问题。建议在置备本地 PV 时自动执行 Local Storage Operator。

先决条件

- 本地磁盘已附加到 OpenShift Container Platform 节点。

流程

1. 定义 PV。使用 **PersistentVolume** 对象定义创建一个文件，如 **example-pv-filesystem.yaml** 或 **example-pv-block.yaml**。此资源必须定义本地卷的节点和路径。



注意

不要在同一设备中使用不同的存储类名称。这将会创建多个 PV。

example-pv-filesystem.yaml

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

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

1. 定义 PV 类型的卷模式，可以是 **Filesystem** 或 **Block**。
2. 创建 PV 资源时使用的存储类的名称。使用唯一标识此 PV 的存储类。
3. 包含要从中选择的本地存储设备列表的路径，或一个目录。您只能指定 **Filesystem volumeMode** 的目录。

注意

原始块卷（**volumeMode: block**）不能以文件系统格式化。仅在 pod 上运行的任何应用程序都可以使用原始块设备时使用此模式。

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-sc (2)
  local:
    path: /dev/xvdf (3)
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
```

```
        operator: In
        values:
        - example-node
```

1. 定义 PV 类型的卷模式，可以是 **Filesystem** 或 **Block**。
2. 创建 PV 资源时使用的存储类的名称。确保使用唯一标识此 PV 的存储类。
3. 包含要从中选择的本地存储设备列表的路径。

2. 在 OpenShift Container Platform 集群中创建 PV 资源。指定您刚才创建的文件：

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

3. 验证是否已创建本地 PV：

```
$ oc get pv
```

输**出示例**

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

4.12.2.4. 创建本地卷持久性卷声明

必须静态创建本地卷作为持久性卷声明（PVC），才能被 pod 访问。

先决条件

- 持久性卷是使用本地卷置备程序创建的。

流程

1. 使用对应的存储类创建 PVC:

```
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. PVC 的名称。
2. PVC 的类型。默认为 **Filesystem**。
3. PVC 可用的存储量。
4. 声明所需的存储类的名称。

2. 通过指定您刚才创建的文件，在 OpenShift Container Platform 集群中创建 PVC：

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

4.12.2.5. 附加本地声明

本地卷映射到持久性卷声明后，可在资源内指定。

先决条件

- 同一命名空间中存在持久性卷声明。

流程

1. 在资源规格中包含定义的声明。以下示例在 pod 中声明持久性卷声明：

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

1. 要挂载的卷的名称。
2. 卷在 pod 中的挂载路径。不要挂载到容器 root、/ 或主机和容器中相同的任何路径。如果容器有足够权限，可能会损坏您的主机系统（如主机的 **/dev/pts** 文件）。使用 **/host** 挂载主机是安全的。
3. 要使用的现有持久性卷声明的名称。

2. 通过指定您刚才创建的文件，在 OpenShift Container Platform 集群中创建资源：

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

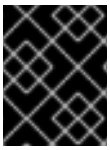
4.12.2.6. 为本地存储设备自动发现和置备

Local Storage Operator 自动进行本地存储发现和置备。使用此功能，您可以在部署过程中不提供动态置备（如使用裸机、VMware 或带有附加设备的 AWS 存储实例）时简化安装。

重要

自动发现和置备只是一个技术预览功能。技术预览功能不被红帽产品服务等级协议 (SLA) 支持，且可能在功能方面有缺陷。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。



重要

在使用内部部署 Red Hat OpenShift Data Foundation 或与平台无关的部署时，完全支持自动发现和置备。

使用以下步骤自动发现本地设备，并为所选设备自动置备本地。

警告

请小心使用 **LocalVolumeSet** 对象。当您从本地磁盘自动置备持久性卷(PV)时，本地 PV 可能会声明所有匹配的设备。如果使用 **LocalVolumeSet** 对象，请确保 Local Storage Operator 是管理该节点上本地设备的唯一实体。不支持针对一个节点创建多个 **LocalVolumeSet** 实例。

先决条件

- 有集群管理员权限。
- 已安装 Local Storage Operator。
- 已将本地磁盘附加到 OpenShift Container Platform 节点。
- 您可以访问 OpenShift Container Platform web 控制台和 **oc** 命令行界面（CLI）。

流程

1. 通过 web 控制台启用本地设备的自动发现：
   a. 点 **Operators → Installed Operators**。
   b. 在 **openshift-local-storage** 命名空间中，点 **Local Storage**。
   c. 点 **Local Volume Discovery** 选项卡。

d. 点 **Create Local Volume Discovery**，然后选择 **Form view** 或 **YAML view**。

e. 配置 **LocalVolumeDiscovery** 对象参数。

f. 点 **Create**。
Local Storage Operator 创建名为 **auto-discover-devices** 的本地卷发现实例。

2. 显示节点上持续可用的设备列表：

a. 登陆到 OpenShift Container Platform Web 控制台。

b. 进入 **Compute** → **Nodes**。

c. 点要打开的节点名称。此时会显示 "Node Details" 页面。

d. 选择 **Disks** 标签显示所选设备的列表。
在添加或删除本地磁盘时，设备列表会持续更新。您可以根据名称、状态、类型、型号、容量和模式过滤设备。

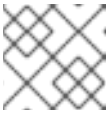
3. 从 web 控制台为发现的设备自动置备本地卷：

a. 导航到 **Operators** → **Installed Operators**，再从 Operators 列表中选择 **Local Storage**。

b. 选择 **Local Volume Set** → **Create Local Volume Set**。

c. 输入卷集合名称和存储类名称。

d. 选择 **All nodes** 或 **Select nodes** 以相应地应用过滤器。



注意

无论是使用 **All nodes** 或 **Select nodes** 进行过滤，只有 worker 节点可用。

e. 选择您要应用到本地卷集的磁盘类型、模式、大小和限制，然后点 **Create**。
几分钟后会显示一条信息，表示 "Operator reconciled successfullyd successfully."

4. 另外，也可通过 CLI 为发现的设备置备本地卷：

a. 创建一个对象 YAML 文件来定义本地卷集，如 **local-volume-set.yaml**，如下例所示：

```
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: local-sc (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 决定为从发现的设备置备的持久性卷创建的存储类。如果不存在，Local Storage Operator 会自动创建存储类。确保使用唯一标识此本地卷的存储类。

2 当使用本地卷设置功能时，Local Storage Operator 不支持使用逻辑卷管理（LVM）设备。

b. 创建本地卷集对象：

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

c. 根据存储类验证本地持久性卷是否被动态置备：

```
$ oc get pv
```

输出示例

```
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
```

注意

结果会在从节点中删除后删除。必须手动删除符号链接。

4.12.2.7. 使用 Local Storage Operator pod 的容限

污点可用于节点，以防止它们运行常规工作负载。要允许 Local Storage Operator 使用污点节点，您必须在 **Pod** 或 **DaemonSet** 定义中添加容限。这允许在这些污点节点上运行所创建的资源。

您可以通过 **LocalVolume** 资源把容限应用到 Local Storage Operator pod，通过节点规格把污点应用到一个节点。节点上的污点指示节点排斥所有不容许该污点的 pod。使用一个没有存在于其他 pod 上的特定污点可确保 Local Storage Operator pod 也可以在该节点上运行。

重要

污点与容限由 key、value 和 effect 组成。作为参数，它表示为 **key=value:effect**。运算符允许您将其中一个参数留空。

先决条件

- 安装了 Local Storage Operator。
- 本地磁盘已附加到带有一个污点的 OpenShift Container Platform 节点上。
- 污点节点可以置备本地存储。

流程

配置本地卷以便在污点节点上调度：

1. 修改定义 **Pod** 的 YAML 文件并添加 **LocalVolume** 规格,如下例所示：

```
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: "local-sc"
        volumeMode: Block 4
        devicePaths: 5
          - /dev/xvdg
```

1. 指定添加到节点的键。
2. 指定 **Equal** 运算符，以要求 **key/value** 参数匹配。如果运算符是 **Exists**，系统会检查键是否存在并忽略它的值。如果运算符是 **Equal**，则键和值必须匹配。
3. 指定污点节点的 **local** 值。
4. 定义本地卷类型的卷模式，可以是 **Filesystem** 或 **Block**。
5. 包含要从中选择的本地存储设备列表的路径。

2. 可选：要只在污点节点上创建本地持久性卷，修改 YAML 文件并添加 **LocalVolume** spec，如下例所示：

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

定义的容限度将传递给生成的守护进程集，允许为包含指定污点的节点创建 diskmaker 和 provisioner pod。

4.12.2.8. Local Storage Operator 指标

OpenShift Container Platform 为 Local Storage Operator 提供以下指标：

- **lso_discovery_disk_count**：每个节点中发现的设备总数
- **lso_lvset_provisioned_PV_count**: **LocalVolumeSet** 对象创建的 PV 总数
- **lso_lvset_unmatched_disk_count**: Local Storage Operator 没有选择进行置备的磁盘总数，因为不匹配条件
- **lso_lvset_orphaned_symlink_count**: 使用 PV 的设备数，它们不再与 **LocalVolumeSet** 对象标准匹配
- **lso_lv_orphaned_symlink_count** ：包含 PV 的设备数，它们不再符合 **LocalVolume** 对象标准
- **lso_lv_provisioned_PV_count**: **LocalVolume**置备的 PV 总数

要使用这些指标，请务必：

- 安装 Local Storage Operator 时启用对监控的支持。
- 当升级到 OpenShift Container Platform 4.9 或更高版本时，通过将 **operator-metering=true** 标签添加到命名空间来手动启用指标支持。

有关指标的更多信息，请参阅管理指标。

4.12.2.9. 删除 Local Storage Operator 资源

4.12.2.9.1. 删除本地卷或本地卷集

在一些情况下，必须删除本地卷和本地卷集。虽然删除资源中的条目并删除持久性卷通常就足够，但如果您想要重复使用同一设备路径或者使其不同的存储类进行管理，则需要额外的步骤。

注意

以下流程概述了删除本地卷的示例。同样的步骤也可以用于删除本地卷设置自定义资源的符号链接。

先决条件

- 持久性卷必须处于 **Released** 或 **Available** 状态。

警告

删除仍在使用中的持久性卷可能会导致数据丢失或崩溃。

流程

1. 编辑之前创建的本地卷以删除所有不需要的磁盘。
   a. 编辑集群资源：

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

   b. 找到 **devicePaths** 下的行，删除所有代表不需要的磁盘的行。
2. 删除所有创建的持久性卷。

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

3. 删除目录并包含节点上的符号链接。

警告

以下步骤涉及以 root 用户身份访问节点。如果在本流程中步骤范围以外修改节点状态，则可能会导致集群不稳定。

```
$ oc debug node/<node-name> -- chroot /host rm -rf /mnt/local-storage/<sc-name>
```
(1)

(1) 用于创建本地卷的存储类的名称。

4.12.2.9.2. 卸载 Local Storage Operator

要卸载 Local Storage Operator，您必须删除 Operator 以及 **openshift-local-storage** 项目中创建的所有资源。

警告

当本地存储 PV 仍在使用时，不建议卸载 Local Storage Operator。当 Operator 被移除后 PV 仍然会被保留。但是如果在没有删除 PV 和本地存储资源的情况下重新安装 Operator，则可能会出现不确定的行为。

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

1. 删除项目中安装的任何本地卷资源，如 **localvolume**、**localvolumeset** 和 **localvolumediscovery**：

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

2. 从 Web 控制台卸载 Local Storage Operator。
   a. 登陆到 OpenShift Container Platform Web 控制台。
   b. 导航到 **Operators → Installed Operators**。
   c. 在过滤器框中键入 **Local Storage** 以查找 Local Storage Operator。
   d. 点击 Local Storage Operator 末尾的 Options 菜单  。
   e. 点击 **Uninstall Operator**。
   f. 在出现的窗口中点击 **Remove**。
3. 由 Local Storage Operator 创建的 PV 将保留在集群中，直到被删除为止。这些卷不再使用后，运行以下命令删除它们：

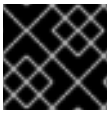
```
$ oc delete pv <pv-name>
```

4. 删除 **openshift-local-storage** 项目：

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

4.12.3. 使用 hostPath 的持久性存储

OpenShift Container Platform 集群中的 hostPath 卷将主机节点的文件系统中的文件或目录挂载到 pod 中。大多数 pod 都不需要 hostPath 卷，但是如果应用程序需要它，它会提供一个快速的测试选项。



重要

集群管理员必须将 pod 配置为以特权方式运行。这样可访问同一节点上的 pod。

4.12.3.1. 概述

OpenShift Container Platform 支持在单节点集群中使用 hostPath 挂载用于开发和测试目的。

在用于生产环境的集群中，不要使用 hostPath。集群管理员会置备网络资源，如 GCE Persistent Disk 卷、NFS 共享或 Amazon EBS 卷。网络资源支持使用存储类设置动态置备。

hostPath 卷必须静态置备 。

重要

不要挂载到容器 root、/ 或主机和容器中相同的任何路径。如果容器有足够权限，可能会损坏您的主机系统。使用 **/host** 挂载主机是安全的。以下示例显示主机中的 / 目录被挂载到位于 **/host** 的容器中。

```
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: ''
```

4.12.3.2. 静态置备 hostPath 卷

使用 hostPath 卷的 pod 必须通过手动（静态）置备来引用。

流程

1. 定义持久性卷（PV）的名称。创建包含 **PersistentVolume** 对象定义的 **pv.yaml** 文件：

```
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. 卷的名称。持久性卷声明或 pod 识别它的名称。
2. 用于将持久性卷声明请求绑定到这个持久性卷。
3. 这个卷可以被一个单一的节点以 **read-write** 的形式挂载。

4 配置文件指定卷在集群节点的 **/mnt/data** 中。不要挂载到容器 root、/ 或主机和容器中相同的任何路径。这可能会导致您的主机系统损坏。使用 **/host** 挂载主机是安全的。

2. 从该文件创建 PV：

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

3. 定义持久性卷声明（PVC）。创建包含 **PersistentVolumeClaim** 对象定义的 **pvc.yaml** 文件：

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

4. 从文件创建 PVC：

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

4.12.3.3. 在特权 pod 中挂载 hostPath 共享

创建持久性卷声明后，应用程序就可以使用它。以下示例演示了在 pod 中挂载此共享。

先决条件

- 已存在一个映射到底层 hostPath 共享的持久性卷声明。

流程

- 创建可挂载现有持久性卷声明的特权 pod:

```
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. pod 的名称。
2. pod 必须以特权运行，才能访问节点的存储。
3. 在特权 pod 中挂载主机路径共享的路径。不要挂载到容器 root、/ 或主机和容器中相同的任何路径。如果容器有足够权限，可能会损坏您的主机系统（如主机的 **/dev/pts** 文件）。使用 **/host** 挂载主机是安全的。
4. 之前创建的 **PersistentVolumeClaim** 对象的名称。

4.12.4. 使用逻辑卷管理器存储的持久性存储

逻辑卷管理器存储使用 TopoLVM CSI 驱动程序在 OpenShift Container Platform 集群中动态置备本地存储。

LVM Storage 使用逻辑卷管理器创建精简配置的卷，并在有限资源的集群中提供块存储的动态置备。

您可以使用 LVM 存储创建卷组、持久性卷声明(PVC)、卷快照和卷克隆。

4.12.4.1. 逻辑卷管理器存储安装

您可以在 OpenShift Container Platform 集群上安装逻辑卷管理器 (LVM) 存储，并将其配置为为您的工作负载动态置备存储。

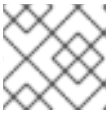
您可以使用 OpenShift Container Platform CLI (**oc**)、OpenShift Container Platform Web 控制台或 Red Hat Advanced Cluster Management (RHACM) 安装 LVM 存储。

> **警告**
>
> 当在多节点集群中使用 LVM 存储时，LVM 存储只支持置备本地存储。LVM 存储不支持跨节点的存储数据复制机制。您必须通过主动或被动复制机制来确保存储数据复制，以避免出现单点故障。

4.12.4.1.1. 安装 LVM 存储的先决条件

安装 LVM 存储的先决条件如下：

- 确保至少有 10 milliCPU 和 100 MiB RAM。
- 确保每个受管集群都有用于置备存储的专用磁盘。LVM 存储只使用那些为空且不包含文件系统签名的磁盘。为确保磁盘为空，且不包含文件系统签名，请在使用磁盘前擦除磁盘。
- 在私有 CI 环境中安装 LVM 存储前，您可以重复使用您在之前的 LVM 存储安装中配置的存储设备，请确保您已擦除未使用的磁盘。如果您在安装 LVM 存储前没有擦除磁盘，则无法重复使用磁盘，而无需人工干预。



注意

您不能擦除正在使用的磁盘。

- 如果要使用 Red Hat Advanced Cluster Management (RHACM) 安装 LVM 存储，请确保已在 OpenShift Container Platform 集群上安装 RHACM。请参阅"使用 RHACM 安装 LVM 存储"部分。

其他资源

- Red Hat Advanced Cluster Management for Kubernetes：在线安装

4.12.4.1.2. 使用 CLI 安装 LVM 存储

作为集群管理员，您可以使用 OpenShift CLI 安装 LVM 存储。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已以具有 **cluster-admin** 和 Operator 安装权限的用户身份登录 OpenShift Container Platform。

流程

1. 使用用于创建命名空间的配置创建 YAML 文件：

   创建命名空间的 YAML 配置示例

   ```
   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
   ```

2. 运行以下命令创建命名空间：

   ```
   $ oc create -f <file_name>
   ```

3. 创建 **OperatorGroup** CR YAML 文件：

   OperatorGroup CR 示例

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

4. 运行以下命令来创建 **OperatorGroup** CR:

```
$ oc create -f <file_name>
```

5. 创建 **Subscription** CR YAML 文件：

Subscription CR 示例

```
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
```

6. 运行以下命令来创建 **Subscription** CR：

```
$ oc create -f <file_name>
```

验证

1. 要验证 LVM 存储是否已安装，请运行以下命令：

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

输**出示例**

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

4.12.4.1.3. 使用 Web 控制台安装 LVM 存储

您可以使用 OpenShift Container Platform Web 控制台安装 LVM Storage。

先决条件

- 您可以访问集群。
- 您可以使用 **cluster-admin** 和 Operator 安装权限访问 OpenShift Container Platform。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → OperatorHub**。
3. 点 **OperatorHub** 页面中的 **LVM Storage**。

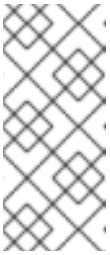
4. 在 Operator 安装页面中设置以下选项 ：

   a. 更新频道为 **stable-4.15**。

   b. **Installation Mode** 为 **A specific namespace on the cluster**。

   c. **Installed Namespace** 为 **Operator recommended namespace openshift-storage**。如果 **openshift-storage** 命名空间不存在，它会在 Operator 安装过程中创建。

   d. 将 **Update approval** 设置为 **Automatic** 或 **Manual**。



注意

如果选择 **Automatic** 更新，Operator Lifecycle Manager (OLM) 将自动更新 LVM Storage 的运行实例，而无需任何干预。

如果选择 手动 更新，则 OLM 会创建一个更新请求。作为集群管理员，您必须手动批准更新请求，以便将 LVM 存储更新至更新的版本。

5. 可选：选择 **Enable Operator recommended cluster monitoring on this Namespace**复选框。

6. 点 **Install**。

验证步骤

- 验证 LVM 存储是否显示绿色勾号，代表安装成功。

4.12.4.1.4. 在断开连接的环境中安装 LVM 存储

您可以在断开连接的环境中的 OpenShift Container Platform 上安装 LVM 存储。此流程中引用的所有部分都链接到 "Additional resources" 部分。

先决条件

- 您可以阅读"关于断开连接的安装镜像"部分。
- 您可以访问 OpenShift Container Platform 镜像存储库。
- 您创建了镜像 registry。

流程

1. 按照"创建镜像设置配置"中的步骤操作。要为 LVM Storage 创建 **ImageSetConfiguration** 自定义资源 (CR)，您可以使用以下示例 **ImageSetConfiguration** CR 配置：

LVM 存储的 ImageSetConfiguration CR 示例

```
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-4.15 4
      type: ocp
    graph: true 5
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.15 6
    packages:
    - name: lvms-operator 7
      channels:
      - name: stable 8
  additionalImages:
  - name: registry.redhat.io/ubi9/ubi:latest 9
  helm: {}
```

1. 设置镜像集合中每个文件的最大大小（以 GiB 为单位）。
2. 指定要保存镜像集的位置。此位置可以是 registry 或本地目录。除非使用技术预览 OCI 功能，否则您必须配置 **storageConfig** 字段。
3. 在使用 registry 时指定镜像流的存储 URL。如需更多信息，请参阅 *为什么使用镜像流*。
4. 指定您要从中检索 OpenShift Container Platform 镜像的频道。
5. 将此字段设置为 **true** 来生成 OpenShift Update Service (OSUS) 图形镜像。如需更多信息，请参阅 *关于 OpenShift Update Service*。
6. 指定您要从中检索 OpenShift Container Platform 镜像的 Operator 目录。
7. 指定要包含在镜像集中的 Operator 软件包。如果此字段为空，则检索目录中的所有软件包。
8. 指定要包含在镜像集中的 Operator 软件包的频道。即使不使用该频道中的捆绑包，还必须包含 Operator 软件包的默认频道。您可以运行以下命令来找到默认频道：**$ oc mirror list operators --catalog=<catalog_name> --package=<package_name>**.
9. 指定要在镜像集中包含的任何其他镜像。

2. 按照"镜像 registry"部分中的步骤操作。
3. 按照"配置镜像 registry 存储库镜像"部分中的步骤操作。

其他资源

- 关于断开连接的安装镜像
- 为 Red Hat OpenShift 创建带有 mirror registry 的 mirror registry
- 镜像 OpenShift Container Platform 镜像存储库
- 创建镜像设置配置
- 将镜像集镜像(mirror)到镜像 registry

- 配置镜像 registry 存储库镜像
- 为什么使用镜像流

4.12.4.1.5. 使用 RHACM 安装 LVM 存储

要使用 Red Hat Advanced Cluster Management (RHACM) 在集群中安装 LVM 存储，您必须创建一个 **Policy** 自定义资源 (CR)。您还可以配置条件来选择您要在其上安装 LVM 存储的集群。

注意

为安装 LVM 存储而创建的 **Policy** CR 也应用于在创建 **Policy** CR 后导入或创建的集群。

先决条件

- 您可以使用具有 **cluster-admin** 和 Operator 安装权限的账户访问 RHACM 集群。
- 您有专用磁盘，LVM 存储可在每个集群上使用。
- 集群必须由 RHACM 管理。

流程

1. 使用 OpenShift Container Platform 凭证登录到 RHACM CLI。
2. 创建命名空间。

```
$ oc create ns <namespace>
```

3. 创建 **Policy** CR YAML 文件：

用于安装和配置 LVM 存储的 Policy CR 示例

```
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: 2
            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: 3
            apiVersion: operators.coreos.com/v1
            kind: OperatorGroup
            metadata:
              name: openshift-storage-operatorgroup
              namespace: openshift-storage
            spec:
              targetNamespaces:
              - openshift-storage
        - complianceType: musthave
          objectDefinition: 4
            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
```

1. 设置 **PlacementRule.spec.clusterSelector** 中的 **key** 字段和 **values** 字段，以匹配您要在其上安装 LVM 存储的集群中配置的标签。
2. 命名空间配置。
3. **OperatorGroup** CR 配置。
4. **Subscription** CR 配置。

4. 运行以下命令来创建 **Policy** CR：

```
$ oc create -f <file_name> -n <namespace>
```

创建 **Policy** CR 后，会在与 **PlacementRule** CR 中配置的选择条件匹配的集群中创建以下自定义资源：

- **Namespace**
- **OperatorGroup**
- 订阅

- Red Hat Advanced Cluster Management for Kubernetes：在线安装
- 关于 LVMCluster 自定义资源

4.12.4.2. 配置在 LVM 存储中使用的设备大小的限制

使用 LVM 存储配置可用于置备存储的设备大小的限制如下：

- 您可以置备的总存储大小受底层逻辑卷管理器(LVM)精简池的大小以及过度置备因素的限制。
- 逻辑卷的大小取决于物理扩展(PE)和逻辑扩展(LE)的大小。
  - 您可以在创建物理和虚拟设备的过程中定义 PE 和 LE 的大小。
  - 默认的 PE 和 LE 大小为 4 MB。
  - 如果增加 PE 的大小，LVM 的最大大小由内核限值和您的磁盘空间决定。

表 4.5. 使用默认 **PE** 和 **LE** 大小的不同架构的大小限制

架构	RHEL 6	RHEL 7	RHEL 8	RHEL 9
32 位	16 TB	-	-	-

架构	RHEL 6	RHEL 7	RHEL 8	RHEL 9
64 位	8 EB [1] 100 TB [2]	8 EB [1] 500 TB [2]	8 EB	8 EB

1. 理论大小。
2. 测试大小。

4.12.4.3. 关于 LVMCluster 自定义资源

您可以配置 **LVMCluster** CR 以执行以下操作：

- 创建可用于置备持久性卷声明 (PVC) 的 LVM 卷组。
- 配置您要添加到 LVM 卷组的设备列表。
- 配置要求以选择要在其上创建 LVM 卷组的节点，以及卷组的精简池配置。
- 强制擦除所选设备。

安装 LVM 存储后，您必须创建一个 **LVMCluster** 自定义资源 (CR)。

LVMCluster CR YAML 文件示例

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
  namespace: openshift-storage
spec:
  tolerations:
  - effect: NoSchedule
    key: xyz
    operator: Equal
    value: "true"
  storage:
    deviceClasses:
    - name: vg1
      fstype: ext4 (1)
      default: true
      nodeSelector: (2)
        nodeSelectorTerms:
        - matchExpressions:
          - key: mykey
            operator: In
            values:
            - ssd
      deviceSelector: (3)
        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
    forceWipeDevicesAndDestroyAllData: true
  thinPoolConfig:
    name: thin-pool-1
    sizePercent: 90 4
    overprovisionRatio: 10
```

1 2 3 4 可选字段

LVMCluster CR 中的字段解释
LVMCluster CR 字段在下表中描述：

表 4.6. **LVMCluster** CR 字段

字段	类型	描述
spec.storage.deviceClasses	数组	包含将本地存储设备分配给 LVM 卷组的配置。 LVM Storage 为您创建的每个设备类创建一个存储类和卷快照类。 创建 **LVMCluster** CR 后，如果您添加或删除设备类，则更新仅在删除和重新创建 **topolvm-node** pod 后在 **LVMCluster** CR 中反映。
deviceClasses.name	**string**	为 LVM 卷组 (VG) 指定一个名称。 您还可以将此字段配置为重复使用您在之前安装中创建的卷组。如需更多信息，请参阅"使用之前 LVM 存储安装中的卷组"。
deviceClasses.fstype	**string**	将此字段设置为 **ext4** 或 **xfs**。默认情况下，此字段设置为 **xfs**。
deviceClasses.default	布尔值	将此字段设置为 **true** 以表示设备类是默认值。否则，您可以将其设置为 **false**。您只能配置单个默认存储类。
deviceClasses.nodeSelector	**object**	包含要在其上创建 LVM 卷组的节点的配置。如果此字段为空，则会考虑所有没有调度污点的节点。 在 control-plane 节点上，当新节点在集群中变为活跃时，LVM Storage 会检测到新节点并使用额外的 worker 节点。
nodeSelector.nodeSelectorTerms	数组	配置用于选择节点的要求。

字段	类型	描述
deviceClasses.deviceSelector	**object**	包含执行以下操作的配置： • 指定您要添加到 LVM 卷组的设备的路径。 • 强制擦除添加到 LVM 卷组的设备。 如需更多信息，请参阅"将设备添加到卷组"。
deviceSelector.paths	数组	指定设备路径。 如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，则 **LVMCluster** CR 会进入 **Failed** 状态。
deviceSelector.optionalPaths	数组	指定可选设备路径。 如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，LVM Storage 会忽略该设备而不造成错误。
deviceSelector.forceWipeDevicesAndDestroyAllData	布尔值	LVM 存储只使用那些为空且不包含文件系统签名的磁盘。为确保磁盘为空，且不包含文件系统签名，请在使用磁盘前擦除磁盘。 要强制擦除所选设备，请将此字段设置为 **true**。默认情况下，此字段设置为 **false**。 **警告** 如果此字段设为 **true**，则 LVM 存储会擦除设备上所有之前的数据。请谨慎使用此功能。 如果满足以下条件，则设备可能会导致数据完整性不一致： • 该设备被用作交换空间。 • 该设备是 RAID 阵列的一部分。 • 该设备已被挂载。 如果这些条件为 true，请不要强制擦除磁盘。相反，您必须手动擦除磁盘。
deviceClasses.thinPoolConfig	**object**	包含在 LVM 卷组中创建精简池的配置。
thinPoolConfig.name	**string**	为精简池指定一个名称。

字段	类型	描述
thinPoolConfig.sizePercent	整数	指定 LVM 卷组中空间的百分比，用于创建精简池。 默认情况下，此字段设置为 90。您可以设置的最小值为 10，最大值为 90。
thinPoolConfig.overprovisionRatio	整数	指定您可以根据精简池中可用存储置备额外存储的因素。 例如，如果此字段设置为 10，您可以在精简池中置备可用存储量最多 10 倍。 要禁用过度置备，请将此字段设置为 1。

其他资源

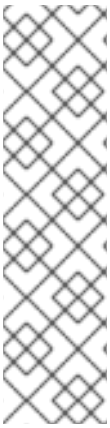
- 使用之前 LVM 存储安装中的卷组
- 关于在卷组中添加设备
- 将 worker 节点添加到单节点 OpenShift 集群

4.12.4.3.1. 关于在卷组中添加设备

LVMCluster CR 中的 **deviceSelector** 字段包含指定您要添加到 LVM 卷组的设备的路径。

您可以在 **deviceSelector.paths** 字段，或 **deviceSelector.optionalPaths** 字段，或这两个字段中指定设备路径。如果您没有在 **deviceSelector.paths** 字段和 **deviceSelector.optionalPaths** 字段中指定设备路径，LVM Storage 会将支持的未使用的设备添加到卷组 (VG)。

您可以将路径添加到 **deviceSelector** 字段中的 RAID 阵列，以将 RAID 阵列与 LVM 存储集成。您可以使用 **mdadm** 工具创建 RAID 阵列。LVM 存储不支持创建软件 RAID。



注意

您只能在 OpenShift Container Platform 安装过程中创建 RAID 阵列。有关创建 RAID 阵列的详情，请查看以下部分：

- "添加资源"中配置启用了 RAID 的数据卷。
- 在安装的系统中创建软件 RAID
- 替换 RAID 中失败的磁盘
- 修复 RAID 磁盘

如果您没有在 **LVMCluster** CR 中添加 **deviceSelector** 字段，LVM Storage 会在设备可用时自动添加新设备。

只有在满足以下条件时，LVM 存储才会将设备添加到 LVM 卷组中：

- 设备路径存在。
- LVM 存储支持该设备。

您还可以添加 RAID 阵列的路径，以将 RAID 阵列与 LVM 存储集成。如需更多信息，请参阅"添加资源"部分中的"将 RAID 阵列与 LVM 存储集成"。

重要

将设备添加到 LVM 卷组后，无法删除它。

其他资源

- 配置启用了 RAID 的数据卷
- LVM 存储不支持的设备

4.12.4.3.2. LVM 存储不支持的设备

当在 **LVMCluster** 自定义资源(CR)的 **deviceSelector** 字段中添加设备路径时，请确保 LVM Storage 支持设备。如果您在不支持的设备中添加路径，则 LVM 存储会排除设备以避免管理逻辑卷的复杂性。

如果您没有在 **deviceSelector** 字段中指定任何设备路径，则 LVM Storage 只添加它支持的未使用设备。

注意

要获取有关设备的信息，请运行以下命令：

```
$ lsblk --paths --json -o \
NAME,ROTA,TYPE,SIZE,MODEL,VENDOR,RO,STATE,KNAME,SERIAL,PARTLABEL
,FSTYPE
```

LVM 存储不支持以下设备：

只读设备
: 将 **ro** 参数设置为 **true** 的设备。

暂停设备
: 将 **state** 参数设置为 **suspended** 的设备。

ROM 设备
: 将 **type** 参数设置为 **rom** 的设备。

LVM 分区设备
: 将 **type** 参数设置为 **lvm** 的设备。

具有无效分区标签的设备
: 将 **partlabel** 参数设置为 **bios**、**boot** 或 **reserved** 的设备。

带无效文件系统的设备
: 将 **fstype** 参数设置为 **null** 或 **LVM2_member** 以外的任何值的设备。

重要

只有在设备不包含子设备时，LVM Storage 才支持将 **fstype** 参数设置为 **LVM2_member** 的设备。

属于另一个卷组的设备

要获取该设备的卷组信息，请运行以下命令：

```
$ pvs <device-name> 1
```

1 将 **<device-name>** 替换为设备名称。

带有绑定挂载的设备

要获取设备的挂载点，请运行以下命令：

```
$ cat /proc/1/mountinfo | grep <device-name> 1
```

1 将 **<device-name>** 替换为设备名称。

包含子设备的设备

注意

建议您在 LVM 存储中使用该设备前擦除该设备以防止意外行为。

4.12.4.4. 创建 LVMCluster 自定义资源的方法

您可以使用 OpenShift CLI (**oc**)或 OpenShift Container Platform Web 控制台创建 **LVMCluster** 自定义资源 (CR)。如果您使用 Red Hat Advanced Cluster Management (RHACM) 安装 LVM 存储，您还可以使用 RHACM 创建 **LVMCluster** CR。

创建 **LVMCluster** CR 后，LVM 存储会创建以下系统管理的 CR：

- 每个设备类的 **storageClass** 和 **volumeSnapshotClass**。

注意

LVM Storage 配置存储类和卷快照类的名称，格式为 **lvms-<device_class_name>**，其中 **<device_class_name>** 是 **LVMCluster** CR 中的 **deviceClasses.name** 字段的值。例如，如果 **deviceClasses.name** 字段设置为 vg1，则存储类的名称和卷快照类为 **lvms-vg1**。

- **LVMVolumeGroup** ：此 CR 是由 LVM 卷组支持的特定类型的持久性卷 (PV)。它在多个节点中跟踪各个卷组。
- **LVMVolumeGroupNodeStatus** ：此 CR 跟踪节点上卷组的状态。

4.12.4.4.1. 使用之前 LVM 存储安装中的卷组

您可以重复使用之前 LVM 存储安装中的现有卷组(VG)，而不是创建新的 VG。

您只能重复使用 VG，但不能重复利用与 VG 关联的逻辑卷。

重要

您只能在创建 **LVMCluster** 自定义资源 (CR) 时执行这个步骤。

先决条件

- 要重复使用的 VG 不得被损坏。
- 要重复使用的 VG 必须具有 **lvms** 标签。有关向 LVM 对象添加标签的更多信息，请参阅使用标签对 LVM 对象进行分组。

流程

1. 打开 **LVMCluster** CR YAML 文件。
2. 配置 **LVMCluster** CR 参数，如下例所示：

LVMCluster CR YAML 文件示例

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
  namespace: openshift-storage
spec:
# ...
  storage:
    deviceClasses:
    - name: vg1 (1)
      fstype: ext4 (2)
      default: true
      deviceSelector: (3)
# ...
        forceWipeDevicesAndDestroyAllData: false (4)
      thinPoolConfig: (5)
# ...
      nodeSelector: (6)
# ...
```

(1) 将此字段设置为之前 LVM 存储安装的 VG 的名称。

(2) 将此字段设置为 **ext4** 或 **xfs**。默认情况下，此字段设置为 **xfs**。

(3) 您可以通过在 **deviceSelector** 字段中指定新设备路径，将新设备添加到要重复使用的 VG。如果您不想在 VG 中添加新设备，请确保当前 LVM 存储安装中的 **deviceSelector** 配置与之前的 LVM 存储安装相同。

(4) 如果此字段设为 **true**，则 LVM 存储会擦除添加到 VG 的设备中的所有数据。

(5) 要保留您要重复使用的 VG 的 **thinPoolConfig** 配置，请确保当前 LVM 存储安装中的 **thinPoolConfig** 配置与之前的 LVM 存储安装相同。否则，您可以根据需要配置 **thinPoolConfig** 字段。

(6) 配置要求以选择要在其上创建 LVM 卷组的节点。如果此字段为空，则会考虑所有没有调度污点的节点。

3. 保存 **LVMCluster** CR YAML 文件。

注意

要查看属于卷组的设备，请运行以下命令：

```
$ pvs -S vgname=<vg_name> 1
```

1 将 **<vg_name>** 替换为卷组的名称。

4.12.4.4.2. 使用 CLI 创建 LVMCluster CR

您可以使用 OpenShift CLI (**oc**)在 worker 节点上创建 **LVMCluster** 自定义资源 (CR)。

重要

您只能在 OpenShift Container Platform 集群中创建 **LVMCluster** 自定义资源 (CR) 的单一实例。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 已以具有 **cluster-admin** 权限的用户身份登录 OpenShift Container Platform。
- 已安装 LVM 存储。
- 已在集群中安装了 worker 节点。
- 您可以阅读 "About the LVMCluster 自定义资源" 部分。

流程

1. 创建 **LVMCluster** 自定义资源(CR) YAML 文件：

LVMCluster CR YAML 文件示例

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
  namespace: openshift-storage
spec:
# ...
  storage:
    deviceClasses: 1
# ...
      nodeSelector: 2
# ...
      deviceSelector: 3
# ...
      thinPoolConfig: 4
# ...
```

1 包含将本地存储设备分配给 LVM 卷组的配置。

2 包含要在其上创建 LVM 卷组的节点的配置。如果此字段为空，则会考虑所有没有调度污点的节点。

3 包含指定您要添加到 LVM 卷组的设备的路径，并强制擦除添加到 LVM 卷组中的设备。

4 包含在 LVM 卷组中创建精简池的配置。

2. 运行以下命令来创建 **LVMCluster** CR：

```
$ oc create -f <file_name>
```

输**出示例**

```
lvmcluster/lvmcluster created
```

验证

1. 检查 **LVMCluster** CR 是否处于 **Ready** 状态：

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

输**出示例**

```
{"deviceClassStatuses": 1
[
  {
    "name": "vg1",
    "nodeStatus": [ 2
        {
            "devices": [ 3
                "/dev/nvme0n1",
                "/dev/nvme1n1",
                "/dev/nvme2n1"
            ],
            "node": "kube-node", 4
            "status": "Ready" 5
        }
    ]
  }
]
"state":"Ready"} 6
```

1 设备类的状态。

2 每个节点上的 LVM 卷组状态。

3 用于创建 LVM 卷组的设备列表。

4 创建设备类的节点。

5 节点上的 LVM 卷组状态。

6 **LVMCluster** CR 的状态。

注意

如果 **LVMCluster** CR 处于 **Failed** 状态，您可以在 **status** 字段中查看失败的原因。

带有 failue 原因的 **status** 字段示例：

```
status:
  deviceClassStatuses:
    - name: vg1
      nodeStatus:
        - node: my-node-1.example.com
          reason: no available devices found for volume group
          status: Failed
  state: Failed
```

2. 可选：要查看 LVM Storage 为每个设备类创建的存储类，请运行以下命令：

```
$ oc get storageclass
```

输出示例

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

3. 可选：要查看 LVM Storage 为每个设备类创建的卷快照类，请运行以下命令：

```
$ oc get volumesnapshotclass
```

输出示例

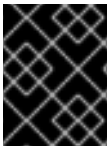
```
NAME         DRIVER           DELETIONPOLICY   AGE
lvms-vg1     topolvm.io       Delete           24h
```

其他资源

- 关于 LVMCluster 自定义资源

4.12.4.4.3. 使用 Web 控制台创建 LVMCluster CR

您可以使用 OpenShift Container Platform Web 控制台在 worker 节点上创建 **LVMCluster** CR。



重要

您只能在 OpenShift Container Platform 集群中创建 **LVMCluster** 自定义资源 (CR) 的单一实例。

先决条件

- 您可以使用 **cluster-admin** 权限访问 OpenShift Container Platform 集群。
- 已安装 LVM 存储。
- 已在集群中安装了 worker 节点。
- 您可以阅读 "About the LVMCluster 自定义资源" 部分。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators**。
3. 在 **openshift-storage** 命名空间中，点 **LVM Storage**。
4. 点 **Create LVMCluster** 并选择 **Form view** 或 **YAML view**。
5. 配置所需的 **LVMCluster** CR 参数。
6. 点 **Create**。
7. 可选： 如果要编辑 **LVMCLuster** CR，请执行以下操作：
   a. 点 **LVMCluster** 选项卡。
   b. 在 **Actions** 菜单中，选择 **Edit LVMCluster**。
   c. 点 **YAML** 并编辑所需的 **LVMCLuster** CR 参数。
   d. 点击 **Save**。

验证

1. 在 **LVMCLuster** 页面中，检查 **LVMCluster** CR 是否处于 **Ready** 状态。
2. 可选： 要查看 LVM Storage 为每个设备类创建的可用存储类，请点 **Storage → StorageClasses**。
3. 可选： 要查看 LVM Storage 为每个设备类创建的可用卷快照类，请点 **Storage → VolumeSnapshotClasses**。

其他资源

- 关于 LVMCluster 自定义资源

4.12.4.4.4. 使用 RHACM 创建 LVMCluster CR

使用 RHACM 安装 LVM 存储后，您必须创建一个 **LVMCluster** 自定义资源(CR)。

先决条件

- 已使用 RHACM 安装 LVM 存储。
- 您可以使用具有 **cluster-admin** 权限的账户访问 RHACM 集群。

- 您可以阅读 "About the LVMCluster 自定义资源" 部分。

流程

1. 使用 OpenShift Container Platform 凭证登录到 RHACM CLI。
2. 使用配置创建 **ConfigurationPolicy** CR YAML 文件以创建 **LVMCluster** CR：

   创建 LVMCluster CR 的 ConfigurationPolicy CR YAML 文件示例

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: lvms
  namespace: openshift-storage
spec:
  object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: lvm.topolvm.io/v1alpha1
      kind: LVMCluster
      metadata:
        name: my-lvmcluster
        namespace: openshift-storage
      spec:
        storage:
          deviceClasses: 1
# ...
            deviceSelector: 2
# ...
            thinPoolConfig: 3
# ...
            nodeSelector: 4
# ...
  remediationAction: enforce
  severity: low
```

   1. 包含将本地存储设备分配给 LVM 卷组的配置。
   2. 包含指定您要添加到 LVM 卷组的设备的路径，并强制擦除添加到 LVM 卷组中的设备。
   3. 包含 LVM 精简池配置。
   4. 包含要在其上创建 LVM 卷组的节点的配置。如果此字段为空，则考虑没有调度污点的所有节点都。

3. 运行以下命令来创建 **ConfigurationPolicy** CR：

```
$ oc create -f <file_name> -n <cluster_namespace> 1
```

   1. 安装 LVM 存储的 OpenShift Container Platform 集群的命名空间。

其他资源

- Red Hat Advanced Cluster Management for Kubernetes：在线安装
- 关于 LVMCluster 自定义资源

4.12.4.5. 删除 LVMCluster 自定义资源的方法

您可以使用 OpenShift CLI (**oc**)或 OpenShift Container Platform Web 控制台删除 **LVMCluster** 自定义资源 (CR)。如果您使用 Red Hat Advanced Cluster Management (RHACM)安装 LVM 存储，您还可以使用 RHACM 删除 **LVMCluster** CR。

删除 **LVMCluster** CR 后，LVM Storage 会删除以下 CR：

- **storageClass**
- **volumeSnapshotClass**
- **LVMVolumeGroup**
- **LVMVolumeGroupNodeStatus**

4.12.4.5.1. 使用 CLI 删除 LVMCluster CR

您可以使用 OpenShift CLI (**oc**)删除 **LVMCluster** 自定义资源(CR)。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 您已删除 LVM 存储置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 运行以下命令来删除 **LVMCluster** CR：

```
$ oc delete lvmcluster <lvmclustername> -n openshift-storage
```

验证

- 要验证 **LVMCluster** CR 已被删除，请运行以下命令：

```
$ oc get lvmcluster -n <namespace>
```

输出示例

```
No resources found in openshift-storage namespace.
```

4.12.4.5.2. 使用 Web 控制台删除 LVMCluster CR

您可以使用 OpenShift Container Platform Web 控制台删除 **LVMCluster** 自定义资源 (CR)。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 您已删除 LVM 存储置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators** 查看所有已安装的 Operator。
3. 点 **openshift-storage** 命名空间中的 **LVM Storage**。
4. 点 **LVMCluster** 选项卡。
5. 在 **Actions** 中，选择 **Delete LVMCluster**。
6. 点击 **Delete**。

验证

- 在 **LVMCLuster** 页面中，检查 **LVMCluster** CR 已被删除。

4.12.4.5.3. 使用 RHACM 删除 LVMCluster CR

如果您使用 Red Hat Advanced Cluster Management (RHACM) 安装 LVM 存储，您可以使用 RHACM 删除 **LVMCluster** CR。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 RHACM 集群。
- 您已删除 LVM 存储置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。

流程

1. 使用 OpenShift Container Platform 凭证登录到 RHACM CLI。
2. 删除为 **LVMCluster** CR 创建的 **ConfigurationPolicy** CR YAML 文件：

```
$ oc delete -f <file_name> -n <cluster_namespace> 1
```

1 安装 LVM 存储的 OpenShift Container Platform 集群的命名空间。

3. 创建 **Policy** CR YAML 文件以删除 **LVMCluster** CR：

删除 LVMCluster CR 的 Policy CR 示例

```
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: (3)
    matchExpressions:
      - key: mykey
        operator: In
        values:
          - myvalue
```

1. **policy-template** 中的 **spec.remediationAction** 被以前的 **spec.remediationAction** 值覆盖。
2. 此 **namespace** 字段必须具有 **openshift-storage** 值。
3. 配置选择集群的要求。在与选择条件匹配的集群上卸载 LVM 存储。

4. 运行以下命令来创建 **Policy** CR：

```
$ oc create -f <file_name> -n <namespace>
```

5. 创建 **Policy** CR YAML 文件来检查 **LVMCluster** CR 已被删除：

检查 LVMCluster CR 已被删除的 Policy CR 示例

```
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. **policy-template spec.remediationAction** 可以通过为 **spec.remediationAction** 加前缀参数进行覆盖。
2. **namespace** 字段必须具有 **openshift-storage** 值。

6. 运行以下命令来创建 **Policy** CR：

```
$ oc create -f <file_name> -n <namespace>
```

验证

- 运行以下命令，检查 **Policy** CR 的状态：

```
$ oc get policy -n <namespace>
```

输出示例

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

重要

Policy CR 必须处于 **Compliant** 状态。

4.12.4.6. 置备存储

使用 **LVMCluster** 自定义资源(CR) 创建 LVM 卷组后，您可以通过创建持久性卷声明 (PVC) 来置备存储。

以下是您可以为每个文件系统类型请求的最小存储大小：

- 块设备: 8 MiB
- **XFS**: 300 MiB

- **ext4**: 32 MiB

要创建 PVC，您必须创建一个 **PersistentVolumeClaim** 对象。

先决条件

- 您已创建了 **LVMCluster** CR。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 创建 **PersistentVolumeClaim** 对象：

   PersistentVolumeClaim 对象示例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: lvm-block-1 1
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block 2
  resources:
    requests:
      storage: 10Gi 3
    limits:
      storage: 20Gi 4
  storageClassName: lvms-vg1 5
```

   1. 为 PVC 指定名称。
   2. 要创建块 PVC，请将此字段设置为 **Block**。要创建文件 PVC，请将此字段设置为 **Filesystem**。
   3. 指定存储大小。如果值小于最小存储大小，则请求的存储大小将舍入到最小存储大小。您可以置备的总存储大小受逻辑卷管理器 (LVM) 精简池的大小和过度置备因素的限制。
   4. 可选：指定存储限制。将此字段设置为大于或等于最小存储大小的值。否则，PVC 创建会失败并显示错误。
   5. **storageClassName** 字段的值的格式为 **lvms-<device_class_name>**，其中 **<device_class_name>** 是 **LVMCluster** CR 中的 **deviceClasses.name** 字段的值。例如，如果 **deviceClasses.name** 字段设置为 **vg1**，您必须将 **storageClassName** 字段设置为 **lvms-vg1**。

   注意

   存储类的 **volumeBindingMode** 字段设置为 **WaitForFirstConsumer**。

3. 运行以下命令来创建 PVC：

```
# oc create -f <file_name> -n <application_namespace>
```

注意

在部署使用它们的 pod 之前，创建的 PVC 会一直处于 **Pending** 状态。

验证

- 要验证 PVC 是否已创建，请运行以下命令：

```
$ oc get pvc -n <namespace>
```

输出示例

```
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
lvm-block-1   Bound    pvc-e90169a8-fd71-4eea-93b8-817155f60e47   1Gi        RWO
lvms-vg1       5s
```

4.12.4.7. 扩展集群存储的方法

OpenShift Container Platform 为裸机用户置备的基础架构上的集群支持额外的 worker 节点。您可以通过添加带有可用存储的新 worker 节点或向现有 worker 节点添加新设备来扩展集群的存储。

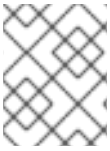
当节点处于活动状态时，逻辑卷管理器 (LVM) 存储会检测并使用额外的 worker 节点。

要在集群中的现有 worker 节点中添加新设备，您必须在 **LVMCluster** 自定义资源(CR)的 **deviceSelector** 字段中向新设备添加路径。

重要

您只能在创建 **LVMCluster** CR 时在 **LVMCluster** CR 中添加 **deviceSelector** 字段。如果您在创建 **LVMCluster** CR 时没有添加 **deviceSelector** 字段，您必须删除 **LVMCluster** CR 并创建一个包含 **deviceSelector** 字段的新 **LVMCluster** CR。

如果您没有在 **LVMCluster** CR 中添加 **deviceSelector** 字段，LVM Storage 会在设备可用时自动添加新设备。



注意

LVM 存储只添加支持的设备。有关不支持的设备的详情，请参考"LVM Storage 不支持的设备"。

其他资源

- 将 worker 节点添加到单节点 OpenShift 集群
- LVM 存储不支持的设备

4.12.4.7.1. 使用 CLI 扩展集群存储

您可以使用 OpenShift CLI (**oc**)扩展集群中的 worker 节点的存储容量。

先决条件

- 每个集群中都有额外的未使用的设备，供逻辑卷管理器(LVM)存储使用。
- 已安装 OpenShift CLI(**oc**)。
- 您已创建了 **LVMCluster** 自定义资源 (CR)。

流程

1. 运行以下命令来编辑 **LVMCluster** CR：

```
$ oc edit <lvmcluster_file_name> -n <namespace>
```

2. 在 **deviceSelector** 字段中添加新设备的路径。

LVMCluster CR 示例

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
spec:
  storage:
    deviceClasses:
# ...
      deviceSelector: 1
        paths: 2
        - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
        optionalPaths: 3
        - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
# ...
```

1 包含指定您要添加到 LVM 卷组的设备的路径。您可以在 **paths** 字段，**optionalPaths** 字段会这两个字段中指定设备路径。如果您没有在 **paths** 和 **optionalPaths** 中指定设备路径，则逻辑卷管理器 (LVM) 存储会将支持的未使用的设备添加到 LVM 卷组。只有在满足以下条件时，LVM 存储才会将设备添加到 LVM 卷组中：

- 设备路径存在。
- LVM 存储支持该设备。有关不支持的设备的详情，请参考"LVM Storage 不支持的设备"。

2 指定设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，则 **LVMCluster** CR 会进入 **Failed** 状态。

3 指定可选设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，LVM Storage 会忽略该设备而不造成错误。

重要

将设备添加到 LVM 卷组后，无法删除它。

3. 保存 **LVMCluster** CR。

其他资源

- 关于 LVMCluster 自定义资源
- LVM 存储不支持的设备
- 关于在卷组中添加设备

4.12.4.7.2. 使用 Web 控制台扩展集群存储

您可以使用 OpenShift Container Platform Web 控制台扩展集群中的 worker 节点的存储容量。

先决条件

- 每个集群中都有额外的未使用的设备，供逻辑卷管理器(LVM)存储使用。
- 您已创建了 **LVMCluster** 自定义资源 (CR)。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators**。
3. 点 **openshift-storage** 命名空间中的 **LVM Storage**。
4. 点 **LVMCluster** 选项卡查看在集群中创建的 **LVMCluster** CR。
5. 在 **Actions** 菜单中，选择 **Edit LVMCluster**。
6. 点 **YAML** 标签。
7. 编辑 **LVMCluster** CR，在 **deviceSelector** 字段中添加新设备路径：

   LVMCluster CR 示例

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: my-lvmcluster
spec:
  storage:
    deviceClasses:
# ...
      deviceSelector: (1)
        paths: (2)
        - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:88:00.0-nvme-1
        optionalPaths: (3)
        - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
        - /dev/disk/by-path/pci-0000:90:00.0-nvme-1
# ...
```

1. 包含指定您要添加到 LVM 卷组的设备的路径。您可以在 **paths** 字段，**optionalPaths** 字段会这两个字段中指定设备路径。如果您没有在 **paths** 和 **optionalPaths** 中指定设备路径，则
   - 设备路径存在。
   - LVM 存储支持该设备。有关不支持的设备的详情，请参考"LVM Storage 不支持的设备"。
2. 指定设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，则 **LVMCluster** CR 会进入 **Failed** 状态。
3. 指定可选设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，LVM Storage 会忽略该设备而不造成错误。

重要

将设备添加到 LVM 卷组后，无法删除它。

8. 点击 **Save**。

其他资源

- 关于 LVMCluster 自定义资源
- LVM 存储不支持的设备
- 关于在卷组中添加设备

4.12.4.7.3. 使用 RHACM 扩展集群存储

您可以使用 RHACM 扩展集群中的 worker 节点的存储容量。

先决条件

- 您可以使用具有 **cluster-admin** 特权的帐户访问 RHACM 集群。
- 已使用 RHACM 创建 **LVMCluster** 自定义资源 (CR)。
- 每个集群中都有额外的未使用的设备，供逻辑卷管理器(LVM)存储使用。

流程

1. 使用 OpenShift Container Platform 凭证登录到 RHACM CLI。
2. 运行以下命令，编辑使用 RHACM 创建的 **LVMCluster** CR：

```
$ oc edit -f <file_name> -ns <namespace>
```
(1)

1. 将 **<file_name>** 替换为 **LVMCluster** CR 的名称。

3. 在 **LVMCluster** CR 中，在 **deviceSelector** 字段中向新设备添加路径。

LVMCluster CR 示例：

```
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:
# ...
                  deviceSelector: (1)
                    paths: (2)
                    - /dev/disk/by-path/pci-0000:87:00.0-nvme-1
                    optionalPaths: (3)
                    - /dev/disk/by-path/pci-0000:89:00.0-nvme-1
# ...
```

(1) 包含指定您要添加到 LVM 卷组的设备的路径。您可以在 **paths** 字段，**optionalPaths** 字段会这两个字段中指定设备路径。如果您没有在 **paths** 和 **optionalPaths** 中指定设备路径，则逻辑卷管理器 (LVM) 存储会将支持的未使用的设备添加到 LVM 卷组。只有在满足以下条件时，LVM 存储才会将设备添加到 LVM 卷组中：

- 设备路径存在。
- LVM 存储支持该设备。有关不支持的设备的详情，请参考"LVM Storage 不支持的设备"。

(2) 指定设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，则 **LVMCluster** CR 会进入 **Failed** 状态。

(3) 指定可选设备路径。如果此字段中指定的设备路径不存在，或者 LVM Storage 不支持该设备，LVM Storage 会忽略该设备而不造成错误。

重要

将设备添加到 LVM 卷组后，无法删除它。

4. 保存 **LVMCluster** CR。

其他资源

- Red Hat Advanced Cluster Management for Kubernetes：在线安装
- 关于 LVMCluster 自定义资源
- LVM 存储不支持的设备

- 关于在卷组中添加设备

4.12.4.8. 扩展持久性卷声明

扩展集群的存储后，您可以扩展现有持久性卷声明 (PVC)。

要扩展 PVC，您必须更新 PVC 中的 **storage** 字段。

先决条件

- 使用动态置备。
- 与 PVC 关联的 **StorageClass** 对象将 **allowVolumeExpansion** 字段设置为 **true**。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 运行以下命令，将 **spec.resources.requests.storage** 字段的值更新为大于当前值的值：

```
$ oc patch <pvc_name> -n <application_namespace> -p \ 1
'{ "spec": { "resources": { "requests": { "storage": "<desired_size>" }}}} --type=merge' 2
```

1. 将 **<pvc_name>** 替换为您要扩展的 PVC 名称。
2. 将 **<desired_size>** 替换为新大小来扩展 PVC。

验证

- 要验证大小是否已完成，请运行以下命令：

```
$ oc get pvc <pvc_name> -n <application_namespace> -o=jsonpath=
{.status.capacity.storage}
```

LVM 存储在扩展过程中为 PVC 添加 **Resizing** 条件。它在 PVC 扩展后删除 **Resizing** 条件。

其他资源

- 扩展集群存储的方法
- 启用卷扩展支持

4.12.4.9. 删除持久性卷声明

您可以使用 OpenShift CLI (**oc**)删除持久性卷声明 (PVC)。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。

流程

1. 登录 OpenShift CLI (**oc**)。

2. 运行以下命令来删除 PVC：

```
$ oc delete pvc <pvc_name> -n <namespace>
```

验证

- 要验证 PVC 已被删除，请运行以下命令：

```
$ oc get pvc -n <namespace>
```

此命令的输出中不能存在已删除的 PVC。

4.12.4.10. 关于卷快照

您可以创建由 LVM Storage 置备的持久性卷声明(PVC) 的快照。

您可以使用卷快照执行以下操作：

- 备份应用程序数据。

重要

卷快照位于与原始数据相同的设备上。要将卷快照用作备份，您必须将快照移到安全位置。您可以使用 OpenShift API 进行数据保护 (OADP)备份和恢复解决方案。有关 OADP 的详情，请参考 "OADP features"。

- 恢复到进行卷快照时的状态。

注意

您还可以创建卷克隆的卷快照。

4.12.4.10.1. 在多节点拓扑中创建卷快照的限制

LVM 存储在多节点拓扑中创建卷快照有以下限制：

- 创建卷快照基于 LVM 精简池功能。
- 创建卷快照后，节点必须具有额外的存储空间才能进一步更新原始数据源。
- 您只能在部署原始数据源的节点上创建卷快照。
- 依赖于使用快照数据的 PVC 的 Pod 只能调度到部署原始数据源的节点。

其他资源

- OADP 功能

4.12.4.10.2. 创建卷快照

您可以根据精简池的可用容量和过度置备限制创建卷快照。要创建卷快照，您必须创建一个 **VolumeSnapshotClass** 对象。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 确保持久性卷声明(PVC)处于 **Bound** 状态。对于快照的一致性，这是必需的。
- 您已停止所有到 PVC 的 I/O。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 创建 **VolumeSnapshot** 对象：

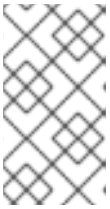
   VolumeSnapshot 对象示例

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: lvm-block-1-snap 1
spec:
  source:
    persistentVolumeClaimName: lvm-block-1 2
  volumeSnapshotClassName: lvms-vg1 3
```

   1 为卷快照指定一个名称。

   2 指定源 PVC 的名称。LVM Storage 会创建这个 PVC 的快照。

   3 将此字段设置为卷快照类的名称。

   注意

   要获取可用卷快照类列表，请运行以下命令：

```
$ oc get volumesnapshotclass
```

3. 运行以下命令，在创建源 PVC 的命名空间中创建卷快照：

```
$ oc create -f <file_name> -n <namespace>
```

   LVM 存储会创建一个 PVC 的只读副本作为卷快照。

验证

- 要验证卷快照是否已创建，请运行以下命令：

```
$ oc get volumesnapshot -n <namespace>
```

  输出示例

```
NAME              READYTOUSE   SOURCEPVC     SOURCESNAPSHOTCONTENT
```

```
RESTORESIZE   SNAPSHOTCLASS   SNAPSHOTCONTENT
CREATIONTIME   AGE
lvm-block-1-snap   true        lvms-test-1                  1Gi          lvms-vg1
snapcontent-af409f97-55fc-40cf-975f-71e44fa2ca91   19s          19s
```

您创建的卷快照的 **READYTOUSE** 字段的值必须是 **true**。

4.12.4.10.3. 恢复卷快照

要恢复卷快照，您必须创建一个 PVC，并将 **dataSource.name** 字段设置为卷快照的名称。

恢复的 PVC 独立于卷快照和源 PVC。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 您已创建了卷快照。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 使用配置创建 **PersistentVolumeClaim** 对象来恢复卷快照：

   恢复卷快照的 PersistentVolumeClaim 对象示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: lvm-block-1-restore
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Block
  Resources:
    Requests:
      storage: 2Gi (1)
  storageClassName: lvms-vg1 (2)
  dataSource:
    name: lvm-block-1-snap (3)
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

   (1) 指定恢复的 PVC 的存储大小。请求的 PVC 的存储大小必须大于或等于您要恢复的卷快照的大小。如果需要较大的 PVC，也可以在恢复卷快照后重新定义 PVC 的大小。

   (2) 将此字段设置为您要恢复的卷快照的源 PVC 中的 **storageClassName** 字段的值。

   (3) 将此字段设置为您要恢复的卷快照的名称。

3. 运行以下命令，在您创建卷快照的命名空间中创建 PVC：

```
$ oc create -f <file_name> -n <namespace>
```

验证

- 要验证卷快照是否已恢复，请运行以下命令：

```
$ oc get pvc -n <namespace>
```

输出示例

```
NAME                    STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
lvm-block-1-restore     Bound    pvc-e90169a8-fd71-4eea-93b8-817155f60e47   1Gi        RWO
lvms-vg1       5s
```

4.12.4.10.4. 删除卷快照

您可以删除持久性卷声明 (PVC) 的卷快照。

重要

当您删除持久性卷声明 (PVC) 时，LVM Storage 只删除 PVC，而不是删除 PVC 的快照。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 您已确保没有删除的卷 snpashot。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 运行以下命令来删除卷快照：

```
$ oc delete volumesnapshot <volume_snapshot_name> -n <namespace>
```

验证

- 要验证卷快照是否已删除，请运行以下命令：

```
$ oc get volumesnapshot -n <namespace>
```

此命令的输出中不能存在删除的卷快照。

4.12.4.11. 关于卷克隆

卷克隆是现有持久性卷声明 (PVC) 的副本。您可以创建一个卷克隆来复制数据的时间点副本。

4.12.4.11.1. 在多节点拓扑中创建卷克隆的限制

LVM 存储在多节点拓扑中创建卷克隆有以下限制：

- 创建卷克隆基于 LVM 精简池功能。
- 在创建卷克隆后，节点必须有额外的存储才能进一步更新原始数据源。
- 您只能在部署原始数据源的节点上创建卷克隆。
- 依赖于使用克隆数据的 PVC 的 Pod 只能调度到部署原始数据源的节点。

4.12.4.11.2. 创建卷克隆

要创建持久性卷声明 (PVC) 的克隆，您必须在创建源 PVC 的命名空间中创建 **PersistentVolumeClaim** 对象。

重要

克隆的 PVC 具有写入访问权限。

先决条件

- 确保源 PVC 处于 **Bound** 状态。这是一致的克隆所必需的。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 创建 **PersistentVolumeClaim** 对象：

用于创建卷克隆的 PersistentVolumeClaim 对象示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: lvm-pvc-clone
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: lvms-vg1 (1)
  volumeMode: Filesystem (2)
  dataSource:
    kind: PersistentVolumeClaim
    name: lvm-pvc (3)
  resources:
    requests:
      storage: 1Gi (4)
```

1. 将此字段设置为源 PVC 中的 **storageClassName** 字段的值。
2. 将此字段设置为源 PVC 中的 **volumeMode** 字段。
3. 指定源 PVC 的名称。
4. 指定克隆的 PVC 的存储大小。克隆的 PVC 的存储大小必须大于或等于源 PVC 的存储大小。

3. 运行以下命令，在创建源 PVC 的命名空间中创建 PVC：

```
$ oc create -f <file_name> -n <namespace>
```

验证

- 要验证卷克隆是否已创建，请运行以下命令：

```
$ oc get pvc -n <namespace>
```

输**出示例**

```
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
lvm-block-1-clone   Bound    pvc-e90169a8-fd71-4eea-93b8-817155f60e47   1Gi        RWO
lvms-vg1       5s
```

4.12.4.11.3. 删除卷克隆

您可以删除卷克隆。

重要

当您删除持久性卷声明 (PVC) 时，LVM Storage 只删除源持久性卷声明 (PVC)，而不是删除 PVC 的克隆。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 运行以下命令来删除克隆的 PVC：

```
# oc delete pvc <clone_pvc_name> -n <namespace>
```

验证

- 要验证卷克隆是否已删除，请运行以下命令：

```
$ oc get pvc -n <namespace>
```

此命令的输出中不能存在删除的卷克隆。

4.12.4.12. 更新 LVM 存储

您可以更新 LVM Storage 以确保与 OpenShift Container Platform 版本兼容。

先决条件

- 您已更新了 OpenShift Container Platform 集群。
- 已安装以前的 LVM 存储版本。
- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 运行以下命令，更新安装 LVM 存储时创建的 **Subscription** 自定义资源 (CR)：

```
$ oc patch subscription lvms-operator -n openshift-storage --type merge --patch '{"spec":
{"channel":"<update_channel>"}}' 1
```

1 将 **<update_channel>** 替换为您要安装的 LVM 存储版本。例如，**stable-4.15**。

3. 运行以下命令，查看更新事件以检查安装是否已完成：

```
$ oc get events -n openshift-storage
```

输出示例

```
...
8m13s       Normal   RequirementsUnknown   clusterserviceversion/lvms-operator.v4.15
requirements not yet checked
8m11s       Normal   RequirementsNotMet    clusterserviceversion/lvms-operator.v4.15   one
or more requirements couldn't be found
7m50s       Normal   AllRequirementsMet    clusterserviceversion/lvms-operator.v4.15   all
requirements found, attempting install
7m50s       Normal   InstallSucceeded      clusterserviceversion/lvms-operator.v4.15   waiting
for install components to report healthy
7m49s       Normal   InstallWaiting        clusterserviceversion/lvms-operator.v4.15   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.v4.15   install
strategy completed with no errors
...
```

验证

- 运行以下命令验证 LVM 存储版本：

```
$ oc get subscription lvms-operator -n openshift-storage -o jsonpath='{.status.installedCSV}'
```

输出示例

```
lvms-operator.v4.15
```

4.12.4.13. 监控 LVM 存储

要启用集群监控，您必须在安装 LVM 存储的命名空间中添加以下标签：

```
openshift.io/cluster-monitoring=true
```

重要

有关在 RHACM 中启用集群监控的详情，请参考 观察性和添加自定义指标。

4.12.4.13.1. 指标

您可以通过查看指标来监控 LVM 存储。

下表描述了 **topolvm** 指标：

表 4.7. **topolvm** 指标

警报	描述
topolvm_thinpool_data_percent	表示 LVM thinpool 中使用的数据空间百分比。
topolvm_thinpool_metadata_percent	表示 LVM thinpool 中使用的元数据空间百分比。
topolvm_thinpool_size_bytes	表示 LVM 精简池的大小（以字节为单位）。
topolvm_volumegroup_available_bytes	表示 LVM 卷组中的可用空间（以字节为单位）。
topolvm_volumegroup_size_bytes	表示 LVM 卷组的大小（以字节为单位）。
topolvm_thinpool_overprovisioned_available	表示 LVM 精简池的可用过度置备大小（以字节为单位）。

注意

指标每 10 分钟更新一次，或者在精简池中有变化（如创建新逻辑卷）时更新。

4.12.4.13.2. 警报

当精简池和卷组达到最大存储容量时，进一步的操作会失败。这会导致数据丢失。

当使用精简池和卷组超过特定值时，LVM 存储会发送以下警报：

表 4.8. LVM 存储警报

警报	描述
VolumeGroupUsageAtThresholdNearFull	当卷组和精简池用量超过节点上的 75% 时，会触发此警报。需要删除数据或卷组扩展。

警报	描述
VolumeGroupUsageAtThresholdCritical	当卷组和精简池使用超过节点上的 85% 时，会触发此警报。在这种情况下，卷组几乎已满。需要删除数据或卷组扩展。
ThinPoolDataUsageAtThresholdNearFull	当卷组中的精简池数据 uusage 超过节点上的 75% 时，会触发此警报。需要删除数据或精简池扩展。
ThinPoolDataUsageAtThresholdCritical	当卷组中的精简池数据使用量超过节点上的 85% 时，会触发此警报。需要删除数据或精简池扩展。
ThinPoolMetaDataUsageAtThresholdNearFull	当卷组中的精简池元数据使用量超过节点上的 75% 时，会触发此警报。需要删除数据或精简池扩展。
ThinPoolMetaDataUsageAtThresholdCritical	当卷组中的精简池元数据使用量超过节点上的 85% 时，会触发此警报。需要删除数据或精简池扩展。

4.12.4.14. 使用 CLI 卸载 LVM 存储

您可以使用 OpenShift CLI (**oc**)卸载 LVM Storage。

先决条件

- 已以具有 **cluster-admin** 权限的用户身份登录 **oc**。
- 已删除由 LVM Storage 置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。
- 已删除 **LVMCluster** 自定义资源 (CR)。

流程

1. 运行以下命令，获取 LVM Storage Operator 的 **currentCSV** 值：

```
$ oc get subscription.operators.coreos.com lvms-operator -n <namespace> -o yaml | grep currentCSV
```

输出示例

```
currentCSV: lvms-operator.v4.15.3
```

2. 运行以下命令来删除订阅：

```
$ oc delete subscription.operators.coreos.com lvms-operator -n <namespace>
```

输出示例

```
subscription.operators.coreos.com "lvms-operator" deleted
```

3. 运行以下命令，删除目标命名空间中 LVM Storage Operator 的 CSV：

```
$ oc delete clusterserviceversion <currentCSV> -n <namespace>
```
(1)

(1) 将 **<currentCSV>** 替换为 LVM Storage Operator 的 **currentCSV** 值。

输**出示例**

```
clusterserviceversion.operators.coreos.com "lvms-operator.v4.15.3" deleted
```

验证

- 要验证 LVM Storage Operator 是否已卸载，请运行以下命令：

```
$ oc get csv -n <namespace>
```

如果 LVM Storage Operator 已被成功卸载，则不会出现在这个命令的输出中。

4.12.4.15. 使用 Web 控制台卸载 LVM 存储

您可以使用 OpenShift Container Platform Web 控制台卸载 LVM Storage。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform。
- 您已删除 LVM 存储置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。
- 您已删除 **LVMCluster** 自定义资源 (CR)。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators**。
3. 点 **openshift-storage** 命名空间中的 **LVM Storage**。
4. 点 **Details** 标签页。
5. 在 **Actions** 菜单中，选择 **Uninstall Operator**。
6. 可选：在提示时，选择 **Delete all operand instances for this operator**复选框来删除 LVM Storage 的操作对象实例。
7. 点 **Uninstall**。

4.12.4.16. 卸载使用 RHACM 安装的 LVM 存储

要卸载使用 RHACM 安装的 LVM 存储，您必须删除为安装和配置 LVM 存储而创建的 RHACM **Policy** 自定义资源 (CR)。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 RHACM 集群。
- 您已删除 LVM 存储置备的持久性卷声明 (PVC)、卷快照和卷克隆。您还已删除了使用这些资源的应用程序。
- 您已删除使用 RHACM 创建的 **LVMCluster** CR。

流程

1. 登录 OpenShift CLI (**oc**)。
2. 使用以下命令删除为安装和配置 LVM 存储而创建的 RHACM **Policy** CR：

```
$ oc delete -f <policy> -n <namespace>
```
(1)

(1) 将 **<policy>** 替换为 **Policy** CR YAML 文件的名称。

3. 使用配置创建 **Policy** CR YAML 文件来卸载 LVM 存储：

卸载 LVM 存储的 Policy CR 示例

```
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
```

4. 运行以下命令来创建 **Policy** CR：

```
$ oc create -f <policy> -ns <namespace>
```

4.12.4.17. 使用 must-gather 下载日志文件和诊断信息

如果 LVM 存储无法自动解决问题，请使用 must-gather 工具收集日志文件和诊断信息，以便您或红帽支持可以查看问题并确定解决方案。

流程

- 从连接到 LVM 存储集群的客户端运行 **must-gather** 命令：

```
$ oc adm must-gather --image=registry.redhat.io/lvms4/lvms-must-gather-rhel9:v4.15 --dest-dir=<directory_name>
```

其他资源

- 关于 must-gather 工具

4.12.4.18. 持久性存储故障排除

在使用逻辑卷管理器 (LVM) 存储配置持久性存储时，您可能会遇到一些需要故障排除的问题。

4.12.4.18.1. 检查 PVC 处于 Pending 状态

持久性卷声明 (PVC) 可能会因为以下原因处于 **Pending** 状态：

- 计算资源不足。
- 网络问题。
- 不匹配的存储类或节点选择器。
- 没有可用的持久性卷 (PV)。
- 具有 PV 的节点处于 **Not Ready** 状态。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录到 OpenShift CLI (**oc**)。

流程

1. 运行以下命令来检索 PVC 列表：

```
$ oc get pvc
```

输**出示例**

```
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
lvms-test  Pending                                     lvms-vg1       11s
```

2. 运行以下命令，检查与 PVC 关联的事件处于 **Pending** 状态：

```
$ oc describe pvc <pvc_name> 1
```

1 将 **<pvc_name>** 替换为 PVC 的名称。例如：**lvms-vg1**。

输**出示例**

```
Type     Reason              Age              From                         Message
----     ------              ----             ----                         -------
Warning  ProvisioningFailed  4s (x2 over 17s)  persistentvolume-controller
storageclass.storage.k8s.io "lvms-vg1" not found
```

4.12.4.18.2. 从缺少的存储类中恢复

如果出现 **storage class not found** 错误，检查 **LVMCluster** 自定义资源 (CR)，并确保所有逻辑卷管理器 (LVM)存储 pod 都处于 **Running** 状态。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录到 OpenShift CLI (**oc**)。

流程

1. 运行以下命令验证 **LVMCluster** CR 是否存在：

```
$ oc get lvmcluster -n openshift-storage
```

输**出示例**

```
NAME           AGE
my-lvmcluster  65m
```

2. 如果 **LVMCluster** CR 不存在，请创建一个 **LVMCluster** CR。如需更多信息，请参阅"创建 LVMCluster 自定义资源"。

3. 在 **openshift-storage** 命名空间中，运行以下命令来检查所有 LVM 存储 pod 是否都处于 **Running** 状态：

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

输**出示例**

```
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
```

此命令的输出必须包含以下 pod 的运行实例：

- **lvms-operator**
- **vg-manager**
- **topolvm-controller**
- **topolvm-node**
  如果 **topolvm-node** pod 处于 **Init** 状态，则会因为无法找到 LVM 存储的可用磁盘失败。要检索必要的信息来排除此问题，请运行以下命令来查看 **vg-manager** pod 的日志：

```
$ oc logs -l app.kubernetes.io/component=vg-manager -n openshift-storage
```

其他资源

- 关于 LVMCluster 自定义资源
- 创建 LVMCluster 自定义资源的方法

4.12.4.18.3. 从节点故障中恢复

由于集群中的节点故障，持久性卷声明 (PVC) 可能会处于 **Pending** 状态。

要识别出现故障的节点，您可以检查 **topolvm-node** pod 的重启计数。增加了重启计数表示底层节点的潜在问题，这可能需要进一步调查和故障排除。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录到 OpenShift CLI (**oc**)。

流程

- 运行以下命令，检查 **topolvm-node** pod 实例的重启计数：

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

输**出示例**

```
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
```

后续步骤

- 如果 PVC 处于 **Pending** 状态，即使您解决了与节点相关的问题后，您必须执行强制清理。如需更多信息，请参阅"通知强制清理"。

其他资源

- 执行强制清理

4.12.4.18.4. 从磁盘失败中恢复

如果您在检查与持久性卷声明 (PVC) 关联的事件时看到失败信息，则可能代表底层卷或磁盘存在问题。

磁盘和卷置备问题会导致通用错误消息，如 **Failed to provision volume with storage class <storage_class_name>**。常规错误消息后带有特定的卷失败错误消息。

下表描述了卷故障错误消息：

表 4.9. 卷故障错误消息

错误消息	描述
检查卷是否存在失败	指明验证卷是否已存在的问题。卷验证失败可能是由网络连接问题或其他故障造成的。
绑定卷失败	如果可用持久性卷 (PV) 与 PVC 的要求不匹配，则无法绑定卷。
FailedMount 或 **FailedAttachVolume**	此错误表示试图将卷挂载到节点时出现问题。如果磁盘失败，pod 尝试使用 PVC 时可能会出现这个错误。
FailedUnMount	此错误表示尝试从节点卸载卷时出现问题。如果磁盘失败，pod 尝试使用 PVC 时可能会出现这个错误。
卷已专门附加到一个节点，不能附加到另一个节点	这个错误可能会出现不支持 **ReadWriteMany** 访问模式的存储解决方案。

先决条件

- 已安装 OpenShift CLI(**oc**)。

- 您已以具有 **cluster-admin** 权限的用户身份登录到 OpenShift CLI (**oc**)。

流程

1. 运行以下命令，检查与 PVC 关联的事件：

```
$ oc describe pvc <pvc_name> 1
```

1 将 **<pvc_name>** 替换为 PVC 的名称。

2. 建立到发生问题的主机的直接连接。
3. 解决磁盘问题。

后续步骤

- 如果卷失败消息在磁盘解决了这个问题后仍然保留或递归，您必须执行强制清理。如需更多信息，请参阅"通知强制清理"。

其他资源

- 执行强制清理

4.12.4.18.5. 执行强制清理

如果在完成故障排除过程后仍存在磁盘或节点相关的问题，您必须执行强制清理。强制清理用于解决永久问题并确保逻辑卷管理器(LVM)存储正常工作。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录到 OpenShift CLI (**oc**)。
- 您已删除所有使用 LVM 存储创建的持久性卷声明(PVC)。
- 您已停止使用 LVM 存储创建的 PVC 的 pod。

流程

1. 运行以下命令切换到 **openshift-storage** 命名空间：

```
$ oc project openshift-storage
```

2. 运行以下命令，检查 **LogicalVolume** 自定义资源 (CR) 是否存在：

```
$ oc get logicalvolume
```

a. 如果存在 **LogicalVolume** CR，请运行以下命令删除它们：

```
$ oc delete logicalvolume <name> 1
```

1 将 **<name>** 替换为 **LogicalVolume** CR 的名称。

b. 删除 **LogicalVolume** CR 后，运行以下命令来删除其终结器：

```
$ oc patch logicalvolume <name> -p '{"metadata":{"finalizers":[]}}' --type=merge 1
```

1 将 **<name>** 替换为 **LogicalVolume** CR 的名称。

3. 运行以下命令，检查 **LVMVolumeGroup** CR 是否存在：

```
$ oc get lvmvolumegroup
```

a. 如果存在 **LVMVolumeGroup** CR，请运行以下命令删除它们：

```
$ oc delete lvmvolumegroup <name> 1
```

1 将 **<name>** 替换为 **LVMVolumeGroup** CR 的名称。

b. 删除 **LVMVolumeGroup** CR 后，运行以下命令来删除其终结器：

```
$ oc patch lvmvolumegroup <name> -p '{"metadata":{"finalizers":[]}}' --type=merge 1
```

1 将 **<name>** 替换为 **LVMVolumeGroup** CR 的名称。

4. 运行以下命令来删除任何 **LVMVolumeGroupNodeStatus** CR：

```
$ oc delete lvmvolumegroupnodestatus --all
```

5. 运行以下命令来删除 **LVMCluster** CR：

```
$ oc delete lvmcluster --all
```

a. 删除 **LVMCluster** CR 后，运行以下命令来删除其终结器：

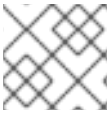
```
$ oc patch lvmcluster <name> -p '{"metadata":{"finalizers":[]}}' --type=merge 1
```

1 将 **<name>** 替换为 **LVMCluster** CR 的名称。

第 5 章 使用 CONTAINER STORAGE INTERFACE (CSI)

5.1. 配置 CSI 卷

容器存储接口 (CSI) 允许 OpenShift Container Platform 使用支持 CSI 接口的存储后端提供的持久性存储。



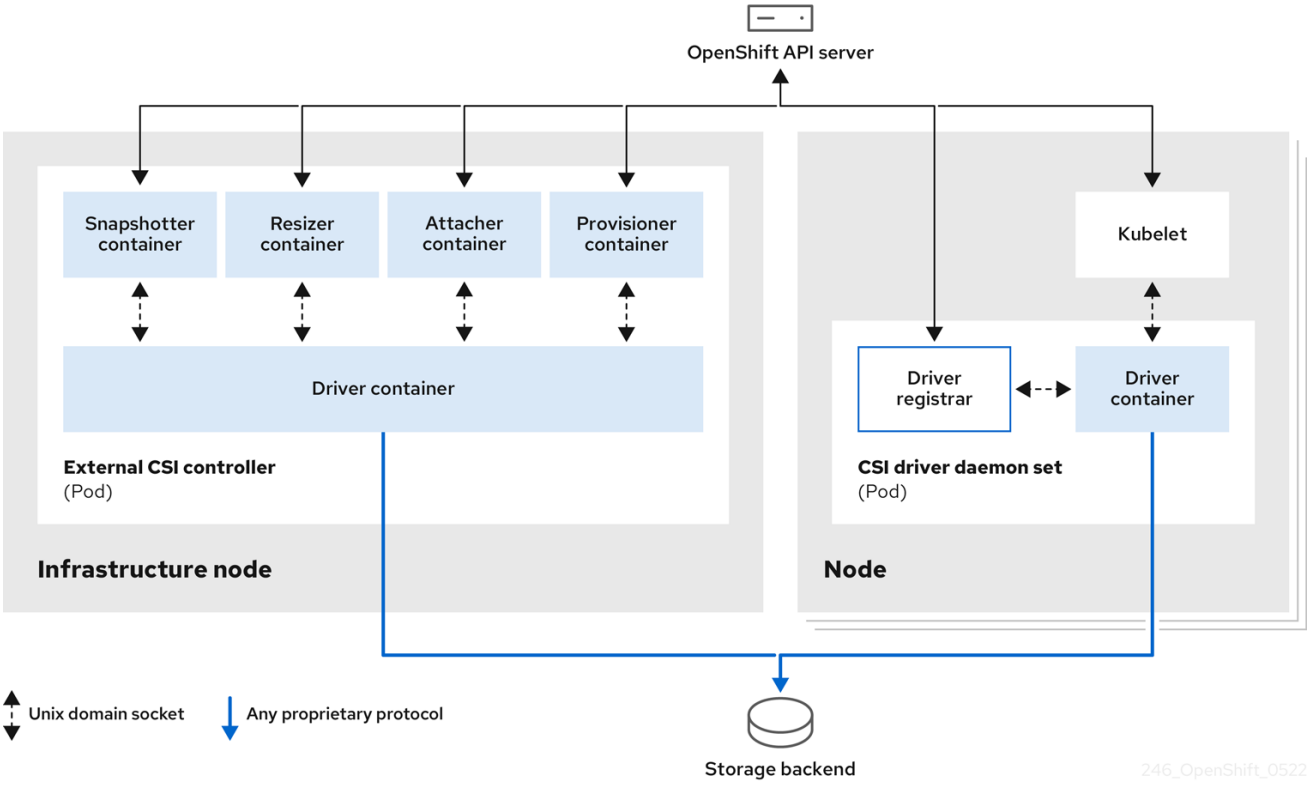
注意

OpenShift Container Platform 4.15 支持 CSI 规范 版本 1.6.0。

5.1.1. CSI 架构

CSI 驱动程序通常由容器镜像提供。这些容器不了解其运行的 OpenShift Container Platform。要在 OpenShift Container Platform 中使用与 CSI 兼容的存储后端，集群管理员必须部署几个组件，作为 OpenShift Container Platform 和存储驱动程序间的桥接。

下图提供了在 OpenShift Container Platform 集群中以 pod 运行的组件的概述。



对于不同的存储后端，可以运行多个 CSI 驱动程序。每个驱动程序需要其自身的外部控制器部署，以及带驱动程序和 CSI 注册器的守护进程集。

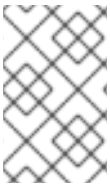
5.1.1.1. 外部 CSI 控制器

外部 CSI 控制器是一个部署，它部署带有以下五个容器的一个或多个 pod：

- snapshotter 容器监视 **VolumeSnapshot** 和 **VolumeSnapshotContent** 对象，并负责创建和删除 **VolumeSnapshotContent** 对象。

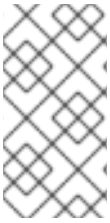
- resizer 容器是一个 sidecar 容器，它会在 PersistentVolumeClaim 对象中监视 **PersistentVolumeClaim** 更新，并在对 **PersistentVolumeClaim** 对象请求更多存储时针对 CSI 端点触发 **ControllerExpandVolume** 操作。
- 一个外部 CSI attacher 容器，它会将 OpenShift Container Platform 的 **attach** 和 **detach** 调用转换为相关的 CSI 驱动程序的 **ControllerPublish** 和 **ControllerUnpublish** 调用。
- 一个外部 CSI 置备程序容器，它可将 OpenShift Container Platform 的 **provision** 和 **delete** 调用转换为相应的 CSI 驱动程序的 **CreateVolume** 和 **DeleteVolume** 调用。
- 一个 CSI 驱动程序容器。

CSI attacher 和 CSI provisioner 容器使用 UNIX 域套接字与 CSI 驱动程序容器进行交互，确保没有 CSI 通讯会离开 pod。从 pod 以外无法访问 CSI 驱动程序。



注意

attach、 **detach**、 **provision**和 **delete** 操作通常需要 CSI 驱动程序在存储后端使用凭证。在 infrastructure 节点上运行 CSI controller pod，因此即使在一个计算节点上发生严重的安全破坏时，凭据也不会暴露给用户进程。



注意

当不支持第三方的 **attach** 或 **detach** 操作时，还需要为 CSI 驱动程序运行外部的附加器。外部附加器不会向 CSI 驱动程序发出任何 **ControllerPublish** 或 **ControllerUnpublish** 操作。然而，它仍必须运行方可实现所需的 OpenShift Container Platform attachment API。

5.1.1.2. CSI 驱动程序守护进程集

CSI 驱动程序守护进程集在每个节点上运行一个 pod，它允许 OpenShift Container Platform 挂载 CSI 驱动程序提供的存储，并使用它作为持久性卷 (PV) 的用户负载 (pod) 。安装了 CSI 驱动程序的 pod 包含以下容器：

- 一个 CSI 驱动程序注册器，它会在节点上运行的 **openshift-node** 服务中注册 CSI 驱动程序。在节点上运行的 **openshift-node** 进程然后使用节点上可用的 UNIX 域套接字直接连接到 CSI 驱动程序。
- 一个 CSI 驱动程序。

在节点上部署的 CSI 驱动程序应该在存储后端中拥有尽量少的凭证。OpenShift Container Platform 只使用节点插件的 CSI 调用集合，如**NodePublish/NodeUnpublish** 和 **NodeStage/NodeUnstage**（如果这些调用已被实现）。

5.1.2. OpenShift Container Platform 支持的 CSI 驱动程序

OpenShift Container Platform 默认安装某些 CSI 驱动程序，为用户提供树状卷插件无法进行的存储选项。

要创建挂载到这些支持的存储资产中的 CSI 置备的持久性卷，OpenShift Container Platform 会默认安装必要的 CSI 驱动程序 Operator、CSI 驱动程序和所需的存储类。如需有关 Operator 和驱动程序的默认命名空间的更多信息，请参阅特定 CSI Driver Operator 的文档。

重要

默认情况下，AWS EFS 和 GCP Filestore CSI 驱动程序不会被安装，必须手动安装。有关安装 AWS EFS CSI 驱动程序的步骤，请参阅设置 AWS Elastic File Service CSI Driver Operator。有关安装 GCP Filestore CSI 驱动程序的步骤，请参阅 Google Compute Platform Filestore CSI Driver Operator。

下表描述了 OpenShift Container Platform 支持的 OpenShift Container Platform 支持的 CSI 驱动程序及其支持的 CSI 功能，如卷快照和调整大小。

表 5.1. OpenShift Container Platform 中支持的 CSI 驱动程序和功能

CSI 驱动程序	CSI 卷快照	CSI 克隆	CSI 调整大小	内联临时卷
AliCloud Disk	▮	-	▮	-
AWS EBS	▮	-	▮	-
AWS EFS	-	-	-	-
Google Compute Platform (GCP) 持久磁盘 (PD)	▮	▮	▮	-
GCP Filestore	▮	-	▮	-
IBM Power® Virtual Server Block	-	-	▮	-
IBM Cloud® Block	▮[3]	-	▮[3]	-
LVM 存储	▮	▮	▮	-
Microsoft Azure Disk	▮	▮	▮	-
Microsoft Azure Stack Hub	▮	▮	▮	-
Microsoft Azure 文件	-	-	▮	▮
OpenStack Cinder	▮	▮	▮	-

CSI 驱动程序	CSI 卷快照	CSI 克隆	CSI 调整大小	内联临时卷
OpenShift Data Foundation	✓	✓	✓	-
OpenStack Manila	✓	-	-	-
共享资源	-	-	-	✓
VMware vSphere	✓[1]	-	✓[2]	-

1.

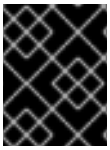
- 对于 vCenter Server 和 ESXi，需要 vSphere 版本 7.0 更新 3 或更高版本。
- 不支持 fileshare 卷。

2.

- 离线卷扩展：最低所需的 vSphere 版本为 6.7 更新 3 P06
- 在线卷扩展：最低的 vSphere 版本为 7.0 更新 2。

3.

- 不支持离线快照或调整大小。卷必须附加到正在运行的 pod。



重要

如果上表中没有列出您的 CSI 驱动程序，则需要按照 CSI 存储厂商提供的安装说明方可使用其支持的 CSI 功能。

5.1.3. 动态置备

动态置备持久性存储取决于 CSI 驱动程序和底层存储后端的功能。CSI 驱动的供应商应该提供了在 OpenShift Container Platform 中创建存储类及进行配置的参数文档。

创建的存储类可以被配置为启用动态置备。

流程

- 创建一个默认存储类，以保证所有不需要特殊存储类的 PVC 由安装的 CSI 驱动程序来置备。

```
# 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 要创建的存储类的名称。

2 已安装的 CSI 驱动程序名称。

5.1.4. 使用 CSI 驱动程序示例

以下示例在没有对该模板进行任何修改的情况下安装了一个默认的 MySQL 模板，。

先决条件

- CSI 驱动程序已被部署。
- 为动态置备创建了存储类。

流程

- 创建 MySQL 模板：

```
# oc new-app mysql-persistent
```

输出示例

```
--> Deploying template "openshift/mysql-persistent" to project default
...
```

```
# oc get pvc
```

输出示例

```
NAME             STATUS    VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
mysql            Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi
RWO            cinder         3s
```

5.1.5. 卷填充器

卷填充器使用持久性卷声明 (PVC) spec 中的 **datasource** 字段来创建预先填充的卷。

目前启用了卷填充，并作为技术预览功能支持。但是，OpenShift Container Platform 不附带任何卷填充器。

重要

卷填充程序只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

如需有关卷填充器的更多信息，请参阅 Kubernetes 卷填充器。

5.2. CSI INLINE 临时卷

借助容器存储接口（CSI）内联临时卷，您可以定义 **Pod** 规格，在 pod 部署时创建内联临时卷，并在 pod 销毁时删除它们。

此功能仅适用于受支持的 Container Storage Interface (CSI)驱动程序：

- 共享资源 CSI 驱动程序
- Azure File CSI 驱动程序
- Secret Store CSI 驱动程序

5.2.1. CSI 内联临时卷概述

通常，由 Container Storage Interface（CSI）驱动程序支持的卷只能用于 **PersistentVolume** 和 **PersistentVolumeClaim** 对象的组合。

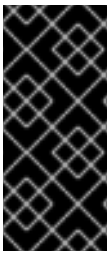
通过此功能，可以在 **Pod** 规格中直接指定 CSI 卷，而不是在 **PersistentVolume** 中指定。内联卷是临时的，在 pod 重启后不会保留。

5.2.1.1. 支持限制

在默认情况下，OpenShift Container Platform 支持 CSI 内联临时卷，但有以下限制：

- 仅支持 CSI 驱动程序。不支持 in-tree 和 FlexVolumes。
- 共享资源 CSI 驱动程序只支持使用内联临时卷访问多个命名空间中的 **Secret** 或 **ConfigMap**，作为技术预览功能。
- 社区或存储供应商提供其他支持这些卷的 CSI 驱动程序。按照 CSI 驱动程序供应商提供的说明进行操作。

CSI 驱动程序可能没有实现内联卷功能，包括 **Ephemeral** 能力。详情请查看 CSI 驱动程序文档。



重要

共享资源 CSI 驱动程序只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

5.2.2. CSI Volume Admission 插件

Container Storage Interface (CSI) Volume Admission 插件允许您限制在 pod 准入上置备 CSI 临时卷的独立 CSI 驱动程序的使用。管理员可以添加 **csi-ephemeral-volume-profile** 标签，然后 Admission 插件会检查该标签，并在强制、警告和审核决策中使用。

5.2.2.1. 概述

要使用 CSI Volume Admission 插件，管理员将 **security.openshift.io/csi-ephemeral-volume-profile** 标签添加到 **CSIDriver** 对象，它会在用来提供 CSI 驱动程序时声明 CSI 驱动程序的有效 pod 安全配置集，如下例所示：

```
kind: CSIDriver
metadata:
  name: csi.mydriver.company.org
  labels:
    security.openshift.io/csi-ephemeral-volume-profile: restricted (1)
```

(1) CSI 驱动程序对象 YAML 文件，将 **csi-ephemeral-volume-profile** 标签设置为 "restricted"

此"有效配置集"通信 pod 可以使用 CSI 驱动程序在 pod 安全标准管理时挂载 CSI 临时卷。

CSI Volume Admission 插件在创建 pod 时检查 pod 卷 ; 使用 CSI 卷的现有 pod 不受影响。如果 pod 使用容器存储接口(CSI)卷，插件会查找 **CSIDriver** 对象并检查 **csi-ephemeral-volume-profile** 标签，然后在强制、警告和审计决策中使用标签值。

5.2.2.2. Pod 安全配置集强制

当 CSI 驱动程序具有 **csi-ephemeral-volume-profile** 标签时，使用 CSI 驱动程序挂载 CSI 临时卷的 pod 必须在强制实施相等或更高权限的 pod 安全标准的命名空间中运行。如果命名空间强制实施更严格的标准，CSI Volume Admission 插件会拒绝准入。下表描述了给定标签值的不同 pod 安全配置集的强制行为。

表 5.2. Pod 安全配置集强制

Pod 安全配置集	驱动程序标签： restricted	驱动程序标签： baseline	驱动程序标签： privileged
Restricted	Allowed	Denied	Denied
Baseline	Allowed	Allowed	Denied
Privileged	Allowed	Allowed	Allowed

5.2.2.3. Pod 安全配置集警告

如果 CSI 驱动程序的有效配置集比 pod 命名空间的 pod 安全警告配置集更宽松，则 CSI Volume Admission 插件可能会发出警告。下表显示了何时为给定标签值的不同 pod 安全配置集发生警告。

表 5.3. Pod 安全配置集警告

Pod 安全配置集	驱动程序标签： restricted	驱动程序标签： baseline	驱动程序标签： privileged
Restricted	No warning	Warning	Warning
Baseline	No warning	No warning	Warning
Privileged	No warning	No warning	No warning

5.2.2.4. Pod 安全配置集审核

如果 CSI 驱动程序的有效配置集比 pod 命名空间的 pod 安全审计配置集更宽松，则 CSI Volume Admission 插件可以将 audit 注解应用到 pod。下表显示了针对给定标签值的不同 pod 安全配置集应用的 audit 注解。

表 5.4. Pod 安全配置集审核

Pod 安全配置集	驱动程序标签： restricted	驱动程序标签： baseline	驱动程序标签： privileged
Restricted	No audit	Audit	Audit
Baseline	No audit	No audit	Audit
Privileged	No audit	No audit	No audit

5.2.2.5. CSI Volume Admission 插件的默认行为

如果 CSI 临时卷引用的 CSI 驱动程序没有 **csi-ephemeral-volume-profile** 标签，CSI Volume Admission 插件会考虑驱动程序具有强制、警告和审计行为的 privileged 配置集。同样，如果 pod 的命名空间没有设置 pod 安全准入标签，Admission 插件会假定允许 restricted 配置集强制、警告和审核决策。因此，如果没有设置标签，则使用该 CSI 驱动程序的 CSI 临时卷默认仅在特权命名空间中可用。

OpenShift Container Platform 和支持临时卷附带的 CSI 驱动程序为 **csi-ephemeral-volume-profile** 标签设置了合理的默认设置：

- 共享资源 CSI 驱动程序： restricted
- Azure File CSI 驱动程序： privileged

如果需要，管理员可以更改标签的默认值。

5.2.3. 在 pod 规格中嵌入 CSI 内联临时卷

您可以在 OpenShift Container Platform 中的 **Pod** 规格中嵌入 CSI 内联临时卷。在运行时，嵌套的内联卷遵循与其关联的 Pod 的临时生命周期，以便 CSI 驱动程序在 Pod 创建和销毁时处理卷操作的所有阶段。

流程

1. 创建 **Pod** 对象定义，并将其保存到文件中。
2. 在该文件中嵌入 CSI 内联临时卷。

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 Pod 使用的卷的名称。

3. 创建在上一步中保存的对象定义文件。

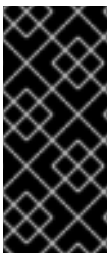
```
$ oc create -f my-csi-app.yaml
```

5.2.4. 其他资源

- Pod 安全标准

5.3. 共享资源 CSI DRIVER OPERATOR

作为集群管理员，您可以在 OpenShift Container Platform 中使用 Shared Resource CSI Driver 来置备包含 **Secret** 或 **ConfigMap** 对象的内联临时卷。这样，pod 和其他 Kubernetes 类型用来公开卷挂载，OpenShift Container Platform 构建可以安全地在集群中的任何命名空间中使用这些对象的内容。要达到此目的，目前有两种类型的共享资源： **Secret** 对象的 **SharedSecret** 自定义资源，以及 **ConfigMap** 对象的 **SharedConfigMap** 自定义资源。



重要

Shared Resource CSI 驱动程序只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

注意

要启用共享资源 CSI 驱动程序，您必须使用功能门启用功能。

5.3.1. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.3.2. 在命名空间间共享 secret

要在集群中的命名空间间共享 secret，您可以为您要共享的 **Secret** 对象创建一个 **SharedSecret** 自定义资源(CR)实例。

先决条件

您必须具有执行以下操作的权限：

- 在集群范围的级别上创建 **sharedsecrets.sharedresource.openshift.io** 自定义资源定义(CRD)的实例。
- 管理集群中命名空间中的角色和角色绑定，以控制哪些用户可以获取、列出和监视这些实例。
- 管理角色和角色绑定，以控制 Pod 指定的服务帐户是否可以挂载引用您要使用的 **SharedSecret** CR 实例的 Container Storage Interface(CSI)卷。
- 访问包含您要共享 Secret 的命名空间。

流程

- 为您要在集群中的命名空间之间共享的 **Secret** 对象创建一个 **SharedSecret** CR 实例：

```
$ 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. 在 pod 中使用 SharedSecret 实例

要从 Pod 访问 **SharedSecret** 自定义资源(CR)实例，您需要向给定的服务帐户 RBAC 权限授予该 **SharedSecret** CR 实例。

先决条件

- 您已为要在集群中的命名空间间共享的 secret 创建 **SharedSecret** CR 实例。

- 您必须具有执行以下操作的权限
  - 输入 **oc get sharedsecrets** 命令并返回非空列表来发现哪些 **SharedSecret** CR 实例可用。
  - 确定您的 Pod 指定的服务帐户是否可以使用给定的 **SharedSecret** CR 实例。也就是说，您可以运行 **oc adm policy who-can use <identifier of specific SharedSecret>** 来查看是否列出命名空间中的服务帐户。
  - 确定您的 Pod 指定的服务帐户是否允许 **csi** 卷使用 csi 卷，或者作为直接创建 pod 的请求用户使用 **csi** 卷。详情请参阅"识别和管理 pod 安全准入"。

注意

如果没有满足此列表中的最后两个先决条件，则建立或询问某人建立所需的基于角色的访问控制(RBAC)，以便您可以发现 **SharedSecret** CR 实例，并启用服务帐户使用 **SharedSecret** CR 实例。

流程

1. 通过使用带有 YAML 内容的 **oc apply**，为给定服务帐户 RBAC 权限授予其 pod 中的 **SharedSecret** CR 实例：

注意

目前，**kubectl** 和 **oc** 具有硬编码的特殊大小写逻辑，以便将 **use** 动词的使用限制为只限于 pod 安全性相关的角色。因此，您无法使用 **oc create role ...** 创建使用 **SharedSecret** CR 实例所需的角色。

```
$ 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. 使用 **oc** 命令创建与角色关联的 **RoleBinding** ：

```
$ oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. 从 Pod 访问 **SharedSecret** CR 实例：

```
$ 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. 在命名空间间共享配置映射

要在集群中的命名空间间共享配置映射，您可以为该配置映射创建 **SharedConfigMap** 自定义资源(CR)实例。

先决条件

您必须具有执行以下操作的权限：

- 在集群范围的级别上创建 **sharedconfigmaps.sharedresource.openshift.io** 自定义资源定义(CRD)的实例。
- 管理集群中命名空间中的角色和角色绑定，以控制哪些用户可以获取、列出和监视这些实例。
- 管理集群中命名空间中的角色和角色绑定，以控制挂载 Container Storage Interface(CSI)卷的 pod 中的哪些服务帐户可以使用这些实例。
- 访问包含您要共享 Secret 的命名空间。

流程

1. 为您要在集群中的命名空间之间共享的配置映射创建 **SharedConfigMap** CR 实例：

```
$ 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
```

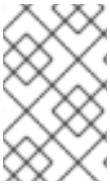
5.3.5. 在 pod 中使用 SharedConfigMap 实例

后续步骤

要从 pod 访问 **SharedConfigMap** 自定义资源(CR)实例，您需要授予给定服务帐户 RBAC 权限以使用该 **SharedConfigMap** CR 实例。

先决条件

- 为您要在集群中的命名空间之间共享的配置映射创建 **SharedConfigMap** CR 实例。
- 您必须具有执行以下操作的权限：
  - 输入 **oc get sharedconfigmaps** 命令并返回非空列表来发现哪些 **SharedConfigMap** CR 实例可用。
  - 确定您的 Pod 指定的服务帐户是否可以使用给定的 **SharedSecret** CR 实例。也就是说，您可以运行 **oc adm policy who-can use <identifier of specific SharedSecret>** 来查看是否列出命名空间中的服务帐户。
  - 确定您的 Pod 指定的服务帐户是否允许 **csi** 卷使用 csi 卷，或者作为直接创建 pod 的请求用户使用 **csi** 卷。详情请参阅"识别和管理 pod 安全准入"。



注意

如果没有满足此列表中的最后两个先决条件，则建立或询问某人建立所需的基于角色的访问控制(RBAC)，以便您可以发现 **SharedConfigMap** CR 实例，并启用服务帐户使用 **SharedConfigMap** CR 实例。

流程

1. 通过使用带有 YAML 内容的 **oc apply**，为给定服务帐户 RBAC 权限授予其 pod 中的 **SharedConfigMap** CR 实例。

注意

目前，**kubectl** 和 **oc** 具有硬编码的特殊大小写逻辑，以便将 **use** 动词的使用限制为只限于 pod 安全性相关的角色。因此，您无法使用 **oc create role ...** 创建使用 **SharedConfigMap** CR 实例所需的角色。

```
$ 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. 使用 **oc** 命令创建与角色关联的 **RoleBinding** ：

```
oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. 从 pod 访问 **SharedConfigMap** CR 实例：

```
$ 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. 共享资源 CSI 驱动程序的额外支持限制

共享资源 CSI 驱动程序有以下值得注意的限制：

- 驱动程序会受 Container Storage Interface(CSI)内联临时卷的限制。
- **readOnly** 字段的值必须是 **true**。在创建 **Pod** 时，如果 **readOnly** 为 **false**，验证准入 Webhook 会拒绝创建 pod。如果出于某种原因，验证准入 Webhook 无法在 pod 启动过程中在卷置备上联系，则驱动程序会向 kubelet 返回错误。要求 **readOnly** 为 **true**，可以保持上游 Kubernetes CSI 驱动程序的建议最佳实践，以将 SELinux 标签应用到关联的卷。
- 驱动程序忽略 **FSType** 字段，因为它仅支持 **tmpfs** 卷。
- 驱动程序忽略 **NodePublishSecretRef** 字段。相反，它使用 **SubjectAccessReviews** 和 **use** 动词来评估 pod 是否可以获取包含 **SharedSecret** 或 **SharedConfigMap** 自定义资源(CR)实例的卷。
- 您无法创建名称以 **openshift** 开头的 **SharedSecret** 或 **SharedConfigMap** 自定义资源 (CR) 实例。

5.3.7. 有关共享资源 pod 卷上的 VolumeAttributes 的更多详情

以下属性通过多种方式影响共享资源 pod 卷：

- **volumeAttributes** 属性中的 **refreshResource** 属性。

- Shared Resource CSI Driver 配置中的 **refreshResources** 属性。
- **volumeAttributes** 属性中的 **sharedSecret** 和 **sharedConfigMap** 属性。

5.3.7.1. refreshResource 属性

Shared Resource CSI Driver 遵循卷的 **volumeAttributes** 属性中的 **refreshResource** 属性。此属性用于控制，在卷作为 pod 启动的一部分被初始置备后，的过程中，对底层 **Secret** 或 **ConfigMap** 对象的内容的更新是否会被复制到卷中。**refreshResource** 的默认值为 **true**，这意味着内容会被更新。

重要

如果 Shared Resource CSI Driver 配置禁用了对 **sharedSecret** 和 **SharedConfigMap** 自定义资源(CR)实例的刷新，则 **volumeAttribute** 属性中的 **refreshResource** 属性没有作用。这个属性的目的是在通常允许刷新时为特定卷挂载禁用刷新。

5.3.7.2. refreshResources 属性

您可以使用全局开关启用或禁用共享资源刷新。这个开关是 Shared Resource CSI Driver 的 **csi-driver-shared-resource-config** 配置映射中的 **refreshResources** 属性，您可以在 **openshift-cluster-csi-drivers** 命名空间中找到。如果将这个 **refreshResources** 属性设置为 **false**，则卷的初始置备后不会更新卷中存储的 **Secret** 或 **ConfigMap** 对象相关内容。

重要

使用此共享资源 CSI 驱动程序配置来禁用刷新会影响所有使用共享资源 CSI Driver 的卷挂载，无论这些卷的 **volumeAttributes** 属性中的 **refreshResource** 属性是什么。

5.3.7.3. 在为 pod 置备共享资源卷前验证 volumeAttribute

在单个卷的 **volumeAttributes** 中，您必须将 **sharedSecret** 或 **sharedConfigMap** 属性的值设置为 **SharedSecret** 或 **SharedConfigMap** CS 实例的值。否则，当在 pod 启动过程中置备卷时，验证会检查该卷的 **volumeAttributes**，并在出现以下条件时向 kubelet 返回错误：

- **sharedSecret** 和 **sharedConfigMap** 属性都指定了值。
- **sharedSecret** 和 **sharedConfigMap** 属性都没有指定值。
- **sharedSecret** 或 **sharedConfigMap** 属性的值与集群中的 **SharedSecret** 或 **SharedConfigMap** CR 实例的名称不对应。

5.3.8. 共享资源、Insights Operator 和 OpenShift Container Platform 构建之间的集成

共享资源、Insights Operator 和 OpenShift Container Platform 构建之间的集成可以在 OpenShift Container Platform 构建中更轻松地使用红帽订阅（RHEL 权利）。

在以前的版本中，在 OpenShift Container Platform 4.9.x 及更早版本中，您可以手动导入凭证并将其复制到运行构建的每个项目或命名空间中。

现在，在 OpenShift Container Platform 4.10 及更新的版本中，OpenShift Container Platform 构建可以通过引用共享资源和 Insights Operator 提供的简单内容访问功能来使用红帽订阅（RHEL 权利）：

- 简单的内容访问功能将您的订阅凭证导入到众所周知的 **Secret** 对象。请参阅以下 "Additional resources" 部分的链接。

- 集群管理员创建与该 **Secret** 对象相关的 **SharedSecret** 自定义资源(CR)实例，并授予特定项目或命名空间的权限。特别是，集群管理员会赋予 **builder** 服务帐户权限，以使用该 **SharedSecret** CR 实例。
- 在这些项目或命名空间中运行的构建可以挂载一个 CSI 卷，该卷引用 **SharedSecret** CR 实例及其授权的 RHEL 内容。

其他资源

- 使用 Insights Operator 导入简单的内容访问证书
- 将订阅权利添加为构建 secret

5.4. CSI 卷快照

本文档论述了如何通过支持的 Container Storage Interface (CSI) 驱动程序使用进行卷快照，以帮助防止 OpenShift Container Platform 中的数据丢失。建议先熟悉持久性卷。

5.4.1. CSI 卷快照概述

*快照（snapshot）*代表了集群中特定时间点的存储卷的状态 。卷快照可以用来置备新卷。

OpenShift Container Platform 默认支持 Container Storage Interface(CSI)卷快照。但是，需要一个特定的 CSI 驱动程序。

通过 CSI 卷快照，集群管理员能够：

- 部署支持快照功能的第三方 CSI 驱动。
- 从一个现有的卷快照创建一个新的 PVC。
- 对现有的 PVC 进行快照。
- 将快照恢复为一个不同的 PVC。
- 删除现有的卷快照。

通过 CSI 卷快照，应用程序开发人员可以：

- 使用卷快照作为构建块来开发应用程序或集群级别的存储备份解决方案。
- 快速回滚到以前的开发版本。
- 不需要每次都进行完全备份，从而可以更有效地使用存储。

在使用卷快照时请注意以下几点：

- 仅支持 CSI 驱动程序。不支持 in-tree 和 FlexVolumes。
- OpenShift Container Platform 仅附带所选 CSI 驱动程序。对于不是由 OpenShift Container Platform Driver Operator 提供的 CSI 驱动程序，建议使用由社区或存储供应商提供的 CSI 驱动程序。按照 CSI 驱动程序供应商的安装说明进行操作。
- CSI 驱动程序可能会也可能不会带有卷快照功能。提供卷快照支持的 CSI 驱动程序可能会使用 **csi-external-snapshotter** sidecar。详情请查看 CSI 驱动程序提供的文档。

5.4.2. CSI 快照控制器和 sidecar

OpenShift Container Platform 提供了一个部署到 control plane 中的快照控制器。另外，您的 CSI 驱动程序厂商会提供 CSI 快照 sidecar，它会在安装 CSI 驱动程序的过程中做为一个辅助（helper）容器。

CSI 快照控制器和 sidecar 通过 OpenShift Container Platform API 提供卷快照。这些外部组件在集群中运行。

外部控制器由 CSI Snapshot Controller Operator 部署。

5.4.2.1. 外部控制器

CSI 快照控制器绑定 **VolumeSnapshot** 和 **VolumeSnapshotContent** 对象。控制器通过创建和删除 **VolumeSnapshotContent** 对象来管理动态置备。

5.4.2.2. 外部 sidecar

您的 CSI 驱动程序厂商提供 **csi-external-snapshotter** sidecar。这是和 CSI 驱动程序一起部署的单独的 helper 容器。sidecar 通过触发 **CreateSnapshot** 和 **DeleteSnapshot** 操作来管理快照。请根据驱动程序厂商提供的说明进行操作。

5.4.3. 关于 CSI Snapshot Controller Operator

CSI Snapshot Controller Operator 在 **openshift-cluster-storage-operator** 命名空间中运行。默认情况下，它由 Cluster Version Operator (CVO) 在所有集群中安装。

CSI Snapshot Controller Operator 安装 CSI 快照控制器，该控制器在 **openshift-cluster-storage-operator** 命名空间中运行。

5.4.3.1. 卷快照 CRD

在 OpenShift Container Platform 安装过程中，CSI Snapshot Controller Operator 在 **snapshot.storage.k8s.io/v1** API 组中创建以下快照自定义资源定义 (CRD) ：

VolumeSnapshotContent

一个快照记录了由集群管理员在集群中置备的卷的状态。
与 **PersistentVolume** 对象类似，**VolumeSnapshotContent** CRD 是一个集群资源，指向存储后端的实际快照。

对于手动预置备的快照，集群管理员会创建大量 **VolumeSnapshotContent** CRD。它们在存储系统中记录了实际卷快照的详情。

VolumeSnapshotContent CRD 没有命名空间，供集群管理员使用。

VolumeSnapshot

与 **PersistentVolumeClaim** 对象类似， **VolumeSnapshot** CRD 定义了开发人员对快照的请求。CSI Snapshot Controller Operator 运行 CSI 快照控制器，该控制器使用适当的 **VolumeSnapshotContent** CRD 处理 **VolumeSnapshot** CRD 的绑定。绑定是一个一对一的映射。
VolumeSnapshot CRD 有命名空间限制。开发人员使用 CRD 作为快照的唯一请求。

VolumeSnapshotClass

集群管理员可以指定属于 **VolumeSnapshot** 对象的不同属性。这些属性可能会在存储系统中使用相同卷的快照之间有所不同，在这种情况下，使用持久性卷声明的相同存储类来表示它们。

VolumeSnapshotClass CRD 定义了在创建快照时要使用的 **csi-external-snapshotter** sidecar 的参数。这可让存储后端知道在支持多个选项时动态创建哪些快照。

动态置备的快照使用 **VolumeSnapshotClass** CRD 指定在创建快照时要使用的 storage-provider 特定参数。

VolumeSnapshotContentClass CRD 没有命名空间，集群管理员使用它为存储后端启用全局配置选项。

5.4.4. 置备卷快照

置备快照的方法有两种：动态和手动。

5.4.4.1. 动态置备

您可以请求从持久性卷声明中动态获取快照，而不使用已存在的快照。参数可以通过 **VolumeSnapshotClass** CRD 指定。

5.4.4.2. 手动调配

作为集群管理员，您可以手动置备大量 **VolumeSnapshotContent** 对象。它们包括了集群用户可以获得的实际卷的快照详情。

5.4.5. 创建卷快照

当您创建 **VolumeSnapshot** 对象时，OpenShift Container Platform 会创建一个卷快照。

先决条件

- 登陆到一个正在运行的 OpenShift Container Platform 集群。
- 使用支持 **VolumeSnapshot** 对象的 CSI 驱动程序创建的 PVC。
- 用于置备存储后端的存储类。
- 您要对其进行快照的 PVC 没有被任何 pod 使用。

警告

为 pod 使用的 PVC 创建卷快照可能会导致无法写入数据和缓存的数据从快照中排除。要确保所有数据都写入磁盘，请在创建快照前删除使用 PVC 的 pod。

流程

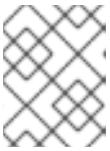
动态创建卷快照：

1. 使用以下 YAML 描述的 **VolumeSnapshotClass** 对象创建一个文件：

volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io (1)
deletionPolicy: Delete
```

1. 用于创建此 **VolumeSnapshotClass** 对象快照的 CSI 驱动程序名称。该名称必须与存储类的 **Provisioner** 字段相同，它负责正在进行快照的 PVC。



注意

根据用来配置持久性存储的驱动程序，可能需要额外的参数。您还可以使用现有的 **VolumeSnapshotClass** 对象。

2. 运行以下命令，创建上一步中保存的对象：

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

3. 创建 **VolumeSnapshot** 对象：

volumesnapshot-dynamic.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: mysnap
spec:
  volumeSnapshotClassName: csi-hostpath-snap (1)
  source:
    persistentVolumeClaimName: myclaim (2)
```

1. 卷快照对特定类的请求。如果 **volumeSnapshotClassName** 设置不存在，且有默认的卷快照类，则会创建一个带有默认卷快照类名称的快照。但如果该字段不存在且不存在默认卷快照类，则不会创建快照。

2. 绑定到持久性卷的 **PersistentVolumeClaim** 对象的名称。这指定了您要对什么创建快照。动态置备快照需要这个信息。

4. 运行以下命令，创建上一步中保存的对象：

```
$ oc create -f volumesnapshot-dynamic.yaml
```

手动置备快照：

1. 除了以上定义卷快照类外，还需要提供 **volumeSnapshotContentName** 参数的值作为快照的源。

volumesnapshot-manual.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  source:
    volumeSnapshotContentName: mycontent 1
```

1 预置备快照需要 **volumeSnapshotContentName** 参数。

2. 运行以下命令，创建上一步中保存的对象：

```
$ oc create -f volumesnapshot-manual.yaml
```

验证

在集群中创建快照后，会提供有关快照的详情。

1. 运行以下命令显示所创建的卷快照详情：

```
$ oc describe volumesnapshot mysnap
```

以下示例显示有关 **mysnap** 卷快照的详细信息：

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 指向控制器创建的实际存储内容的指针。

2 创建快照的时间。快照包含在这个指定时间点上可用的卷内容。

3 如果该值设为 **true**，则快照可用来恢复为一个新 PVC。
如果该值设为 **false**，则创建快照。但是，存储后端需要执行额外的任务来使快照可用，以便将其恢复为新卷。例如: Amazon Elastic Block Store 数据可能被移到不同的、更低成本的位置，这个过程可能需要几分钟时间。

2. 要验证卷快照是否已创建，请运行以下命令：

```
$ oc get volumesnapshotcontent
```

显示指向实际内容的指针。如果 **boundVolumeSnapshotContentName** 字段已被填充，则代表一个 **VolumeSnapshotContent** 对象已存在，快照被创建。

3. 要验证快照是否已就绪，请确认 **VolumeSnapshot** 带有 **readyToUse: true**。

5.4.6. 删除卷快照

您可以配置 OpenShift Container Platform 如何删除卷快照。

流程

1. 指定 **VolumeSnapshotClass** 对象中所需的删除策略，如下例所示：

volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io
deletionPolicy: Delete (1)
```

(1) 当删除卷快照时，如果设置了 **Delete** 值，则底层快照会与 **VolumeSnapshotContent** 对象一起删除。如果设置了 **Retain** 值，则基本快照和 **VolumeSnapshotContent** 对象仍保留。如果设置了 **Retain** 值，且在不删除对应的 **VolumeSnapshotContent** 对象的情况下删除了 **VolumeSnapshot** 对象，则内容会保留。快照本身也保留在存储后端中。

2. 输入以下命令删除卷快照：

```
$ oc delete volumesnapshot <volumesnapshot_name>
```

输出示例

```
volumesnapshot.snapshot.storage.k8s.io "mysnapshot" deleted
```

3. 如果删除策略被设置为 **Retain**，请输入以下命令删除卷快照内容：

```
$ oc delete volumesnapshotcontent <volumesnapshotcontent_name>
```

4. 可选：如果 **VolumeSnapshot** 对象没有成功删除，请输入以下命令删除左侧资源的所有终结程序，以便删除操作可以继续进行：

重要

如果您确信不存在来自持久性卷声明或卷快照内容到 **VolumeSnapshot** 对象的引用时，才删除终结器。即使使用了 **--force** 选项，在删除所有终结器前，删除操作也不会删除快照对象。

```
$ oc patch -n $PROJECT volumesnapshot/$NAME --type=merge -p '{"metadata":
{"finalizers":null}}'
```

输**出示例**

```
volumesnapshotclass.snapshot.storage.k8s.io "csi-ocs-rbd-snapclass" deleted
```

删除终结器并删除卷快照。

5.4.7. 恢复卷快照

VolumeSnapshot CRD 内容可用于将现有卷恢复到以前的状态。

绑定 **VolumeSnapshot** CRD 并将 **readyToUse** 值设置为 **true** 后，您可以使用该资源置备一个预先填充快照数据的新卷。

先决条件

- 登陆到一个正在运行的 OpenShift Container Platform 集群。
- 使用支持卷快照的容器存储接口 (CSI) 驱动程序创建的持久性卷声明 (PVC) 。
- 用于置备存储后端的存储类。
- 卷快照已创建并可使用。

流程

1. 在 PVC 上指定 **VolumeSnapshot** 数据源，如下所示：

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. **VolumeSnapshot** 对象的名称，代表作为源的快照。
2. 必须设置为 **VolumeSnapshot** 的值。
3. 必须设置为 **snapshot.storage.k8s.io** 的值 。

2. 运行以下命令来创建一个 PVC：

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

3. 运行以下命令验证恢复的 PVC 是否已创建：

```
$ oc get pvc
```

此时会显示一个新的 PVC，如 **myclaim-restore**。

5.5. CSI 卷克隆

卷克隆会复制现有的持久性卷，以帮助防止 OpenShift Container Platform 中的数据丢失。此功能仅可用于受支持的 Container Storage Interface (CSI) 驱动程序。在置备 CSI 卷克隆前，您应该先熟悉持久性卷。

5.5.1. CSI 卷克隆概述

容器存储接口 (CSI) 卷克隆代表着在特定时间点上，一个已存在的持久性卷的副本。

卷克隆与卷快照类似，但效率更高。例如，集群管理员可以通过创建现有集群卷的另一个实例来复制集群卷。

克隆会在后端设备上创建指定卷的副本，而不是创建一个新的空卷。在进行动态置备后，您可以像使用任何标准卷一样使用卷克隆。

克隆不需要新的 API 对象。**PersistentVolumeClaim** 对象中现有的 **dataSource** 项应该可以接受同一命名空间中的一个已存在的 PersistentVolumeClaim。

5.5.1.1. 支持限制

在默认情况下，OpenShift Container Platform 支持 CSI 卷克隆，但有以下限制：

- 目标持久性卷声明 (PVC) 必须与源 PVC 位于同一个命名空间中。
- 不同的存储类支持克隆。
  - 对于与源不同的存储类，目标卷可能相同。
  - 您可以使用默认存储类，在 **spec** 中省略 **storageClassName**。
- 仅支持 CSI 驱动程序。不支持 in-tree 和 FlexVolumes。
- 特定的 CSI 驱动程序可能会还没有实现卷克隆功能。详情请查看 CSI 驱动程序文档。

5.5.2. 置备 CSI 卷克隆

创建一个克隆的持久性卷声明 (PVC) API 对象时，会触发一个 CSI 卷克隆的置备。克隆会预先获得另一个 PVC 的内容，遵循与任何其他持久性卷相同的规则。其中一个例外是，您必须添加一个指代到同一命名空间中现有 PVC 的 **dataSource**。

先决条件

- 登陆到一个正在运行的 OpenShift Container Platform 集群。

- 使用支持卷克隆的 CSI 驱动程序创建的 PVC。
- 为动态置备配置了存储后端。静态置备程序不支持克隆。

流程

从现有 PVC 克隆 PVC：

1. 使用以下 YAML 描述的 **PersistentVolumeClaim** 对象创建并保存一个文件：

   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 置备存储后端的存储类的名称。可以使用默认存储类，**storageClassName** 在 spec 中可以忽略。

2. 运行以下命令，创建上一步中保存的对象：

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

   一个新的 PVC **pvc-1-clone** 被创建。

3. 运行以下命令验证卷克隆是否已创建并就绪：

```
$ oc get pvc pvc-1-clone
```

   pvc-1-clone 显示的状态为 **Bound**。

   现在，您已准备好使用新克隆的 PVC 来配置 pod。

4. 使用 YAML 描述的 **Pod** 对象创建并保存文件。例如：

```
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 在 CSI 卷克隆操作中创建的克隆 PVC。

创建的 **Pod** 对象现在可以使用、克隆、快照或删除独立于它的原始 **dataSource** 的克隆 PVC。

5.6. 管理默认存储类

5.6.1. 概述

管理默认存储类可让您完成几个不同的目标：

- 禁用动态置备来强制静态置备。
- 当您有其他首选存储类时，防止存储操作器重新创建初始默认存储类。
- 重命名或更改默认存储类

要实现这些目标，您可以更改 **ClusterCSIDriver** 对象中的 **spec.storageClassState** 字段的设置。此字段可能的设置有：

- **Managed**：（默认）Container Storage Interface (CSI) Operator 会主动管理其默认存储类，集群管理员对默认存储类进行的大多数手动更改会被删除，如果您试图手动默认存储类，则它们会被持续重新创建。
- **Unmanaged** ：您可以修改默认存储类。CSI Operator 不会主动管理存储类，因此它不会协调它自动创建的默认存储类。
- **Removed**: CSI operator 删除默认存储类。

以下 Container Storage Interface (CSI) 驱动程序操作器支持管理默认存储类：

- 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. 使用 Web 控制台管理默认存储类

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 使用 cluster-admin 权限访问集群。

流程

使用 Web 控制台管理默认存储类：

1. 登录到 web 控制台。
2. 点 **Administration** > **CustomResourceDefinitions**。
3. 在 **CustomResourceDefinitions** 页面中，键入 **clustercsidriver** 来查找 **ClusterCSIDriver** 对象。
4. 点 **ClusterCSIDriver**，然后点 **Instances** 选项卡。
5. 点所需实例的名称，然后点 **YAML** 选项卡。
6. 添加 **spec.storageClassState** 字段，值为 **Managed**、**Unmanaged** 或 **Removed**。

   示例

```
...
spec:
  driverConfig:
    driverType: ''
  logLevel: Normal
  managementState: Managed
  observedConfig: null
  operatorLogLevel: Normal
  storageClassState: Unmanaged (1)
...
```

   (1) **spec.storageClassState** 字段设置为 "Unmanaged"

7. 点击 **Save**。

5.6.3. 使用 CLI 管理默认存储类

先决条件

- 使用 cluster-admin 权限访问集群。

流程

要使用 CLI 管理存储类，请运行以下命令：

```
oc patch clustercsidriver $DRIVERNAME --type=merge -p "{\"spec\":
{\"storageClassState\":\"${STATE}\"}}" (1)
```

(1) 其中 **${STATE}** 为 "Removed" 或 "Managed" 或 "Unmanaged"。

其中 **$DRIVERNAME** 是置备程序名称。您可以通过运行命令 **oc get sc** 来查找置备程序名称。

5.6.4. 缺少或多个默认存储类

5.6.4.1. 多个默认存储类

如果您将非默认存储类标记为默认存储类，且没有取消设置现有的默认存储类，或者在默认存储类已存在时创建默认存储类，则可能会出现多个默认存储类。当存在多个默认存储类时，任何请求默认存储类 (**pvc.spec.storageClassName**=nil) 的持久性卷声明 (PVC) 都会获得最近创建的默认存储类，无论该存储类的默认存储类是什么，管理员都会在警报仪表板中收到警报，该类有多个默认存储类，**MultipleDefaultStorageClasses**。

5.6.4.2. 缺少默认存储类

PVC 可能会尝试使用不存在的默认存储类：

- 管理员删除默认存储类或将其标记为非默认，然后用户会创建一个请求默认存储类的 PVC。
- 在安装过程中，安装程序会创建一个请求默认存储类的 PVC，该类尚未创建。

在前面的场景中，PVC 会无限期处于 pending 状态。要解决这种情况，请创建一个默认存储类，或声明其中一个现有存储类作为默认值。创建或声明默认存储类后，PVC 就会获取新的默认存储类。如果可能，PVC 最终会绑定到静态或动态置备的 PV，并移出待处理状态。

5.6.5. 更改默认存储类

使用以下步骤更改默认存储类。

例如，您有两个定义的存储类 **gp3** 和 **standard**，您想要将默认存储类从 **gp3** 改为 **standard**。

先决条件

- 使用 cluster-admin 权限访问集群。

流程

更改默认存储类：

1. 列出存储类：

```
$ oc get storageclass
```

输出示例

```
NAME            TYPE
gp3 (default)   kubernetes.io/aws-ebs (1)
standard        kubernetes.io/aws-ebs
```

(1) **(default)** 表示默认存储类。

2. 将所需的存储类设为默认存储类。

对于所需的存储类，运行以下命令将 **storageclass.kubernetes.io/is-default-class** 注解设置为 **true** ：

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

注意

您可以短时间内有多个默认存储类。但是，您应该确保最终只有一个默认存储类。

当存在多个默认存储类时，任何请求默认存储类 (**pvc.spec.storageClassName**=nil) 的持久性卷声明 (PVC) 都会获得最近创建的默认存储类，无论该存储类的默认存储类是什么，管理员都会在警报仪表板中收到警报，该类有多个默认存储类，**MultipleDefaultStorageClasses**。

3. 从旧的默认存储类中删除默认存储类设置。
   对于旧的默认存储类，运行以下命令将 **storageclass.kubernetes.io/is-default-class** 注解的值改为 **false** ：

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-
default-class": "false"}}}'
```

4. 确认更改：

```
$ oc get storageclass
```

输**出示例**

```
NAME                TYPE
gp3                 kubernetes.io/aws-ebs
standard (default)  kubernetes.io/aws-ebs
```

5.7. CSI 自动迁移

通常 OpenShift Container Platform 附带的树内存储驱动程序已弃用，并被对应的 Container Storage Interface(CSI)驱动程序替代。OpenShift Container Platform 为树内卷插件提供自动迁移到对应的 CSI 驱动程序。

5.7.1. 概述

此功能会自动将使用树内存储插件置备的卷迁移到对应的 Container Storage Interface (CSI) 驱动程序中。

此过程不执行任何数据迁移，OpenShift Container Platform 只在内存中转换持久性卷对象。因此，翻译的持久性卷对象不会存储在磁盘上，也不会更改它的内容。CSI 自动迁移应该可以无缝进行。这个功能不会改变如何使用所有现有 API 对象： 例如 **PersistentVolume**、**PersistentVolume**、PersistentVolume**VolumeClaims** 和 **StorageClasses**。

以下到 CSI 驱动程序的树内会自动迁移：

- Azure Disk
- OpenStack Cinder

- Amazon Web Services（AWS）Elastic Block Storage（EBS）
- Google Compute Engine Persistent Disk (GCP PD)
- Azure File
- VMware vSphere

这些卷类型的 CSI 迁移被视为正式发布(GA)，且无需人工干预。

如果原始 in-tree 存储插件不支持，则 CSI 自动迁移不会启用任何新的 CSI 驱动程序功能，如快照或扩展。

5.7.2. 存储类影响

对于新的 OpenShift Container Platform 4.13 及之后的版本，安装默认存储类是 CSI 存储类。所有使用这个存储类置备的卷都是 CSI 持久性卷(PV)。

对于从 4.12 版本升级到 4.13 及更高版本中的集群，会创建 CSI 存储类，如果在升级前设置了默认存储类，则设置为默认值。在不太可能的情况下，存在相同名称的存储类时，现有的存储类不会改变。任何现有的 in-tree 存储类保留，可能需要某些功能，如卷扩展才能用于现有的树内 PV。虽然引用 in-tree 存储插件的存储类将继续工作，但我们建议将默认存储类切换到 CSI 存储类。

要更改默认存储类，请参阅更改默认存储类。

5.8. 在非正常节点关闭后分离 CSI 卷

此功能允许容器存储接口 (CSI) 驱动程序在节点非正常关闭时自动分离卷。

5.8.1. 概述

当 kubelet 的节点关闭管理器检测到后续的节点关闭操作时，会出现一个安全的节点关闭。当 kubelet 不会检测到节点关闭操作时，会发生非正常关闭操作，这可能会因为系统或硬件故障而发生这种情况。另外，当 shutdown 命令没有触发 kubelet on Linux 使用的 Inhibitor Locks 机制时，kubelet 可能无法检测到节点关闭操作，例如，如果为该节点没有正确配置 shutdownGracePeriod 和 shutdownGracePeriodCriticalPods 详情。

通过此功能，当发生非正常节点关闭时，您可以手动在节点上添加 **out-of-service** 污点，以允许卷自动从节点分离。

5.8.2. 手动为自动卷分离添加 out-of-service 污点

先决条件

- 使用 cluster-admin 权限访问集群。

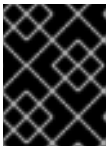
流程

允许卷在非正常节点关闭后自动从节点分离：

1. 节点检测到不健康后，关闭 worker 节点。
2. 运行以下命令检查状态，确保节点已关闭：

```
oc get node <node name> 1
```

1 <node name> = 非正常关闭节点的名称



重要

如果节点没有完全关闭，请不要继续污点该节点。如果节点仍处于 up 状态并且应用了污点，则可能会出现文件系统崩溃。

3. 运行以下命令来污点对应的节点对象：

```
oc adm taint node <node name> node.kubernetes.io/out-of-service=nodeshutdown:NoExecute 1
```

1 <node name> = 非正常关闭节点的名称

应用污点后，卷将从关闭节点分离，允许将其磁盘附加到其他节点。

Example

生成的 YAML 文件类似如下：

```
spec:
  taints:
  - effect: NoExecute
    key: node.kubernetes.io/out-of-service
    value: nodeshutdown
```

4. 重启该节点。

5. 删除污点。

5.9. ALICLOUD DISK CSI DRIVER OPERATOR

5.9.1. 概述

OpenShift Container Platform 可以使用 Alibaba AliCloud Disk Storage 的 Container Storage Interface(CSI)驱动程序置备持久性卷(PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 AliCloud Disk 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中安装 AliCloud Disk CSI Driver Operator 和 AliCloud Disk CSI 驱动程序。

- *AliCloud Disk CSI Driver Operator* 提供了一个存储类(**licloud-disk**)，您可以使用它来创建持久性卷声明(PVC)。AliCloud Disk CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。

- *AliCloud Disk CSI 驱动程序* 允许您创建并挂载 AliCloud Disk PV。

5.9.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

其他资源

- 配置 CSI 卷

5.10. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR

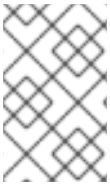
5.10.1. 概述

OpenShift Container Platform 可以使用 AWS EBS CSI 驱动程序置备持久性卷 (PV)。

在使用 Container Storage Interface (CSI) Operator 和驱动时，我们建议用户需要熟悉持久性存储和配置 CSI 卷。

要创建挂载到 AWS EBS 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 AWS EBS CSI Driver Operator (Red Hat operator)和 AWS EBS CSI 驱动程序。

- *AWS EBS CSI Driver Operator* 默认提供了一个 StorageClass，您可使用它来创建 PVC。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。您还可以选择创建 AWS EBS StorageClass，如使用 Amazon Elastic Block Store 的持久性存储 所述。
- *AWS EBS CSI 驱动程序*允许您创建并挂载 AWS EBS PV。



注意

如果您在 OpenShift Container Platform 4.5 集群上安装了 AWS EBS CSI Operator 和驱动程序，必须在升级到 OpenShift Container Platform 4.15 前卸载 4.5 Operator 和驱动程序。

5.10.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

重要

OpenShift Container Platform 默认使用 CSI 插件置备 Amazon Elastic Block Store (Amazon EBS) 存储。

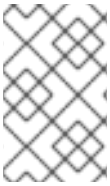
有关在 OpenShift Container Platform 中动态置备 AWS EBS 持久性卷的详情，请参阅使用 Amazon Elastic Block Store 的持久性存储。

5.10.3. 用户管理的加密

用户管理的加密功能允许您在安装过程中提供加密 OpenShift Container Platform 节点根卷的密钥，并允许所有受管存储类使用这些密钥加密置备的存储卷。您必须在 install-config YAML 文件中的 **platform.<cloud_type>.defaultMachinePlatform** 字段中指定自定义密钥。

此功能支持以下存储类型：

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk 存储
- Google Cloud Platform (GCP) 持久磁盘 (PD) 存储
- IBM Virtual Private Cloud (VPC) Block 存储



注意

如果没有在存储类中定义加密密钥，则只在存储类中设置 **encrypted: "true"**。AWS EBS CSI 驱动程序使用 AWS 受管别名/aws/ebs，这由 Amazon EBS 默认在每个区域中创建，以加密置备的存储卷。另外，受管存储类都具有 **encrypted: "true"** 设置。

有关使用用户管理的 Amazon EBS 加密安装的详情，请参考安装配置参数。

其他资源

- 使用 Amazon Elastic Block Store 的持久性存储
- 配置 CSI 卷

5.11. AWS ELASTIC FILE SERVICE CSI DRIVER OPERATOR

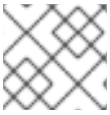
5.11.1. 概述

OpenShift Container Platform 可以使用 AWS Elastic File Service (EFS) 的 Container Storage Interface (CSI) 驱动程序置备持久性卷 (PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

安装 AWS EFS CSI Driver Operator 后，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 AWS EFS CSI Operator 和 AWS EFS CSI 驱动程序。这可让 AWS EFS CSI Driver Operator 创建挂载到 AWS EFS 资产中的 CSI 置备 PV。

- 安装之后，*AWS EFS CSI Driver Operator* 不会默认创建存储类来创建持久性卷声明 (PVC)。但是，您可以手动创建 AWS EFS **StorageClass**。AWS EFS CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷。这消除了集群管理员预置备存储的需求。
- *AWS EFS CSI 驱动程序* 允许您创建并挂载 AWS EFS PV。



注意

AWS EFS 只支持区域卷，不支持 zonal 卷。

5.11.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.11.3. 设置 AWS EFS CSI Driver Operator

1. 安装 AWS EFS CSI Driver Operator （一个红帽 operator）。
2. 如果您将 AWS EFS 与 AWS Secure Token Service (STS)搭配使用，请获取 STS 的角色 Amazon Resource Name (ARN)。安装 AWS EFS CSI Driver Operator 需要此项。
3. 安装 AWS EFS CSI Driver Operator。
4. 安装 AWS EFS CSI 驱动程序。

5.11.3.1. 获取安全令牌服务的角色 Amazon 资源名称

此流程解释了如何获取角色 Amazon 资源名称(ARN)，以使用 AWS 安全令牌服务(STS)上的 OpenShift Container Platform 配置 AWS EFS CSI Driver Operator。

重要

在安装 AWS EFS CSI Driver Operator 前执行这个步骤（请参阅 *安装 AWS EFS CSI Driver Operator* 流程）。

先决条件

- 使用具有 cluster-admin 角色的用户访问集群。
- AWS 帐户凭证

流程

您可以以多种方式获取 ARN 角色。以下流程演示了使用与集群安装相同的概念和 CCO 实用程序 (**ccoctl**) 二进制工具的方法。

获取使用 STS 配置 AWS EFS CSI Driver Operator 的角色 ARN：

1. 从 OpenShift Container Platform 发行镜像中提取 **ccoctl**，用于使用 STS 安装集群。如需更多信息，请参阅"配置 Cloud Credential Operator 工具程序"。
2. 创建并保存 EFS **CredentialsRequest** YAML 文件，如以下示例所示，然后将其放在 **credrequests** 目录中：

   示例

```
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. 运行 **ccoctl** 工具在 AWS 中生成新的 IAM 角色，并在本地文件系统中创建一个 YAML 文件 (**<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>** 是用于标记为跟踪而创建的云资源的名称。
- **region=<aws_region** > 是创建云资源的 AWS 区域。
- **dir=<path_to_directory_with_list_of_credentials_requests>/credrequests** 是包含上一步中 EFS CredentialsRequest 文件的目录。
- **<aws_account_id>** 是 AWS 帐户 ID。

示例

```
$ 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
```

输**出示例**

```
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. 从上一步中的*示例输出*的第一行中复制角色 ARN。角色 ARN 在 "Role" 和 "created" 之间。在本例中，角色 ARN 是 "arn:aws:iam::123456789012:role/my-aws-efs -openshift-cluster-csi-drivers-aws-efs-cloud"。
安装 AWS EFS CSI Driver Operator 时，您将需要角色 ARN。

后续步骤

安装 AWS EFS CSI Driver Operator 。

其他资源

- 安装 AWS EFS CSI Driver Operator
- 配置 Cloud Credential Operator 工具
- 安装 AWS EFS CSI 驱动程序

5.11.3.2. 安装 AWS EFS CSI Driver Operator

默认情况下，AWS EFS CSI Driver Operator （一个 Red Hat Operator）不会在 OpenShift Container Platform 中安装。使用以下步骤在集群中安装和配置 AWS EFS CSI Driver Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

从 web 控制台安装 AWS EFS CSI Driver Operator：

1. 登录到 web 控制台。
2. 安装 AWS EFS CSI Operator：
   a. 点 **Operators → OperatorHub**。
   b. 通过在过滤框中键入 AWS EFS CSI 来找到 **AWS EFS CSI** Operator。
   c. 点 **AWS EFS CSI Driver Operator** 按钮。

重要

确保选择 **AWS EFS CSI Driver Operator**，而不是 **AWS EFS Operator**。**AWS EFS Operator** 是一个社区 Operator，不受红帽支持。

   d. 在 **AWS EFS CSI Driver Operator** 页面中，点 **Install**。
   e. 在 **Install Operator** 页面中，确保：
      - 如果您使用 AWS Secure Token Service (STS) 的 AWS EFS，在角色 **ARN** 字段中输入从 *Obtaining a role Amazon Resource Name for Security Token Service* 过程的最后一步中复制的 ARN 角色。
      - 选择 **All namespaces on the cluster (default)**。
      - 安装的命名空间 被设置为 **openshift-cluster-csi-drivers**。
   f. 点 **Install**。
   安装完成后，AWS EFS CSI Operator 会在 web 控制台的 **Installed Operators** 部分列出。

后续步骤

安装 AWS EFS CSI 驱动程序。

5.11.3.3. 安装 AWS EFS CSI 驱动程序

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

1. 点 **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**。
2. 在 **Instances** 选项卡上，单击 **Create ClusterCSIDriver**。
3. 使用以下 YAML 文件：

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: efs.csi.aws.com
spec:
  managementState: Managed
```

4. 点 **Create**。
5. 等待以下 Conditions 更改为 "True" 状态：
   - AWSEFSDriverNodeServiceControllerAvailable
   - AWSEFSDriverControllerServiceControllerAvailable

5.11.4. 创建 AWS EFS 存储类

存储类用于区分和划分存储级别和使用。通过定义存储类，用户可以获得动态置备的持久性卷。

安装后，*AWS EFS CSI Driver Operator* （一个 Red Hat operator） 不会默认创建存储类。但是，您可以手动创建 AWS EFS 存储类。

5.11.4.1. 使用控制台创建 AWS EFS 存储类

流程

1. 在 OpenShift Container Platform 控制台中点 **Storage** → **StorageClasses**。
2. 在 **StorageClasses** 页面中，点 **Create StorageClass**。
3. 在 **StorageClass** 页面中，执行以下步骤：
   a. 输入一个名称来指代存储类。
   b. 可选：输入描述。
   c. 选择 reclaim 策略。
   d. 从 **Provisioner** 下拉列表中，选择 **efs.csi.aws.com**。
   e. 可选：为所选置备程序设置配置参数。
4. 点 **Create**。

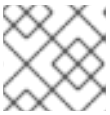
5.11.4.2. 使用 CLI 创建 AWS EFS 存储类

流程

- 创建 **StorageClass** 对象：

```
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** 必须是 **efs-ap** 才能启用动态置备。
2. **fileSystemId** 必须是手动创建的 EFS 卷的 ID。
3. **directoryPerms** 是卷的根目录的默认权限。在本例中，该卷只能被所有者访问。
4. 5. **gidRangeStart** 和 **gidRangeEnd** 设置用于设置 AWS 访问点 GID 的 POSIX 组 ID(GID)范围。如果未指定，则默认范围为 50000-7000000。每个置备的卷（即 AWS 访问点）都会被分配一个这个范围内的唯一 GID。
6. **BasePath** 是 EFS 卷上用于创建动态置备卷的目录。在这种情况下，PV 被置备为 EFS 卷上的 "/dynamic_provisioning/<random uuid>"。只有 子目录挂载到使用该 PV 的 pod。



注意

集群管理员可创建几个 **StorageClass** 对象，各自使用不同的 EFS 卷。

5.11.5. AWS EFS CSI 跨帐户支持

跨帐户支持允许您在一个 AWS 帐户中有一个 OpenShift Container Platform 集群，并使用 AWS Elastic File System (EFS) Container Storage Interface (CSI) 驱动程序在另一个 AWS 帐户中挂载文件系统。

注意

OpenShift Container Platform 集群和 EFS 文件系统必须位于同一区域。

先决条件

- 使用管理员权限访问 OpenShift Container Platform 集群
- 两个有效的 AWS 帐户

流程

以下流程演示了如何设置：

- AWS 帐户 A 中的 OpenShift Container Platform 集群
- 在帐户 B 中挂载 AWS EFS 文件系统

在帐户间使用 AWS EFS：

1. 使用 AWS 帐户 A 安装 OpenShift Container Platform 集群，并安装 EFS CSI Driver Operator。
2. 在 AWS 帐户 B 中创建 EFS 卷：
   a. 创建一个带有 CIDR 虚拟私有云 (VPC)（例如称为 "my-efs-vpc"），例如"172.20.0.0/16" 以及 AWS EFS 卷的子网。
   b. 在 AWS 控制台中，转至 https://console.aws.amazon.com/efs。
   c. 点 **Create new filesystem** ：
      i. 创建一个文件系统，如 "my-filesystem"。
      ii. 选择之前创建的 VPC ("my-efs-vpc")。
      iii. 接受其余设置的默认值。
   d. 确保已创建了卷和挂载目标：
      i. https://console.aws.amazon.com/efs#/file-systems.
      ii. 点您的卷，在 **Network** 选项卡中，等待所有 Mount Targets 可用（大约 1-2 分钟）。
   e. 在 **Network** 选项卡中，复制安全组 ID。下一步需要它。
3. 配置 AWS 帐户 B 上 AWS EFS 卷的网络访问：
   a. 进入 https://console.aws.amazon.com/ec2/v2/home#SecurityGroups。
   b. 通过过滤之前复制的组 ID 来查找 AWS EFS 卷使用的安全组。
   c. 在 **Inbound rules** 选项卡中，点 **Edit inbound rules**，然后添加新规则来允许 OpenShift Container Platform 节点访问 AWS EFS 卷（即，使用集群中的 NFS 端口）：
      - 类型 ：NFS
      - 协议 ：TCP
      - 端口范围 ：2049
      - 源 ：OpenShift Container Platform 集群节点的自定义/IP 地址范围（如 "10.0.0.0/16"）
   d. 保存规则。

注意

如果您遇到挂载问题，请重新检查端口号、IP 地址范围，并验证 AWS EFS 卷是否使用预期的安全组。

4. 在 AWS 帐户 A 的 OpenShift Container Platform 集群 VPC 和 AWS 帐户 B 中的 AWS EFS VPC 之间创建 VPC 对等：
确保两个 VPC 使用不同的网络 CIDR，并在创建 VPC 对等后，在每个 VPC 中添加路由来连接两个 VPC 网络。

   a. 在帐户 B 中创建一个名为 "my-efs-crossaccount-peering-connection" 的对等连接。对于本地 VPC ID，请使用 EFS-located VPC。要与 VPC 用于帐户 A 的对等，对于 VPC ID，请使用 OpenShift Container Platform 集群 VPC ID。

   b. 接受 AWS 帐户 A 中的对等连接。

   c. 修改 AWS 帐户 B 中每个子网的路由表 (EFS-volume 使用的子网)：

      i. 在左侧窗格中，在 **Virtual private cloud** 下，单向下箭头以展开可用选项。

      ii. 在 **Virtual private cloud** 下，点 **Route tables"**。

      iii. 点 **Routes** 选项卡。

      iv. 在 **Destination** 下，输入 10.0.0.0/16。

      v. 在 **Target** 下，使用来自创建的对等连接的对等连接类型点。

   d. 修改 AWS 帐户 A 中每个子网的路由表 (OpenShift Container Platform 集群节点使用的子网)：

      i. 在左侧窗格中，在 **Virtual private cloud** 下，单向下箭头以展开可用选项。

      ii. 在 **Virtual private cloud** 下，点 **Route tables"**。

      iii. 点 **Routes** 选项卡。

      iv. 在 **Destination** 下，在帐户 B 中输入 VPC 的 CIDR，本例中为 172.20.0.0/16。

      v. 在 **Target** 下，使用来自创建的对等连接的对等连接类型点。

5. 在 AWS 帐户 B 中创建一个 IAM 角色，如 "my-efs-acrossaccount-role"，它与 AWS 帐户 A 具有信任关系，并使用调用"my-efs-acrossaccount-driver-policy"的权限添加内联 AWS EFS 策略。
这个角色供在 AWS 帐户 A 的 OpenShift Container Platform 集群上运行的 CSI 驱动程序控制器服务决定 AWS 帐户 B 中文件系统的挂载目标。

```
# 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. 在 AWS 帐户 A 中，将内联策略附加到 AWS EFS CSI 驱动程序控制器服务帐户的 IAM 角色中，并具有在之前创建的 IAM 角色上执行安全令牌服务(STS)所需的权限。

```
# 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. 在 AWS 帐户 A 中，将 AWS 管理的策略 "AmazonElasticFileSystemClientFullAccess" 附加到 OpenShift Container Platform 集群 master 角色。角色名称的格式是 **<clusterID>-master-role** (例如，**my-0120ef-czjrl-master-role**)。

8. 创建一个 Kubernetes secret，使用 **awsRoleArn** 作为键，以及之前创建的角色作为值：

```
$ 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'
```

由于驱动程序控制器需要从 secret 获取跨帐户角色信息，您需要将 secret 角色绑定添加到 AWS EFS CSI 驱动程序控制器 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. 在帐户 B 中为文件系统 (AWS EFS 卷) 中创建一个 **filesystem** 策略，它允许 AWS 帐户 A 在其上执行挂载。

```
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. 使用类似如下的配置创建 AWS EFS 卷存储类：

```
# 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. 在 AWS 中创建和配置对 EFS 卷的访问

此流程解释了如何在 AWS 中创建和配置 EFS 卷，以便在 OpenShift Container Platform 中使用它们。

先决条件

- AWS 帐户凭证

流程

在 AWS 中创建和配置对 EFS 卷的访问：

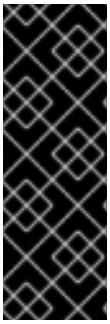
1. 在 AWS 控制台中，打开 https://console.aws.amazon.com/efs。
2. 点击 **Create** 文件系统 ：
   - 输入文件系统的名称。

- 对于 **Virtual Private Cloud (VPC)** 请选择 OpenShift Container Platform 的虚拟私有云 (VPC)。
- 接受所有其他选择的默认设置。

3. 等待卷和挂载目标完成完全创建：
   a. 访问 https://console.aws.amazon.com/efs#/file-systems。
   b. 单击您的卷，在 **Network** 选项卡中，等待所有挂载目标变为可用状态（约 1-2 分钟）。
4. 在 **Network** 选项卡上，复制安全组 ID（下一步中您将需要此 ID）。
5. 进入 https://console.aws.amazon.com/ec2/v2/home#SecurityGroups，并查找 EFS 卷使用的安全组。
6. 在 **Inbound rules** 选项卡中，点 **Edit inbound rules**，然后使用以下设置添加新规则，以允许 OpenShift Container Platform 节点访问 EFS 卷：
   - 类型 ：NFS
   - 协议 ：TCP
   - 端口范围 ：2049
   - 源 ：您的节点的自定义/IP 地址范围（例如："10.0.0.0/16"）
     此步骤允许 OpenShift Container Platform 使用集群中的 NFS 端口。
7. 保存规则。

5.11.7. Amazon Elastic File Storage 的动态置备

AWS EFS CSI 驱动程序支持不同于其他 CSI 驱动程序的动态置备形式。它将新 PV 调配为预先存在的 EFS 卷的子目录。PV 相互独立。但是，它们共享相同的 EFS 卷。删除卷时，置备的所有 PV 也会被删除。EFS CSI 驱动程序为每个此类子目录创建一个 AWS Access Point。由于 AWS AccessPoint 限制，您只能从一个 **StorageClass**/EFS 卷动态置备 1000 个 PV。



重要

请注意，EFS 不强制执行 **PVC.spec.resources**。

在以下示例中，您请求 5 GiB 的空间。但是，创建的 PV 是无限的，可以存储任何数量的数据（如 PB）。当在卷中存储太多数据时，一个被破坏的应用甚至恶意应用程序也可能会导致大量开支。

强烈建议在 AWS 中使用 EFS 卷大小监控。

先决条件

- 您已创建了 Amazon Elastic File Storage (Amazon EFS)卷。
- 您已创建了 AWS EFS 存储类。

流程

启用动态置备：

- 照常创建 PVC （或 StatefulSet 或 Template），引用之前创建的 **StorageClass**。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test
spec:
  storageClassName: efs-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

如果您在设置动态置备时遇到问题，请参阅 AWS EFS 故障排除。

其他资源

- 创建并配置对 AWS EFS 卷的访问
- 创建 AWS EFS 存储类

5.11.8. 使用 Amazon Elastic File Storage 创建静态 PV

可以使用 Amazon Elastic File Storage (Amazon EFS) 卷作为单个 PV，而无需动态置备。整个卷挂载到 pod。

先决条件

- 您已创建了 Amazon EFS 卷。

流程

- 使用以下 YAML 文件创建 PV：

```
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 2
    volumeAttributes:
      encryptInTransit: "false" 3
```

1. **spec.capacity** 没有任何含义，CSI 驱动程序会忽略它。只有在绑定到 PVC 时才会使用它。应用可以将任意数量的数据存储到卷中。

2. **volumeHandle** 必须与您在 AWS 中创建的 EFS 卷相同。如果您提供自己的访问点，**volumeHandle** 应为 **<EFS volume ID>::<access point ID>**。例如：**fs-6e633ada::fsap-081a1d293f0004630**。

3. 如果需要，您可以在传输中禁用加密。加密功能会被默认启用。

如果您在设置静态 PV 时遇到问题，请参阅 AWS EFS 故障排除。

5.11.9. Amazon Elastic File Storage 安全性

以下信息对于 Amazon Elastic File Storage (Amazon EFS) 安全非常重要。

例如，在使用接入点（例如，使用前面描述的动态置备）时，Amazon 会自动将文件的 GID 替换为接入点的 GID。此外，EFS 在评估文件系统权限时，会考虑访问点的用户 ID、组 ID 和次要组 ID。EFS 忽略 NFS 客户端的 ID。有关接入点的详情请参考 https://docs.aws.amazon.com/efs/latest/ug/efs-access-points.html。

因此，EFS 卷静默忽略 FSGroup；OpenShift Container Platform 无法将卷上的文件 GID 替换为 FSGroup。任何可以访问挂载的 EFS 接入点的 pod 都可以访问其中的任何文件。

与此无关，传输中的加密默认是启用的。如需更多信息，请参阅 https://docs.aws.amazon.com/efs/latest/ug/encryption-in-transit.html。

5.11.10. Amazon Elastic File Storage 故障排除

以下信息提供了有关如何对 Amazon Elastic File Storage (Amazon EFS) 问题进行故障排除的指导：

- AWS EFS Operator 和 CSI 驱动程序在命名空间 **openshift-cluster-csi-drivers** 中运行。
- 要启动收集 AWS EFS Operator 和 CSI 驱动程序的日志，请运行以下命令：

```
$ 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
```

- 要显示 AWS EFS Operator 错误，请查看 **ClusterCSIDriver** 状态：

```
$ oc get clustercsidriver efs.csi.aws.com -o yaml
```

- 如果卷无法挂载到容器集（如下命令的输出中所示）：

```
$ 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 指示卷未挂载的警告消息。

此错误通常是由 AWS 在 OpenShift Container Platform 节点和 Amazon EFS 之间丢弃数据包造成的。

检查以下内容是否正确：

- AWS 防火墙和安全组
- 网络：端口号和 IP 地址

5.11.11. 卸载 AWS EFS CSI Driver Operator

在卸载 AWS EFS CSI Driver Operator （红帽操作器）后，无法访问所有 EFS PV。

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

从 web 控制台卸载 AWS EFS CSI Driver Operator：

1. 登录到 web 控制台。
2. 停止所有使用 AWS EFS PV 的应用程序。
3. 删除所有 AWS EFS PV：
   a. 点 **Storage → PersistentVolumeClaims**。
   b. 选择 AWS EFS CSI Driver Operator 使用的每个 PVC，点击 PVC 最右侧的下拉菜单，然后点 **Delete PersistentVolumeClaims**。
4. 卸载 AWS EFS CSI 驱动程序 ：

注意

在卸载 Operator 前，必须先删除 CSI 驱动程序。

   a. 点 **Administration → CustomResourceDefinitions → ClusterCSIDriver**。
   b. 在 **Instances** 选项卡上，单击左侧的 **efs.csi.aws.com**，单击下拉菜单，然后单击 **Delete ClusterCSIDriver**。

c. 出现提示时，单击 **Delete**。

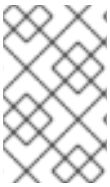
5. 卸载 AWS EFS CSI Operator：

a. 点 **Operators → Installed Operators**。

b. 在 **Installed Operators** 页面中，在 **Search by name** 框中输入 AWS EFS CSI 来查找 Operator，然后点击它。

c. 在 **Installed Operators > Operator** 详情页面的右上角，点 **Actions → Uninstall Operator**。

d. 当在 **Uninstall Operator** 窗口中提示时，点 **Uninstall** 按钮从命名空间中删除 Operator。Operator 在集群中部署的任何应用程序都需要手动清理。
卸载后，AWS EFS CSI Driver Operator 不会在 web 控制台的 **Installed Operators** 部分列出。



注意

在销毁集群 (**openshift-install destroy cluster**) 前，您必须删除 AWS 中的 EFS 卷。如果有使用集群的 VPC 的 EFS 卷，OpenShift Container Platform 集群将无法被销毁。Amazon 不允许删除这样的 VPC。

5.11.12. 其他资源

- 配置 CSI 卷

5.12. AZURE DISK CSI DRIVER OPERATOR

5.12.1. 概述

OpenShift Container Platform 可以使用 Microsoft Azure Disk Storage 的 Container Storage Interface（CSI）驱动程序置备持久性卷（PV）。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 Azure Disk 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 Azure Disk CSI Driver Operator 和 Azure Disk CSI 驱动程序。

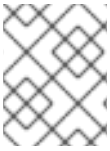
- *Azure Disk CSI Driver Operator* 提供了一个名为 **managed-csi** 的存储类，您可以使用它来创建持久性卷声明(PVC)。Azure Disk CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。

- *Azure Disk CSI 驱动程序* 允许您创建并挂载 Azure Disk PV。

5.12.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。



注意

OpenShift Container Platform 为 Azure Disk in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。如需更多信息，请参阅 CSI 自动迁移。

5.12.3. 创建带有存储帐户类型的存储类

存储类用于区分和划分存储级别和使用。通过定义存储类，用户可以获得动态置备的持久性卷。

在创建存储类时，您可以指定存储帐户类型。这与您的 Azure 存储帐户 SKU 层对应。有效选项包括 **Standard_LRS**,**Premium_LRS**,**StandardSSD_LRS**,**UltraSSD_LRS**,**Premium_ZRS**,**StandardSSD_ZRS**, 和 **PremiumV2_LRS**。有关查找 Azure SKU 层的详情，请参考 SKU 类型。

ZRS 和 PremiumV2_LRS 都有一些区域限制。有关这些限制的详情，请参阅 ZRS 限制和 Premium_LRS 限制。

先决条件

- 使用管理员权限访问 OpenShift Container Platform 集群

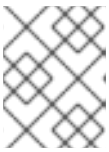
流程

使用以下步骤创建带有存储帐户类型的存储类。

1. 使用类似如下的 YAML 文件创建设计存储帐户类型的存储类：

```
$ 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. 存储类名称。
2. 存储帐户类型。这与您的 Azure 存储帐户 SKU 层对应：`Standard_LRS`, **Premium_LRS**, **StandardSSD_LRS**, **UltraSSD_LRS**, **Premium_ZRS**, **StandardSSD_ZRS**, **PremiumV2_LRS**。



注意

对于 PremiumV2_LRS，在 **storageclass.parameters** 中指定 **cachingMode: None**。

2. 通过列出存储类来确保创建了存储类：

```
$ oc get storageclass
```

输出示例

```
$ 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 带有存储帐户类型的新存储类。

5.12.4. 用户管理的加密

用户管理的加密功能允许您在安装过程中提供加密 OpenShift Container Platform 节点根卷的密钥，并允许所有受管存储类使用这些密钥加密置备的存储卷。您必须在 install-config YAML 文件中的 **platform.<cloud_type>.defaultMachinePlatform** 字段中指定自定义密钥。

此功能支持以下存储类型：

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk 存储
- Google Cloud Platform (GCP) 持久磁盘 (PD) 存储
- IBM Virtual Private Cloud (VPC) Block 存储

注意

如果 OS (root) 磁盘已被加密，且存储类中没有定义加密密钥，Azure Disk CSI 驱动程序默认使用 OS 磁盘加密密钥来加密置备的存储卷。

有关为 Azure 安装用户管理加密的详情，请参考为 Azure 启用用户管理的加密。

5.12.5. 用于部署带有使用 PVC 的巨型磁盘的机器的机器集

您可以创建在 Azure 上运行的机器集，该机器集用来部署带有巨型磁盘的机器。ultra 磁盘是高性能存储，用于要求最苛刻的数据工作负载。

in-tree 插件和 CSI 驱动程序都支持使用 PVC 启用巨型 磁盘。您还可以在不创建 PVC 的情况下将巨型磁盘部署为数据磁盘。

其他资源

- Microsoft Azure ultra 磁盘文档
- 使用树内 PVC 部署机器的机器集
- 在计算磁盘上部署机器的机器集作为数据磁盘

5.12.5.1. 使用机器集创建带有巨型磁盘的机器

您可以通过编辑机器集 YAML 文件在 Azure 上部署带有巨型磁盘的机器。

先决条件

- 已有 Microsoft Azure 集群。

流程

1. 运行以下命令，复制现有的 Azure **MachineSet** 自定义资源(CR)并编辑它：

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

其中 **<machine-set-name>** 是您要使用巨型磁盘置备机器的机器集。

2. 在指示的位置中添加以下行：

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

1 指定标签，用于选择此机器集创建的节点。此流程使用 **disk.ulssd** 用于这个值。

2 这些行支持使用 ultra 磁盘。

3. 运行以下命令，使用更新的配置创建机器集：

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

4. 创建一个包含以下 YAML 定义的存储类：

```
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. 指定存储类的名称。此流程使用 **ultra-disk-sc** 作为这个值。
2. 指定存储类的 IOPS 数量。
3. 指定存储类的吞吐量，单位为 MBps。
4. 对于 Azure Kubernetes Service(AKS)版本 1.21 或更高版本，请使用 **disk.csi.azure.com**。对于 AKS 的早期版本，请使用 **kubernetes.io/azure-disk**。
5. 可选：指定此参数以等待创建使用磁盘的 pod。

5. 创建一个持久性卷声明(PVC)来引用包含以下 YAML 定义的 **ultra-disk-sc** 存储类：

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

1. 指定 PVC 的名称。此流程使用 **ultra-disk** 作为这个值。
2. 此 PVC 引用了 **ultra-disk-sc** 存储类。
3. 指定存储类的大小。最小值为 **4Gi**。

6. 创建包含以下 YAML 定义的 pod：

```
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. 指定启用巨型磁盘的机器集标签。此流程使用 **disk.ulssd** 用于这个值。
2. 这个 pod 引用了 **ultra-disk** PVC。

验证

1. 运行以下命令验证机器是否已创建：

```
$ oc get machines
```

机器应处于 **Running** 状态。

2. 对于正在运行并附加节点的机器，请运行以下命令验证分区：

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

在这个命令中，**oc debug node/<node-name>** 会在节点 **<node-name>** 上启动一个 debugging shell，并传递一个带有 **--** 的命令。传递的命令 **chroot /host** 提供对底层主机操作系统二进制文件的访问，**lsblk** 显示连接至主机操作系统计算机的块设备。

后续步骤

- 要在 pod 中使用大量磁盘，请创建使用挂载点的工作负载。创建一个类似以下示例的 YAML 文件：

```
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. 启用 ultra 磁盘的机器集的故障排除资源

使用本节中的信息从您可能会遇到的问题了解和恢复。

5.12.5.2.1. 无法挂载由巨型磁盘支持的持久性卷声明

如果挂载了被巨型磁盘支持的持久性卷声明的问题，pod 会一直处于 **ContainerCreating** 状态，并触发警报。

例如，如果没有在支持托管 pod 的节点的机器上设置 **additionalCapabilities.ultraSSDEnabled** 参数，则会出现以下出错信息：

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

- 要解决这个问题，请运行以下命令来描述 pod：

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

5.12.6. 其他资源

- 使用 Azure Disk 的持久性存储
- 配置 CSI 卷

5.13. AZURE FILE CSI DRIVER OPERATOR

5.13.1. 概述

OpenShift Container Platform 可以使用 Microsoft Azure File Storage 的 Container Storage Interface(CSI)驱动程序置备持久性卷(PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 Azure File 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 Azure File CSI Driver Operator 和 Azure File CSI 驱动程序。

- *Azure File CSI Driver Operator* 提供了一个名为 **azurefile-csi** 的存储类，您可以使用它来创建持久性卷声明(PVC)。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。
- *Azure File CSI 驱动程序* 允许您创建并挂载 Azure File PV。Azure File CSI 驱动程序支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。

Azure File CSI Driver Operator 不支持：

- 虚拟硬盘(VHD)
- 在启用了联邦信息处理标准(FIPS)模式的节点上运行，用于服务器消息块(SMB)文件共享。但是，网络文件系统(NFS)支持 FIPS 模式。

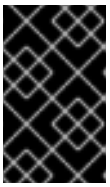
有关支持的功能的更多信息，请参阅支持的 CSI 驱动程序和功能。

5.13.2. NFS 支持

OpenShift Container Platform 4.14 及更新的版本支持带有 Network File System (NFS) 的 Azure File Container Storage Interface (CSI) Driver Operator，请考虑以下事项：

- 使用调度到 control plane 节点的 Azure File NFS 卷创建 pod 会导致挂载被拒绝。
  要临时解决这个问题：如果您的 control plane 节点可以调度，pod 可以在 worker 节点上运行，使用 **nodeSelector** 或 Affinity 将 pod 调度到 worker 节点上。

- FS 组策略行为：



重要

带有 NFS 的 Azure File CSI 不遵循 pod 请求的 fsGroupChangePolicy。带有 NFS 的 Azure File CSI 会应用默认的 OnRootMismatch FS Group 策略，无论 pod 请求的策略是什么。

- Azure File CSI Operator 不会自动为 NFS 创建存储类。您必须手动创建它。使用类似如下的文件：

```
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. 存储类名称。
2. 指定 Azure File CSI 供应商。
3. 指定 NFS 作为存储后端协议。

5.13.3. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

其他资源

- 使用 Azure File 的持久性存储
- 配置 CSI 卷

5.14. AZURE STACK HUB CSI DRIVER OPERATOR

5.14.1. 概述

OpenShift Container Platform 可以使用 Azure Stack Hub 存储的 Container Storage Interface (CSI) 驱动程序置备持久性卷 (PV)。Azure Stack Hub 是 Azure Stack 产品组合的一部分，允许您在内部环境中运行应用程序，并在数据中心内提供 Azure 服务。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 Azure Stack Hub 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 Azure Stack Hub CSI Driver Operator 和 Azure Stack Hub CSI 驱动程序。

- *Azure Stack Hub CSI Driver Operator* 提供了一个存储类 (**managed-csi**)，并将 "Standard_LRS" 用作默认存储帐户类型，您可以使用它来创建持久性卷声明 (PVC)。Azure Stack Hub CSI Driver Operator 通过允许按需创建存储卷来支持动态卷置备，不再需要集群管理员预置备存储。
- *Azure Stack Hub CSI 驱动程序*允许您创建并挂载 Azure Stack Hub PV。

5.14.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.14.3. 其他资源

- 配置 CSI 卷

5.15. GCP PD CSI DRIVER OPERATOR

5.15.1. 概述

OpenShift Container Platform 可以使用 Google Cloud Platform（GCP）持久性存储的 Container Storage Interface（CSI）驱动程序来置备持久性卷（PV）。

在使用 Container Storage Interface (CSI) Operator 和驱动时，我们建议用户需要熟悉持久性存储和配置 CSI 卷。

要创建挂载到 GCP PD 存储资产中的 CSI 置备持久性卷（PV），OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 GCP PD CSI Driver Operator 和 GCP PD CSI 驱动程序。

- **GCP PD CSI Driver Operator**：默认情况下，Operator 提供了一个可用来创建 PVC 的存储类。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。您还可以选择创建 GCP PD 存储类，如使用 GCE Persistent Disk 的 Persistent Storage 所述。
- **GCP PD** 驱动程序：该驱动程序可让您创建并挂载 GCP PD PV。

注意

OpenShift Container Platform 为 GCE Persistent Disk in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。如需更多信息，请参阅 CSI 自动迁移。

5.15.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.15.3. GCP PD CSI 驱动程序存储类参数

Google Cloud Platform（GCP）持久磁盘（PD）Container Storage Interface（CSI）驱动程序使用 CSI **external-provisioner** sidecar 作为控制器。这是和 CSI 驱动程序一起部署的单独的 helper 容器。sidecar 通过触发 **CreateVolume** 操作来管理持久性卷（PV）。

GCP PD CSI 驱动程序使用 **csi.storage.k8s.io/fstype** 参数键来支持动态置备。下表描述了 OpenShift Container Platform 支持的所有 GCP PD CSI 存储类参数。

表 5.5. CreateVolume 参数

参数	值	默认	描述
type	**pd-ssd** 或者 **pd-standard**	**pd-standard**	允许您选择标准的 PV 或使用固态硬盘的 PV。
replication-type	**none** 或 **regional-pd**	**none**	允许您在 zonal 或区域 PV 之间进行选择。
disk-encryption-kms-key	用于加密新磁盘的密钥的完全限定资源标识符。	空字符串	使用客户管理的加密密钥（CMEK）加密新磁盘。

5.15.4. 创建自定义加密的持久性卷

创建 **PersistentVolumeClaim** 对象时，OpenShift Container Platform 会置备一个新的持久性卷（PV）并创建一个 **PersistentVolume** 对象。您可以通过加密新创建的 PV，在 Google Cloud Platform（GCP）中添加自定义加密密钥来保护集群中的 PV。

要加密，您新创建的 PV 使用新的或现有的 Google Cloud Key Management Service（KMS）密钥在集群中使用用户管理的加密密钥（CMEK）。

先决条件

- 登陆到一个正在运行的 OpenShift Container Platform 集群。
- 您已创建了 Cloud KMS 密钥环以及密钥版本。

有关 CMEK 和 Cloud KMS 资源的更多信息，请参阅使用客户管理的加密密钥（CMEK）。

流程

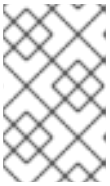
要创建自定义加密 PV，请完成以下步骤：

1. 使用 Cloud KMS 密钥创建存储类。以下示例启用加密卷的动态置备：

```
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 此字段必须是用于加密新磁盘的密钥的资源标识符。值是区分大小写的。有关提供关键 ID 值的更多信息，请参阅检索资源 ID 和 获取 Cloud KMS 资源 ID。



注意

您不能将 **disk-encryption-kms-key** 参数添加到现有的存储类中。但是，您可以删除存储类并使用相同的名称和不同的参数集合重新创建该存储类。如果您这样做，现有类的置备程序必须是 **pd.csi.storage.gke.io**。

2. 使用 **oc** 命令在 OpenShift Container Platform 集群上部署存储类：

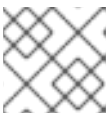
```
$ oc describe storageclass csi-gce-pd-cmek
```

输**出示例**

```
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. 创建名为 **pvc.yaml** 的文件，该文件与您在上一步中创建的存储类对象的名称匹配：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: podpvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-gce-pd-cmek
  resources:
    requests:
      storage: 6Gi
```



注意

如果将新存储类标记为默认值，可以省略 **storageClassName** 字段。

4. 在集群中应用 PVC:

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

5. 获取 PVC 的状态，并验证它是否已创建并绑定到新置备的 PV:

```
$ oc get pvc
```

输**出示例**

```
NAME     STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
podpvc   Bound    pvc-e36abf50-84f3-11e8-8538-42010a800002   10Gi       RWO            csi-
gce-pd-cmek  9s
```

注意

如果您的存储类将 **volumeBindingMode** 字段设置为 **WaitForFirstConsumer**，您必须创建一个 pod 来使用 PVC，然后才能验证它。

您的 CMEK 保护 PV 现在可以与 OpenShift Container Platform 集群一起使用。

5.15.5. 用户管理的加密

用户管理的加密功能允许您在安装过程中提供加密 OpenShift Container Platform 节点根卷的密钥，并允许所有受管存储类使用这些密钥加密置备的存储卷。您必须在 install-config YAML 文件中的 **platform.<cloud_type>.defaultMachinePlatform** 字段中指定自定义密钥。

此功能支持以下存储类型：

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk 存储
- Google Cloud Platform (GCP) 持久磁盘 (PD) 存储
- IBM Virtual Private Cloud (VPC) Block 存储

有关使用用户管理的 GCP PD 加密安装的详情，请参考安装配置参数。

5.15.6. 其他资源

- 使用 GCE Persistent Disk 的持久性存储
- 配置 CSI 卷

5.16. GOOGLE COMPUTE PLATFORM FILESTORE CSI DRIVER OPERATOR

5.16.1. 概述

OpenShift Container Platform 可以使用 Google Compute Platform (GCP) 文件存储存储的 Container Storage Interface (CSI) 驱动程序置备持久性卷 (PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 GCP Filestore Storage 资产中的 CSI 置备 PV，您可以在 **openshift-cluster-csi-drivers** 命名空间中安装 GCP Filestore CSI Driver Operator 和 GCP Filestore CSI 驱动程序。

- *GCP Filestore CSI Driver Operator* 默认不提供存储类，但如果需要可以创建一个。GCP Filestore CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。
- *GCP Filestore CSI 驱动程序*允许您创建并挂载 GCP Filestore PV。

5.16.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.16.3. 安装 GCP Filestore CSI Driver Operator

默认情况下，Google Compute Platform (GCP) Filestore Container Storage Interface (CSI) Driver Operator 不会在 OpenShift Container Platform 中安装。使用以下步骤在集群中安装 GCP Filestore CSI Driver Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

从 web 控制台安装 GCP Filestore CSI Driver Operator：

1. 登录到 web 控制台。
2. 运行以下命令，在 GCE 项目中启用 Filestore API：

```
$ gcloud services enable file.googleapis.com  --project <my_gce_project> 1
```

1 将 **<my_gce_project>** 替换为您的 Google Cloud 项目。

   您还可以使用 Google Cloud web 控制台进行此操作。

3. 安装 GCP Filestore CSI Operator：
   a. 点 **Operators → OperatorHub**。
   b. 通过在过滤器框中输入 GCP Filestore 来查找 **GCP Filestore** CSI Operator。
   c. 点 **GCP Filestore CSI Driver Operator** 按钮。
   d. 在 **GCP Filestore CSI Driver Operator** 页面中，点 **Install**。
   e. 在 **Install Operator** 页面中，确保：
      - 选择 **All namespaces on the cluster (default)**。
      - 安装的命名空间 被设置为 **openshift-cluster-csi-drivers**。

f. 点 **Install**。
安装完成后，GCP Filestore CSI Operator 会在 web 控制台的 **Installed Operators** 部分列出。

4. 安装 GCP Filestore CSI 驱动程序：

a. 点 **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**。

b. 在 **Instances** 选项卡上，单击 **Create ClusterCSIDriver**。
使用以下 YAML 文件：

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: filestore.csi.storage.gke.io
spec:
  managementState: Managed
```

c. 点 **Create**。

d. 等待以下 Conditions 变为 "true" 状态：

- GCPFilestoreDriverCredentialsRequestControllerAvailable
- GCPFilestoreDriverNodeServiceControllerAvailable
- GCPFilestoreDriverControllerServiceControllerAvailable

其他资源

- 在 Google Cloud 中启用 API。
- 使用 Google Cloud Web 控制台启用 API。

5.16.4. 为 GCP Filestore 存储创建存储类

安装 Operator 后，您应该创建一个存储类来动态置备 Google Compute Platform (GCP) 文件存储卷。

先决条件

- 已登陆到正在运行的 OpenShift Container Platform 集群。

流程

创建存储类：

1. 使用以下 YAML 文件示例创建存储类：

YAML 文件示例

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: filestore-csi
provisioner: filestore.csi.storage.gke.io
parameters:
```

```
    connect-mode: DIRECT_PEERING 1
    network: network-name 2
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

1. 对于共享 VPC，使用 **connect-mode** 参数设置为 **PRIVATE_SERVICE_ACCESS**。对于非共享 VPC，值是 **DIRECT_PEERING**，这是默认设置。
2. 指定应该在其中创建 Filestore 实例的 GCP 虚拟私有云(VPC) 网络的名称。

2. 指定在其中创建 Filestore 实例的 VPC 网络的名称。
   建议您指定应在其中创建 Filestore 实例的 VPC 网络。如果没有指定 VPC 网络，Container Storage Interface (CSI) 驱动程序会尝试在项目的默认 VPC 网络中创建实例。

   在 IPI 安装中，VPC 网络名称通常是带有后缀 "-network" 的集群名称。但是，在 UPI 安装中，VPC 网络名称可以是用户选择的任何值。

   对于一个共享的 VPC (**connect-mode** = **PRIVATE_SERVICE_ACCESS**)，网络需要是一个完整的 VPC 名。例如：**projects/shared-vpc-name/global/networks/gcp-filestore-network**。

   您可以使用以下命令检查 **MachineSets** 对象来查找 VPC 网络名称：

```
$ oc -n openshift-machine-api get machinesets -o yaml | grep "network:"
          - network: gcp-filestore-network
(...)
```

   在本例中，这个集群中的 VPC 网络名称为 "gcp-filestore-network"。

5.16.5. 销毁集群和 GCP 文件存储

通常，如果您销毁集群，OpenShift Container Platform 安装程序会删除属于该集群的所有云资源。但是，当销毁集群时，Google Compute Platform (GCP) Filestore 实例不会被自动删除，因此您必须在销毁集群前手动删除使用 Filestore 存储类的所有持久性卷声明 (PVC)。

流程

删除所有 GCP Filestore PVC：

1. 列出使用存储类 **filestore-csi** 创建的所有 PVC：

```
$ oc get pvc -o json -A | jq -r '.items[] | select(.spec.storageClassName == "filestore-csi")
```

2. 删除上一命令列出的所有 PVC：

```
$ oc delete <pvc-name> 1
```

1. 将 <pvc-name> 替换为您要删除的任何 PVC 的名称。

5.16.6. 其他资源

- 配置 CSI 卷

5.17. IBM VPC BLOCK CSI DRIVER OPERATOR

5.17.1. 概述

OpenShift Container Platform 可以使用 IBM® Virtual Private Cloud(VPC) Block Storage 的 Container Storage Interface(CSI)驱动程序置备持久性卷(PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 IBM® VPC Block 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 IBM® VPC Block CSI Driver Operator 和 IBM® VPC Block CSI 驱动程序。

- *IBM® VPC Block CSI Driver Operator* 针对不同的级别提供了三个存储类，名称为 **ibmc-vpc-block-10iops-tier** （默认）、**ibmc-vpc-block-5iops-tier** 和 **ibmc-vpc-block-custom**，可用于创建持久性卷声明(PVC)。IBM® VPC Block CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。
- *IBM® VPC Block CSI 驱动程序* 允许您创建并挂载 IBM® VPC Block PV。

5.17.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.17.3. 用户管理的加密

用户管理的加密功能允许您在安装过程中提供加密 OpenShift Container Platform 节点根卷的密钥，并允许所有受管存储类使用这些密钥加密置备的存储卷。您必须在 install-config YAML 文件中的 **platform.<cloud_type>.defaultMachinePlatform** 字段中指定自定义密钥。

此功能支持以下存储类型：

- Amazon Web Services (AWS) Elastic Block storage (EBS)
- Microsoft Azure Disk 存储
- Google Cloud Platform (GCP) 持久磁盘 (PD) 存储
- IBM Virtual Private Cloud (VPC) Block 存储

有关使用用户管理的 IBM VPC 加密安装的详情，请参考 IBM Cloud 的用户管理加密 和准备在 IBM Cloud 上安装。

其他资源

- 配置 CSI 卷

5.18. IBM POWER VIRTUAL SERVER BLOCK CSI DRIVER OPERATOR

5.18.1. 简介

IBM Power® Virtual Server Block CSI Driver 通过 IBM Power® Virtual Server Block CSI Driver Operator 安装，Operator 基于 **library-go**。OpenShift Container Platform **library-go** 框架是一个功能集合，允许用户轻松构建 OpenShift Operator。CSI Driver Operator 的大部分功能已经可用。IBM Power® Virtual Server Block CSI Driver Operator 由 Cluster Storage Operator 安装。如果平台类型是 Power Virtual Servers，Cluster Storage Operator 会安装 IBM Power® Virtual Server Block CSI Driver Operator。

5.18.2. 概述

OpenShift Container Platform 可以使用 IBM® Power Virtual Server Block Storage 的 Container Storage Interface (CSI) 驱动程序置备持久性卷 (PV)。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 IBM® Power Virtual Server Block 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 IBM® Power Virtual Server Block CSI Driver Operator 和 IBM® Power Virtual Server Block CSI 驱动程序。

- *IBM® Power Virtual Server Block CSI Driver Operator* 提供了两个存储类，名为 **ibm-powervs-tier1**（默认）和 **ibm-powervs-tier3**，为不同的层提供可用于创建持久性卷声明 (PVC)。IBM Power® Virtual Server Block CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。
- *IBM Power® Virtual Server Block CSI 驱动程序* 允许您创建并挂载 IBM Power® Virtual Server Block PV。

5.18.3. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

其他资源

- 配置 CSI 卷

5.19. OPENSTACK CINDER CSI DRIVER OPERATOR

5.19.1. 概述

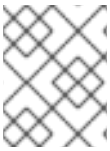
OpenShift Container Platform 可以使用 OpenStack Cinder 的 Container Storage Interface（CSI）驱动程序置备持久性卷（PV）。

在使用 Container Storage Interface (CSI) Operator 和驱动时，我们建议用户需要熟悉持久性存储和配置 CSI 卷。

要创建挂载到 OpenStack Cinder 存储资产中的 CSI 置备 PV，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中安装 OpenStack Cinder CSI Driver Operator 和 OpenStack Cinder CSI 驱动程序。

- *OpenStack Cinder CSI Driver Operator* 提供了一个 CSI 存储类，可用于创建 PVC。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。

- *OpenStack Cinder CSI 驱动程序* 允许您创建并挂载 OpenStack Cinder PV。



注意

OpenShift Container Platform 为 Cinder in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。如需更多信息，请参阅 CSI 自动迁移。

5.19.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

重要

OpenShift Container Platform 默认使用 CSI 插件来置备 Cinder 存储。

5.19.3. 使 OpenStack Cinder CSI 成为默认存储类

OpenStack Cinder CSI 驱动程序使用 **cinder.csi.openstack.org** 参数键来支持动态置备。

要在 OpenShift Container Platform 中启用 OpenStack Cinder CSI 置备，建议您使用 **standard-csi** 覆盖默认的树内存储类。另外，您可以创建持久性卷声明（PVC），并将存储类指定为 "standard-csi"。

在 OpenShift Container Platform 中，默认存储类引用树内 Cinder 驱动程序。但是，启用了 CSI 自动迁移后，使用默认存储类创建的卷实际上使用 CSI 驱动程序。

流程

使用以下步骤通过覆盖默认的树内存储类来应用 **standard-csi** 存储类。

1. 列出存储类：

```
$ oc get storageclass
```

输**出示例**

```
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. 对于默认存储类，将注解 **storageclass.kubernetes.io/is-default-class** 的值改为 **false**，如下例所示：

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 通过添加或修改注解 **storageclass.kubernetes.io/is-default-class=true** 来使另一个存储类作为默认设置。

```
$ oc patch storageclass standard-csi -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 验证 PVC 现在默认引用 CSI 存储类：

```
$ oc get storageclass
```

输**出示例**

```
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. 可选： 您可以定义一个新的 PVC 而无需指定存储类：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cinder-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

没有指定特定存储类的 PVC 会使用默认存储类自动置备。

6. 可选： 配置了新文件后，在集群中创建该文件：

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

其他资源

- 配置 CSI 卷

5.20. OPENSTACK MANILA CSI DRIVER OPERATOR

5.20.1. 概述

OpenShift Container Platform 可以使用 OpenStack Manila 共享文件系统服务的 Container Storage Interface（CSI）驱动程序置备持久性卷（PV）。

在使用 Container Storage Interface (CSI) Operator 和驱动时，我们建议用户需要熟悉持久性存储和配置 CSI 卷。

要创建挂载到 Manila 存储资产中的 CSI 置备 PV，OpenShift Container Platform 会在所有启用了 Manila 服务的 OpenStack 集群上默认安装 Manila CSI Driver Operator 和 Manila CSI 驱动程序。

- *Manila CSI Driver Operator* 会创建所需的存储类，以为所有可用 Manila 共享类型创建 PVC。Operator 安装在 **openshift-cluster-csi-drivers** 命名空间中。
- *Manila CSI 驱动程序*允许您创建并挂载 Manila PV。该驱动程序安装在 **openshift-manila-csi-driver** 命名空间中。

5.20.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.20.3. Manila CSI Driver Operator 限制

以下限制适用于 Manila Container Storage Interface (CSI) Driver Operator：

仅支持 **NFS**

OpenStack Manila 支持许多网络附加存储协议，如 NFS、CIFS 和 CEPHFS，它们可以在 OpenStack 云中有选择地启用。OpenShift Container Platform 中的 Manila CSI Driver Operator 只支持使用 NFS 协议。如果在底层的 OpenStack 云中没有 NFS 并启用了 NFS，则无法使用 Manila CSI Driver Operator 为 OpenShift Container Platform 置备存储。

如果后端是 **CephFS-NFS**，则不支持快照

要获取持久性卷 (PV) 快照并将卷恢复到快照，您必须确保使用的 Manila 共享类型支持这些功能。Red Hat OpenStack 管理员必须启用对快照 (**share type extra-spec snapshot_support**）的支持，并从与您要使用的存储类相关联的快照（**share type extra-spec create_share_from_snapshot_support**）创建共享。

不支持 **FSGroups**

因为 Manila CSI 提供了由多个读入者和多个写入者访问的共享文件系统，所以不支持 FSGroups。即使使用 ReadWriteOnce 访问模式创建的持久性卷也是如此。因此，务必不要在您手动创建的任何存储类中指定 **fsType** 属性，以便与 Manila CSI Driver 搭配使用。

重要

在 Red Hat OpenStack Platform 16.x 和 17.x 中，带有 CephFS 的共享文件系统服务 (Manila) 完全支持通过 Manila CSI 向 OpenShift Container Platform 提供共享。但是，此解决方案不适用于大规模扩展。务必查看 Red Hat OpenStack Platform 的 CephFS NFS Manila-CSI Workload Recommendations 中的重要建议。

5.20.4. 动态置备 Manila CSI 卷

OpenShift Container Platform 为每个可用 Manila 共享类型安装一个存储类。

所创建的 YAML 文件与 Manila 及其 Container Storage Interface（CSI）插件完全分离。作为应用程序开发人员，您可以动态置备 ReadWriteMany（RWX）存储，并部署应用程序 pod 使用 YAML 清单安全地使用存储。

在您的内部环境中，可以使用与 AWS、GCP、Azure 和其他平台中的 OpenShift Container Platform 使用的相同的 pod 和持久性卷声明（PVC）定义（PVC 定义中的存储类引用除外）。

注意

Manila 服务是可选的。如果 Red Hat OpenStack Platform（RHOSP）中没有启用该服务，则不会安装 Manila CSI 驱动程序，也不会创建 Manila 存储类。

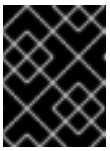
先决条件

- RHOSP 被部署为带有适当的 Manila 共享基础架构，以便使用它来在 OpenShift Container Platform 中动态置备和挂载卷。

流程 (UI)

使用 web 控制台动态创建 Manila CSI 卷：

1. 在 OpenShift Container Platform 控制台中，点击 **Storage → Persistent Volume Claims**。
2. 在持久性卷声明概述页中，点 **Create Persistent Volume Claim**。
3. 在接下来的页面中定义所需选项。
   a. 选择适当的存储类。
   b. 输入存储声明的唯一名称。
   c. 选择访问模式来指定您要创建的 PVC 的读写访问权限。



重要

如果您希望此 PVC 所使用的持久性卷（PV）可以被挂载到集群中的多个节点上的多个 pod 时，使用 RWX。

4. 定义存储声明的大小。
5. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

流程 (CLI)

使用命令行界面（CLI）动态创建 Manila CSI 卷：

1. 使用以下 YAML 描述的 **PersistentVolumeClaim** 对象创建并保存一个文件：

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. 如果您希望此 PVC 所使用的持久性卷（PV）可以被挂载到集群中的多个节点上的多个 pod 时，使用 RWX。
2. 置备存储后端的存储类的名称。Manila StorageClasses 由 Operator 置备，并包含 **csi-manila-** 前缀。

2. 运行以下命令，创建上一步中保存的对象：

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

创建了一个新的 PVC。

3. 运行以下命令验证卷已创建并就绪：

```
$ oc get pvc pvc-manila
```

pvc-manila 显示它的状态为 **Bound**。

现在，您可以使用新的 PVC 来配置 pod。

其他资源

- 配置 CSI 卷

5.21. SECRET STORE CSI 驱动程序

5.21.1. 概述

Kubernetes secret 以 Base64 编码的形式存储。etcd 为这些 secret 提供加密，但在检索 secret 时，它们会被解密并提供给用户。如果没有在集群中正确配置基于角色的访问控制，则具有 API 或 etcd 访问权限的任何人都可以检索或修改 secret。另外，有权在命名空间中创建 pod 的任何人都可以使用该命名空间中的任何 secret 来读取该命名空间中的任何 secret。

要安全地存储和管理您的 secret，您可以将 OpenShift Container Platform Secrets Store Container Storage Interface (CSI) Driver Operator 配置为使用供应商插件从外部 secret 管理系统（如 Azure Key Vault）挂载 secret。应用程序可以使用 secret，但 secret 在应用程序 pod 被销毁后不会在系统中保留。

Secret Store CSI Driver Operator（**secrets-store.csi.k8s.io**）允许 OpenShift Container Platform 将存储在企业级外部 secret 中的多个 secret、密钥和证书作为卷挂载到 pod 中。Secrets Store CSI Driver Operator 使用 gRPC 与供应商通信，以从指定的外部 secret 存储获取挂载内容。附加卷后，其中的数据将挂载到容器的文件系统。Secret 存储卷以 in-line 形式挂载。

有关 CSI 内联卷的更多信息，请参阅 CSI 内联临时卷。

重要

Secret Store CSI Driver Operator 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

在使用 CSI 驱动时，建议先熟悉 持久性存储和配置 CSI 卷。

5.21.1.1. Secret 存储供应商

以下 secret 存储供应商可用于 Secret Store CSI Driver Operator：

- AWS Secrets Manager
- AWS Systems Manager Parameter Store
- Azure Key Vault

5.21.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.21.3. 安装 Secret Store CSI 驱动程序

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 集群的管理员访问权限。

流程

安装 Secret Store CSI 驱动程序：

1. 安装 Secret Store CSI Driver Operator：
   a. 登录到 web 控制台。
   b. 点 **Operators → OperatorHub**。
   c. 通过在过滤器框中输入 "Secrets Store CSI" 来查找 Secrets Store CSI Driver Operator。
   d. 点 **Secrets Store CSI Driver Operator** 按钮。
   e. 在 **Secrets Store CSI Driver Operator** 页面中，点 **Install**。
   f. 在 **Install Operator** 页面中，确保：
      - 选择 **All namespaces on the cluster (default)**。
      - 安装的命名空间 被设置为 **openshift-cluster-csi-drivers**。
   g. 点 **Install**。
   安装完成后，Secret Store CSI Driver Operator 会在 web 控制台的 **Installed Operators** 部分列出。
2. 为驱动程序创建 **ClusterCSIDriver** 实例 (**secrets-store.csi.k8s.io**)：
   a. 点 **Administration → CustomResourceDefinitions → ClusterCSIDriver**。
   b. 在 **Instances** 选项卡上，单击 **Create ClusterCSIDriver**。

使用以下 YAML 文件：

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: secrets-store.csi.k8s.io
spec:
  managementState: Managed
```

c. 点 **Create**。

后续步骤

- 将 secret 从外部 secret 存储挂载到 CSI 卷

5.21.4. 卸载 Secret Store CSI Driver Operator

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 集群的管理员访问权限。

流程

卸载 Secret Store CSI Driver Operator：

1. 停止所有使用 **secrets-store.csi.k8s.io** 供应商的应用程序 pod。
2. 为所选 secret 存储删除任何第三方供应商插件。
3. 删除 Container Storage Interface (CSI) 驱动程序和相关清单：
   a. 点 **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**。
   b. 在 **Instances** 选项卡上，对于 **secrets-store.csi.k8s.io**，点左侧的下拉菜单，然后点 **Delete ClusterCSIDriver**。
   c. 出现提示时，单击 **Delete**。
4. 验证 CSI 驱动程序 pod 是否不再运行。
5. 卸载 Secret Store CSI Driver Operator：

注意

在卸载 Operator 前，必须先删除 CSI 驱动程序。

   a. 点 **Operators** → **Installed Operators**。
   b. 在 **Installed Operators** 页面中，在 **Search by name** 框中输入 "Secrets Store CSI" 来查找 Operator，然后点击它。
   c. 在 **Installed Operators** > **Operator** 详情页面 的右上角，点 **Actions** → **Uninstall Operator**。

d. 当在 **Uninstall Operator** 窗口中提示时，点 **Uninstall** 按钮从命名空间中删除 Operator。Operator 在集群中部署的任何应用程序都需要手动清理。
卸载后，Secret Store CSI Driver Operator 不再列在 web 控制台的 **Installed Operators** 部分。

5.21.5. 其他资源

- 配置 CSI 卷

5.22. VMWARE VSPHERE CSI DRIVER OPERATOR

5.22.1. 概述

OpenShift Container Platform 可以使用 Container Storage Interface（CSI）VMDK（VMDK）卷的 VMware vSphere 驱动程序来置备持久性卷（PV）。

在使用 CSI Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要创建挂载到 vSphere 存储资产中的 CSI 置备持久性卷(PV)，OpenShift Container Platform 在 **openshift-cluster-csi-drivers** 命名空间中默认安装 vSphere CSI Driver Operator 和 vSphere CSI 驱动程序。

- **vSphere CSI Driver Operator** ：Operator 提供了一个称为 **thin-csi** 的存储类，您可以使用它来创建持久性卷声明(PVC)。vSphere CSI Driver Operator 支持动态卷置备，允许按需创建存储卷，使集群管理员无需预置备存储。如果需要，您可以禁用此默认存储类 (请参阅管理默认存储类)。

- **vSphere CSI driver**：这个驱动程序可让您创建并挂载 vSphere PV。在 OpenShift Container Platform 4.15 中，驱动程序版本为 3.0.2。vSphere CSI 驱动程序支持底层红帽核心操作系统发行版本支持的所有文件系统，包括 XFS 和 Ext4。有关支持的文件系统的更多信息，请参阅 可用文件系统概述。

注意

对于新安装，OpenShift Container Platform 4.13 及更新的版本为 vSphere in-tree 卷插件提供自动迁移到对应的 CSI 驱动程序。更新至 OpenShift Container Platform 4.15 及更新的版本还提供自动迁移。有关更新和迁移的更多信息，请参阅 CSI 自动迁移。

CSI 自动迁移应该可以无缝进行。迁移不会改变您使用所有现有 API 对象的方式，如持久性卷、持久性卷声明和存储类。

5.22.2. 关于 CSI

在过去，存储厂商一般会把存储驱动作为 Kubernetes 的一个部分提供。随着容器存储接口 (CSI) 的实现，第三方供应商可以使用标准接口来提供存储插件，而无需更改核心 Kubernetes 代码。

CSI Operators 为 OpenShift Container Platform 用户提供了存储选项，如卷快照，它无法通过 in-tree 卷插件实现。

5.22.3. vSphere CSI 限制

以下限制适用于 vSphere Container Storage Interface (CSI) Driver Operator：

- vSphere CSI 驱动程序支持动态和静态置备。但是，当在 PV 规格中使用静态置备时，请不要在 **csi.volumeAttributes** 中使用键 **storage.kubernetes.io/csiProvisionerIdentity**，因为这个键代表动态置备的 PV。
- OpenShift Container Platform 不支持使用 vSphere 客户端接口在数据存储之间迁移持久性卷。

5.22.4. vSphere 存储策略

vSphere CSI Driver Operator 存储类使用 vSphere 的存储策略。OpenShift Container Platform 会自动创建一个存储策略，该策略以云配置中配置的数据存储为目标：

```
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
```

5.22.5. ReadWriteMany vSphere 卷支持

如果底层 vSphere 环境支持 vSAN 文件服务，则 OpenShift Container Platform 安装的 vSphere Container Storage Interface (CSI) Driver Operator 支持 provisioning of ReadWriteMany (RWX) 卷。如果没有配置 vSAN 文件服务，则 ReadWriteOnce (RWO) 是唯一可用的访问模式。如果您没有配置 vSAN 文件服务，且您请求 RWX，则卷将无法被创建，并记录错误。

有关在您的环境中配置 vSAN 文件服务的更多信息，请参阅 vSAN File Service。

您可以通过生成以下持久性卷声明 (PVC) 来请求 RWX 卷：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteMany
  storageClassName: thin-csi
```

请求 RWX 卷类型的 PVC 会导致置备 vSAN 文件服务支持的持久性卷 (PV)。

5.22.6. VMware vSphere CSI Driver Operator 要求

要安装 vSphere Container Storage Interface (CSI) Driver Operator，必须满足以下要求：

- VMware vSphere 版本： 7.0 更新 2 或更高版本； 8.0 更新 1 或更高版本
- vCenter 版本： 7.0 更新 2 或更高版本； 8.0 更新 1 或更高版本

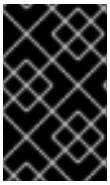
- 硬件版本 15 或更高版本的虚拟机
- 集群中还没有安装第三方 vSphere CSI 驱动程序

如果集群中存在第三方 vSphere CSI 驱动程序，OpenShift Container Platform 不会覆盖它。存在第三方 vSphere CSI 驱动程序可防止 OpenShift Container Platform 更新到 OpenShift Container Platform 4.13 或更高版本。

注意

只有在安装清单中使用 **platform: vsphere** 部署的集群中才支持 VMware vSphere CSI Driver Operator。

您可以为 Container Storage Interface (CSI) 驱动程序、vSphere CSI Driver Operator 和 vSphere Problem Detector Operator 创建自定义角色。自定义角色可以包含为每个 vSphere 对象分配最小权限集的权限集。这意味着 CSI 驱动程序、vSphere CSI Driver Operator 和 vSphere Problem Detector Operator 可以建立与这些对象的基本交互。



重要

在 vCenter 中安装 OpenShift Container Platform 集群会根据完整权限列表进行测试，如 "Required vCenter account privileges" 部分所述。通过遵循完整的特权列表，您可以减少创建具有一组受限权限的自定义角色时可能会出现意外和不支持的行为的可能性。

要删除第三方 CSI 驱动程序，请参阅删除第三方 vSphere CSI 驱动程序。

5.22.7. 删除第三方 vSphere CSI Driver Operator

OpenShift Container Platform 4.10 及更新的版本包括红帽支持的 vSphere Container Storage Interface (CSI) Operator Driver 的内置版本。如果您安装了由社区或其他供应商提供的 vSphere CSI 驱动程序，则可能会对集群禁用对下一个 OpenShift Container Platform 主版本（如 4.13 或更高版本）的更新。

OpenShift Container Platform 4.12 及更新的版本仍被完全支持，且对 4.12 的 z-stream 版本的更新（如 4.12.z）不被支持，但您必须在升级到下一个 OpenShift Container Platform 主版本前删除第三方 vSphere CSI Driver 来修正这个状态。删除第三方 vSphere CSI 驱动程序不需要删除关联的持久性卷(PV)对象，也不会发生数据丢失。

注意

这些说明可能不完整，因此请参阅厂商或社区供应商卸载指南，以确保删除驱动程序和组件。

卸载第三方 vSphere CSI 驱动程序：

1. 删除第三方 vSphere CSI 驱动程序（VMware vSphere Container Storage 插件）部署和 Daemonset 对象。
2. 删除之前使用第三方 vSphere CSI 驱动程序安装的 configmap 和 secret 对象。
3. 删除第三方 vSphere CSI 驱动程序 **CSIDriver** 对象：

```
~ $ oc delete CSIDriver csi.vsphere.vmware.com
```

```
csidriver.storage.k8s.io "csi.vsphere.vmware.com" deleted
```

从 OpenShift Container Platform 集群中删除了第三方 vSphere CSI 驱动程序后，Red Hat 的 vSphere CSI Driver Operator 安装会自动恢复，以及阻止升级到 OpenShift Container Platform 4.11 或更高版本的任何条件。如果您已有 vSphere CSI PV 对象，它们的生命周期现在由红帽的 vSphere CSI Driver Operator 管理。

5.22.8. vSphere 持久性磁盘加密

您可以在 vSphere 上运行的 OpenShift Container Platform 上对虚拟机 (VM) 和动态置备的持久性卷 (PV) 进行加密。

注意

OpenShift Container Platform 不支持 RWX 加密 PV。您无法从使用加密存储策略的存储类中请求 RWX PV。

您必须对虚拟机进行加密，然后才能加密 PV，您可以在安装过程中或安装后进行这些 PV。

有关加密虚拟机的详情，请参考：

- 加密虚拟机的要求
- 在安装过程中：安装 RHCOS 的 7 步并启动 OpenShift Container Platform bootstrap 过程
- 安装后在 vSphere 集群上启用加密

加密虚拟机后，您可以使用 vSphere Container Storage Interface (CSI) 驱动程序配置支持动态加密卷置备的存储类。这可以通过两种方式之一完成：

- 数据存储 URL ：此方法不灵活，它会强制您使用单个数据存储。它还不支持拓扑感知置备。
- 基于标签的放置：加密置备的卷，并使用基于标签的放置来针对特定的数据存储为目标。

5.22.8.1. 使用数据存储 URL

流程

使用数据存储 URL 加密：

1. 在支持加密的数据存储中找到默认存储策略的名称。
   这与用于加密虚拟机的策略相同。
2. 创建使用此存储策略的存储类：

```
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 支持加密的数据存储中的默认存储策略名称

5.22.8.2. 使用基于标签的放置

流程

使用基于标签的放置进行加密：

1. 在 vCenter 中，创建一个用于标记数据存储的类别，该类别将可供此存储类使用。此外，确保 **StoragePod(Datastore clusters)**、**Datastore**, and **Folder** 被选为创建类别的可关联实体。
2. 在 vCenter 中，创建使用之前创建的类别的标签。
3. 将之前创建的标签分配给可用于存储类的每个数据存储。确保数据存储与参与 OpenShift Container Platform 集群的主机共享。
4. 在 vCenter 中，从主菜单中点 **Policies and Profiles**。
5. 在 **Policies and Profiles** 页面中，在导航窗格中点 **VM Storage Policies**。
6. 点 **CREATE**。
7. 输入存储策略的名称。
8. 选择 **Enable host based rules** 和 **Enable tag based placement rules**。
9. 在 **Next** 选项卡中：
   a. 选择 **Encryption** 和 **Default Encryption Properties**。
   b. 选择之前创建的标签类别，然后选择 tag selected。验证策略是否选择匹配的数据存储。
10. 创建存储策略。
11. 创建使用存储策略的存储类：

```
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 为加密创建的存储策略的名称

5.22.9. vSphere CSI 拓扑概述

OpenShift Container Platform 提供了将 OpenShift Container Platform for vSphere 部署到不同的区域和区域的功能，允许您在多个计算集群和数据中心上部署，这有助于避免单点故障。

这可以通过在 vCenter 中定义区域和区域类别来实现，然后将这些类别分配给不同的故障域，如计算集群，方法是为这些区域和区域类别创建标签。创建适当的类别并给 vCenter 对象分配了标签后，您可以创建额外的机器集来创建负责在这些故障域中调度 pod 的虚拟机(VM)。

以下示例定义了两个区域和两个区的故障域：

表 5.6. 带有一个区域和两个区域的 vSphere 存储拓扑

Compute 集群	故障域	描述
Compute 集群: ocp1, Datacenter: Atlanta	openshift-region: us-east-1 (tag), openshift-zone: us-east-1a (tag)	这会在区域 us-east-1 中定义一个带有区域 us-east-1a 的故障域。
计算 cluster: ocp2, Datacenter: Atlanta	openshift-region: us-east-1 (tag), openshift-zone: us-east-1b (tag)	这会在名为 us-east-1b 的同一区域中定义不同的故障域。

5.22.9.1. 在安装过程中创建 vSphere 存储拓扑

5.22.9.1.1. 流程

- 在安装过程中指定拓扑。请参阅为 *VMware vCenter 配置区域和区域*部分。

不需要额外的操作，且 OpenShift Container Platform 创建的默认存储类是拓扑感知的，并允许在不同故障域中置备卷。

其他资源

- 为 VMware vCenter 配置区域和区域

5.22.9.2. 安装后创建 vSphere 存储拓扑

5.22.9.2.1. 流程

1. 在 VMware vCenter vSphere 客户端 GUI 中，定义适当的区域和区域校准和标签。
   虽然 vSphere 允许您使用任意名称创建类别，但 OpenShift Container Platform 强烈建议您使用 **openshift-region** 和 **openshift-zone** 名称来定义拓扑类别。

   有关 vSphere 类别和标签的更多信息，请参阅 VMware vSphere 文档。

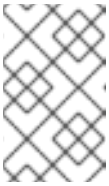
2. 在 OpenShift Container Platform 中，创建故障域。请参阅*在 vSphere 上为集群指定多个区域和区*部分。

3. 创建标签以在故障域间分配给数据存储：
   当 OpenShift Container Platform 跨越多个故障域时，可能无法在这些故障域间共享数据存储，即持久性卷 (PV) 的拓扑感知置备会很有用。

   a. 在 vCenter 中，创建一个类别来标记数据存储。例如，**openshift-zonal-datastore-cat**。您可以使用任何其他类别名称，只要类别唯一用于标记参与 OpenShift Container Platform 集群的数据存储。此外，确保 **StoragePod**、**Datastore**, and **Folder** 被选为创建类别的可关联实体。

   b. 在 vCenter 中，创建使用之前创建的类别的标签。本例使用标签名称 **openshift-zonal-datastore**。

c. 将之前创建的标签（本例中为 **openshift-zonal-datastore**）分配给故障域中的每个数据存储，并被视为动态置备。



注意

您可以使用您想用于数据存储类别和标签的任何名称。本例中使用的名称作为建议提供。确保定义仅唯一标识与 OpenShift Container Platform 集群中所有主机共享的数据存储的标签和类别。

4. 根据需要，创建一个存储策略，该策略以每个故障域中基于标签的数据存储为目标：

   a. 在 vCenter 中，从主菜单中点 **Policies and Profiles**。

   b. 在 **Policies and Profiles** 页面中，在导航窗格中点 **VM Storage Policies**。

   c. 点 **CREATE**。

   d. 输入存储策略的名称。

   e. 对于规则，选择 Tag 放置规则并选择以所需数据存储为目标的标签和类别（本例中为 **openshift-zonal-datastore** 标签）。
   数据存储列在存储兼容性表中。

5. 创建新使用新区存储策略的存储类：

   a. 点 **Storage** > **StorageClasses**。

   b. 在 **StorageClasses** 页面中，点 **Create StorageClass**。

   c. 在 **Name** 中输入新存储类的名称。

   d. 在 **Provisioner** 下，选择 **csi.vsphere.vmware.com**。

   e. 在 **Additional parameter** 下，对于 StoragePolicyName 参数，将 **Value** 设置为之前创建的新区存储策略的名称。

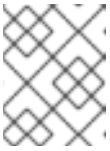
   f. 点 **Create**。

   输**出示例**

```
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) 新的拓扑了解存储类名称。

(2) 指定区存储策略。



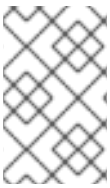
注意

您还可以通过编辑前面的 YAML 文件并运行 **oc create -f $FILE** 命令来创建存储类。

其他资源

- 在 vSphere 上为集群指定多个区域和区域
- VMware vSphere 标签文档

5.22.9.3. 在没有 infra 拓扑的情况下创建 vSphere 存储拓扑



注意

OpenShift Container Platform 建议使用基础架构对象在拓扑了解设置中指定故障域。在 infrastructure 对象中指定故障域，并在 **ClusterCSIDriver** 对象中指定 topology-categories 是一个不受支持的操作。

5.22.9.3.1. 流程

1. 在 VMware vCenter vSphere 客户端 GUI 中，定义适当的区域和区域校准和标签。
   虽然 vSphere 允许您使用任意名称创建类别，但 OpenShift Container Platform 强烈建议您使用 **openshift-region** 和 **openshift-zone** 名称来定义拓扑。

   有关 vSphere 类别和标签的更多信息，请参阅 VMware vSphere 文档。

2. 要允许容器存储接口(CSI)驱动程序检测到这个拓扑，请编辑 **clusterCSIDriver** 对象 YAML 文件 **driverConfig** 部分：

   - 指定之前创建的 **openshift-zone** 和 **openshift-region** 类别。
   - 将 **driverType** 设置为 **vSphere**。

   ```
   ~ $ oc edit clustercsidriver csi.vsphere.vmware.com -o yaml
   ```

   输出示例

   ```
   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. 确保 **driverType** 设置为 **vSphere**。
2. 在 vCenter 前面创建的 **openshift-zone** 和 **openshift-region** 类别。

3. 运行以下命令，验证 **CSINode** 对象是否有拓扑键：

```
~ $ oc get csinode
```

输**出示例**

```
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
```

输**出示例**

```
...
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. vSphere **openshift-zone** 和 **openshift-region** 目录中的拓扑键。

注意

CSINode 对象可能需要一些时间才能接收更新的拓扑信息。更新驱动程序后，CSI **Node** 对象应具有拓扑键。

4. 创建标签以在故障域间分配给数据存储：
当 OpenShift Container Platform 跨越多个故障域时，可能无法在这些故障域间共享数据存储，即持久性卷 (PV) 的拓扑感知置备会很有用。

   a. 在 vCenter 中，创建一个类别来标记数据存储。例如，**openshift-zonal-datastore-cat**。您可以使用任何其他类别名称，只要类别唯一用于标记参与 OpenShift Container Platform 集群的数据存储。此外，确保 **StoragePod**、**Datastore**, and **Folder** 被选为创建类别的可关联实体。

b. 在 vCenter 中，创建使用之前创建的类别的标签。本例使用标签名称 **openshift-zonal-datastore**。

c. 将之前创建的标签（本例中为 **openshift-zonal-datastore**）分配给故障域中的每个数据存储，并被视为动态置备。

注意

您可以使用您想要的任何名称进行类别和标签。本例中使用的名称作为建议提供。确保定义仅唯一标识与 OpenShift Container Platform 集群中所有主机共享的数据存储的标签和类别。

5. 创建一个存储策略，该策略以每个故障域中基于标签的数据存储为目标：

a. 在 vCenter 中，从主菜单中点 **Policies and Profiles**。

b. 在 **Policies and Profiles** 页面中，在导航窗格中点 **VM Storage Policies**。

c. 点 **CREATE**。

d. 输入存储策略的名称。

e. 对于规则，选择 Tag 放置规则并选择以所需数据存储为目标的标签和类别（本例中为 **openshift-zonal-datastore** 标签）。
数据存储列在存储兼容性表中。

6. 创建新使用新区存储策略的存储类：

a. 点 **Storage** > **StorageClasses**。

b. 在 **StorageClasses** 页面中，点 **Create StorageClass**。

c. 在 **Name** 中输入新存储类的名称。

d. 在 **Provisioner** 下，选择 **csi.vsphere.vmware.com**。

e. 在 **Additional parameter** 下，对于 StoragePolicyName 参数，将 **Value** 设置为之前创建的新区存储策略的名称。

f. 点 **Create**。

输出示例

```
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) 新的拓扑了解存储类名称。

2 指定区存储策略。

注意

您还可以通过编辑前面的 YAML 文件并运行 **oc create -f $FILE** 命令来创建存储类。

其他资源

- VMware vSphere 标签文档

5.22.9.4. 结果

从拓扑创建持久性卷声明(PVC)和 PV 感知存储类实际上是区域性的，应该根据 pod 的调度方式在相应区中使用数据存储：

```
~ $ oc get pv <pv-name> -o yaml
```

输出示例

```
...
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 具有区域的密钥。

3 PV 使用区域存储类。

5.22.10. 其他资源

- 配置 CSI 卷

第 6 章 通用临时卷

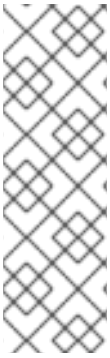
6.1. 概述

通用临时卷是临时卷类型，可以由支持持久性卷和动态置备的所有存储驱动程序提供。通用临时卷与 **emptyDir** 卷类似，它们为从头开始数据提供每个 pod 目录，这通常在置备后为空。

通用临时卷在 pod 规格中内联指定，并遵循 pod 的生命周期。它们与 pod 一起创建和删除。

通用临时卷具有以下功能：

- 存储可以是本地的或者网络附加存储。
- 卷可以有 pod 无法超过的固定大小。
- 根据驱动程序和参数，卷可能有一些初始数据。
- 支持卷上的典型的操作，假设驱动程序支持它们，包括快照、克隆、调整大小和存储容量跟踪。



注意

通用临时卷不支持离线快照和调整大小。

由于这个限制，以下 Container Storage Interface(CSI)驱动程序不支持通用临时卷的以下功能：

- Azure Disk CSI 驱动程序不支持调整大小。
- Cinder CSI 驱动程序不支持快照。

6.2. 生命周期和持久性卷声明

卷声明的参数允许在 pod 的卷源内。支持声明(PVC)的标签、注解和整个字段。当创建这样的 pod 时，临时卷控制器随后会在与 pod 相同的命名空间中创建实际的 PVC 对象（来自 *创建通用临时卷* 流程中显示的模板），并确保在 pod 被删除时删除 PVC。这会以两种方式之一触发卷绑定和置备：

- 另外，如果存储类使用即时卷绑定。
  通过立即绑定，调度程序被强制选择在卷可用后有权访问的节点。
- 当 pod 放入节点时（**WaitForFirstConsumervolume** 绑定模式）。
  建议这个卷绑定选项用于通用临时卷，因为调度程序可以为 pod 选择适当的节点。

就资源限制而言，具有通用临时存储的 pod 是提供该临时存储的 PVC 的所有者。当 pod 被删除时，Kubernetes 垃圾回收器会删除 PVC，然后，后者通常会触发删除卷，因为存储类的默认重新声明策略是删除卷。您可以使用一个带有保留策略的存储类创建 quasi-ephemeral 本地存储：存储会活跃 pod，在这种情况下，您必须确保卷清理单独发生。虽然这些 PVC 存在，它们可以像任何其他 PVC 一样使用。特别是，可以在卷克隆或快照中被引用为数据源。PVC 对象也保存卷的当前状态。

其他资源

- 创建通用临时卷

6.3. 安全性

您可以启用通用临时卷功能，以便可以创建 pod 的用户也可以间接创建持久性卷声明 (PVC)。即使这些用户没有直接创建 PVC 的权限，此功能也可以正常工作。集群管理员必须了解这一点。如果这不适用于您的安全模型，请使用准入 Webhook 来拒绝对象，如具有通用临时卷的 pod。

PVC 的一般命名空间配额仍适用，因此即使允许用户使用此新机制，它们也无法使用它来绕过其他策略。

6.4. 持久性卷声明命名

自动创建的持久性卷声明(PVC)由 pod 名称和卷名称的组合命名，中间带有连字符(-)。这种命名惯例还引入了不同 pod 之间以及 pod 和手动创建 PVC 之间潜在的冲突。

例如，**pod-a** 带有卷 **scratch**，**pod** 带有卷 **a-scratch**，它们都使用相同的 PVC 名称 **pod-a-scratch**。

检测到这样的冲突，如果为 pod 创建，PVC 仅用于临时卷。此检查基于所有权关系。现有 PVC 不会被覆盖或修改，但这不会解决冲突。如果没有正确的 PVC，pod 无法启动。

重要

在在同一命名空间中命名 pod 和卷时要小心，以便不会发生命名冲突。

6.5. 创建通用临时卷

流程

1. 创建 **pod** 对象定义，并将其保存到文件中。
2. 在文件中包括通用临时卷信息。

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 通用临时卷声明。

第 7 章 扩展持久性卷

7.1. 启用卷扩展支持

在扩展持久性卷前，**StorageClass** 对象必须将 **allowVolumeExpansion** 字段设置为 **true**。

流程

- 编辑 **StorageClass** 对象并添加 **allowVolumeExpansion** 属性：

```
$ oc edit storageclass <storage_class_name> 1
```

1 指定存储类的名称。

以下示例演示了在存储类配置的底部添加这一行。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
...
parameters:
  type: gp2
reclaimPolicy: Delete
allowVolumeExpansion: true 1
```

1 将这个属性设置为 **true** 允许在创建后扩展 PVC。

7.2. 扩展 CSI 卷

您可以在存储卷被创建后，使用 Container Storage Interface (CSI) 来扩展它们。

CSI 卷扩展不支持以下内容：

- 在扩展卷失败时进行恢复
- 缩小

先决条件

- 底层 CSI 驱动程序支持调整大小。
- 使用动态置备。
- 控制 **StorageClass** 对象的 **allowVolumeExpansion** 被设置为 **true**。如需更多信息，请参阅"启用卷扩展支持"。

流程

1. 对于持久性卷声明(PVC)，将 **.spec.resources.requests.storage** 设置为所需的新大小。
2. 观察 PVC 的 **status.conditions** 字段来查看调整大小是否完成。OpenShift Container Platform 在扩展过程中为 PVC 添加 **Resizing** 条件，该条件会在扩展完成后删除。

7.3. 使用支持的驱动程序扩展 FLEXVOLUME

当使用 FlexVolume 连接到后端存储系统时，您可以在创建后扩展持久性存储卷。这可以通过在 OpenShift Container Platform 中手动更新持久性卷声明 (PVC) 实现 。

当把驱动的 **RequiresFSResize** 设置为 **true** 时，FlexVolume 允许进行扩展。在 pod 重启时，FlexVolume 可以被扩展。

与其他卷类型类似，FlexVolume 也可以在被 pod 使用时扩展。

先决条件

- 底层卷驱动程序支持调整大小。
- 驱动程序的 **RequiresFSResize** 功能被设置为 **true**。
- 使用动态置备。
- 控制 **StorageClass** 对象的 **allowVolumeExpansion** 被设置为 **true**。

流程

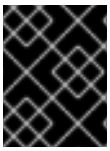
- 要在 FlexVolume 插件中使用 resizing 功能，您必须使用以下方法实现 **ExpandableVolumePlugin** 接口：

  RequiresFSResize

  如果为 **true**，直接更新容量。如果为 **false**，则调用 **ExpandFS** 方法来实现对文件系统大小的调整。

  ExpandFS

  如果为 **true**，在物理卷扩展后调用**ExpandFS** 来调整文件系统的大小。卷驱动程序也可以与执行物理卷调整一起调整文件系统的大小。



重要

因为 OpenShift Container Platform 不支持在 control plane 节点上安装 FlexVolume 插件，所以不支持 FlexVolume 的 control-plane 扩展。

7.4. 扩展本地卷

您可以使用本地存储操作器(LSO)手动扩展持久性卷(PV)和持久性卷声明(PVC)。

流程

1. 扩展底层设备。确定这些设备中提供了适当的容量。
2. 通过编辑 PV 的 **.spec.capacity** 字段来更新对应的 PV 对象以匹配新设备大小。
3. 对于用于将 PVC 绑定到 PVet 的存储类，设置 **allowVolumeExpansion:true**。
4. 对于 PVC，将 **.spec.resources.requests.storage** 设置为与新大小匹配。

根据需要，kubelet 应该自动扩展卷上的底层文件系统，并更新 PVC 的 status 字段来反映新的大小。

7.5. 使用文件系统扩展持久性卷声明（PVC）

根据需要重新定义文件系统大小的卷类型（如 GCE、EBS 和 Cinder）扩展 PVC 分为两个步骤。首先，扩展云供应商中的卷对象。其次，扩展节点上的文件系统。

只有在使用这个卷启动新的 pod 时，才会在该节点中扩展文件系统。

先决条件

- 控制 **StorageClass** 对象必须将 **allowVolumeExpansion** 设置为 **true**。

流程

1. 通过编辑 **spec.resources.requests**来修改 PVC 并请求一个新的大小。例如，以下命令将 **ebs** PVC 扩展至 8 Gi：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ebs
spec:
  storageClass: "storageClassWithFlagSet"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi (1)
```

(1) 将 **spec.resources.requests** 更新至更大的 PVC 来扩展 PVC。

2. 重新定义云供应商对象大小后， PVC 被设置为 **FileSystemResizePending**。输入以下命令检查条件：

```
$ oc describe pvc <pvc_name>
```

3. 当云供应商对象完成重新定义大小时，**PersistentVolume** 对象中的 **PersistentVolume.Spec.Capacity** 会显示新请求的大小。此时，您可从 PVC 创建或重新创建新 Pod 来完成文件系统大小调整。当 Pod 运行后，新请求的大小就可用，同时 **FileSystemResizePending** 条件从 PVC 中删除。

7.6. 在扩展卷失败时进行恢复

如果扩展底层存储失败，OpenShift Container Platform 管理员可以手动恢复 PVC 的状态，并取消改变大小的请求。否则，控制器会持续重试大小的请求。

流程

1. 把与 PVC 进行绑定的 PV 的 reclaim 策略设为 **Retain**。编辑 PV，把 **persistentVolumeReclaimPolicy** 的值改为 **Retain**。

2. 删除 PVC。

3. 手动编辑 PV 并从 PV specs 中删除 **claimRef** 条目，以确保新创建的 PVC 可以绑定到标记为 **Retain** 的 PV。这会将 PV 标记为 **Available**。
4. 以较小的大小，或底层存储架构可以分配的大小，重新创建 PVC。
5. 将 PVC 的 **volumeName** 值设为 PV 的名称。这使 PVC 只会绑定到置备的 PV。
6. 恢复 PV 上的 reclaim 策略。

其他资源

- 控制 **StorageClass** 对象的 **allowVolumeExpansion** 被设置为 **true** （请参阅启用卷扩展支持）。

第 8 章 动态置备

8.1. 关于动态置备

StorageClass 资源对象描述并分类了可请求的存储，并提供了根据需要为动态置备存储传递参数的方法。**StorageClass** 也可以作为控制不同级别的存储和访问存储的管理机制。集群管理员(**cluster-admin**)或存储管理员(**storage-admin**)可以在无需了解底层存储卷资源的情况下，定义并创建用户可以请求的 **StorageClass** 对象。

OpenShift Container Platform 的持久性卷框架启用了这个功能，并允许管理员为集群提供持久性存储。该框架还可让用户在不了解底层存储架构的情况下请求这些资源。

很多存储类型都可用于 OpenShift Container Platform 中的持久性卷。虽然它们都可以由管理员静态置备，但有些类型的存储是使用内置供应商和插件 API 动态创建的。

8.2. 可用的动态置备插件

OpenShift Container Platform 提供了以下置备程序插件，用于使用集群配置的供应商 API 创建新存储资源的动态部署：

存储类型	provisioner 插件名称	备注
Red Hat OpenStack Platform（RHOSP）Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface（CSI）	**manila.csi.openstack.org**	安装后, OpenStack Manila CSI Driver Operator 和 ManilaDriver 会自动为所有可用 Manila 共享类型创建动态置备所需的存储类。
Amazon Elastic Block Store (Amazon EBS)	**kubernetes.io/aws-ebs**	当在不同的区中使用多个集群进行动态置备时，使用 **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>**（每个集群的**<cluster_name>** 和 **<cluster_id>** 是唯一的）来标记（tag）每个节点。
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** 服务帐户需要相应的权限，以创建并获取 Secret 来存储 Azure 存储帐户和密钥。
GCE 持久性磁盘 (gcePD)	**kubernetes.io/gce-pd**	在多区（multi-zone）配置中，建议在每个 GCE 项目中运行一个 OpenShift Container Platform 集群，以避免在当前集群没有节点的区域中创建 PV。

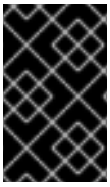
存储类型	provisioner 插件名称	备注
IBM Power® Virtual Server Block	**powervs.csi.ibm.com**	安装后，IBM Power® Virtual Server Block CSI Driver Operator 和 IBM Power® Virtual Server Block CSI Driver 会自动为动态置备创建所需的存储类。
VMware vSphere	**kubernetes.io/vsphere-volume**	

重要

任何选择的置备程序插件还需要根据相关文档为相关的云、主机或者第三方供应商配置。

8.3. 定义存储类

StorageClass 对象目前是一个全局范围的对象，必须由 **cluster-admin** 或 **storage-admin** 用户创建。



重要

根据使用的平台，Cluster Storage Operator 可能会安装一个默认的存储类。此存储类由 Operator 拥有和控制。不能在定义注解和标签之外将其删除或修改。如果需要实现不同的行为，则必须定义自定义存储类。

以下小节描述了 **StorageClass** 对象的基本定义，以及每个支持的插件类型的具体示例。

8.3.1. 基本 StorageClass 对象定义

以下资源显示了用来配置存储类的参数和默认值。这个示例使用 AWS ElasticBlockStore (EBS) 对象定义。

StorageClass 定义示例

```
kind: StorageClass (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: <storage-class-name> (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs (5)
parameters: (6)
  type: gp3
...
```

(1) （必需）API 对象类型。

(2) （必需）当前的 apiVersion。

3 （必需）存储类的名称。

4 （可选）存储类的注解。

5 （必需）与这个存储类关联的置备程序类型。

6 （可选）特定置备程序所需的参数，这将根据插件的不同而有所不同。

8.3.2. 存储类注解

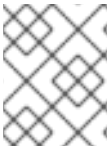
要将存储类设置为集群范围的默认值，请在存储类元数据中添加以下注解：

```
storageclass.kubernetes.io/is-default-class: "true"
```

例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

这允许任何没有指定特定存储类的持久性卷声明（PVC）通过默认存储类自动置备。但是，您的集群可以有多个存储类，但只有其中一个是默认存储类。



注意

beta 注解 **storageclass.beta.kubernetes.io/is-default-class** 当前仍然可用，但将在以后的版本中被删除。

要设置存储类描述，请在存储类元数据中添加以下注解：

```
kubernetes.io/description: My Storage Class Description
```

例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

8.3.3. RHOSP Cinder 对象定义

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: <storage-class-name> 1
provisioner: kubernetes.io/cinder
parameters:
  type: fast 2
  availability: nova 3
  fsType: ext4 4
```

1 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。

2 在 Cinder 中创建的卷类型。默认为空。

3 可用区。如果没有指定可用区，则通常会在所有 OpenShift Container Platform 集群有节点的所有活跃区域间轮换选择。

4 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

8.3.4. RHOSP Manila Container Storage Interface（CSI）对象定义

安装后, OpenStack Manila CSI Driver Operator 和 ManilaDriver 会自动为所有可用 Manila 共享类型创建动态置备所需的存储类。

8.3.5. AWS Elastic Block Store (EBS) 对象定义

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 （必需）存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。

2 (必需) 从 **io1**, **gp3**, **sc1**, **st1** 选择。默认值为 **gp3**。可用的 Amazon 资源名 (ARN) 值请查看 AWS 文档 。

3 可选：只适用于 **io1** 卷。每个 GiB 每秒一次 I/O 操作。AWS 卷插件乘以这个值，再乘以请求卷的大小以计算卷的 IOPS。数值上限为 20,000 IOPS，这是 AWS 支持的最大值。详情请查看 AWS 文档 。

4 可选：是否加密 EBS 卷。有效值为 **true** 或者 **false**。

5 可选：加密卷时使用的密钥的完整 ARN。如果没有提供任何信息，但 **encypted** 被设置为 **true**，则 AWS 会生成一个密钥。有效 ARN 值请查看 AWS 文档 。

6 可选：在动态置备卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

8.3.6. Azure Disk 对象定义

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 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。

2 强烈建议使用 **WaitForFirstConsumer**。这会置备卷，同时允许有足够的存储空间从可用区将 pod 调度到空闲 worker 节点上。

3 可能的值有 **Shared**（默认）、**Managed** 和 **Dedicated**。

重要

红帽仅在存储类中支持使用 **kind: Managed**。

使用 **Shared** 和 **Dedicated** 时，Azure 会创建非受管磁盘，而 OpenShift Container Platform 为机器 OS（root）磁盘创建一个受管磁盘。但是，因为 Azure Disk 不允许在节点上同时使用受管和非受管磁盘，所以使用 **Shared** 或 **Dedicated** 创建的非受管磁盘无法附加到 OpenShift Container Platform 节点。

4 Azure 存储帐户 SKU 层。默认为空。请注意，高级虚拟机可以同时附加 **Standard_LRS** 和 **Premium_LRS** 磁盘，标准虚拟机只能附加 **Standard_LRS** 磁盘，受管虚拟机只能附加受管磁盘，非受管虚拟机则只能附加非受管磁盘。

a. 如果 **kind** 设为 **Shared**，Azure 会在与集群相同的资源组中的几个共享存储帐户下创建所有未受管磁盘。

b. 如果 **kind** 设为 **Managed**，Azure 会创建新的受管磁盘。

c. 如果 **kind** 设为 **Dedicated**，并且指定了 **StorageAccount**，Azure 会将指定的存储帐户用于与集群相同的资源组中新的非受管磁盘。为此，请确保：

- 指定的存储帐户必须位于同一区域。
- Azure Cloud Provider 必须具有存储帐户的写入权限。

d. 如果 **kind** 设为 **Dedicated**，并且未指定 **StorageAccount**，Azure 会在与集群相同的资源组中为新的非受管磁盘创建一个新的专用存储帐户。

8.3.7. Azure File 对象定义

Azure File 存储类使用 secret 来存储创建 Azure File 共享所需的 Azure 存储帐户名称和存储帐户密钥。这些权限是在以下流程中创建的。

流程

1. 定义允许创建和查看 secret 的 **ClusterRole** 对象：

```
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 要查看并创建 secret 的集群角色名称。

2. 将集群角色添加到服务帐户：

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
system:serviceaccount:kube-system:persistent-volume-binder
```

3. 创建 Azure File **StorageClass** 对象：

```
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 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。

2 Azure 存储帐户的位置，如 **eastus**。默认为空，表示将在 OpenShift Container Platform 集群的位置创建新的 Azure 存储帐户。

3 Azure 存储帐户的 SKU 层，如 **Standard_LRS**。默认为空，表示将使用 **Standard_LRS** SKU 创建新的 Azure 存储帐户。

4 Azure 存储帐户的名称。如果提供了存储帐户，则忽略 **skuName** 和 **location**。如果没有提供存储帐户，则存储类会为任何与定义的 **skuName** 和 **location** 匹配的帐户搜索与资源组关联的存储帐户。

8.3.7.1. 使用 Azure File 时的注意事项

默认 Azure File 存储类不支持以下文件系统功能：

- 符号链接
- 硬链接
- 扩展属性
- 稀疏文件
- 命名管道

另外，Azure File 挂载目录的所有者用户标识符 (UID) 与容器的进程 UID 不同。可在 **StorageClass** 对象中指定 **uid** 挂载选项来定义用于挂载的目录的特定用户标识符。

以下 **StorageClass** 对象演示了修改用户和组标识符，以及为挂载的目录启用符号链接。

```
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. 指定用于挂载的目录的用户标识符。
2. 指定用于挂载的目录的组标识符。
3. 启用符号链接。

8.3.8. GCE PersistentDisk (gcePD) 对象定义

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. 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。
2. 选择 **pd-standard** 或 **pd-ssd**。默认为 **pd-standard**。

8.3.9. VMware vSphere 对象定义

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: csi.vsphere.vmware.com 2
```

1. 存储类的名称。持久性卷声明使用此存储类来置备关联的持久性卷。
2. 有关在 OpenShift Container Platform 中使用 VMware vSphere CSI 的更多信息，请参阅 Kubernetes 文档。

8.4. 更改默认存储类

使用以下步骤更改默认存储类。

例如，您有两个定义的存储类 **gp3** 和 **standard**，您想要将默认存储类从 **gp3** 改为 **standard**。

先决条件

- 使用 cluster-admin 权限访问集群。

流程

更改默认存储类：

1. 列出存储类：

```
$ oc get storageclass
```

输出示例

```
NAME            TYPE
gp3 (default)    kubernetes.io/aws-ebs 1
standard        kubernetes.io/aws-ebs
```

1. **(default)** 表示默认存储类。

2. 将所需的存储类设为默认存储类。

对于所需的存储类，运行以下命令将 **storageclass.kubernetes.io/is-default-class** 注解设置为 **true** ：

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

注意

您可以短时间内有多个默认存储类。但是，您应该确保最终只有一个默认存储类。

当存在多个默认存储类时，任何请求默认存储类 (**pvc.spec.storageClassName**=nil) 的持久性卷声明 (PVC) 都会获得最近创建的默认存储类，无论该存储类的默认存储类是什么，管理员都会在警报仪表板中收到警报，该类有多个默认存储类，**MultipleDefaultStorageClasses**。

3. 从旧的默认存储类中删除默认存储类设置。
   对于旧的默认存储类，运行以下命令将 **storageclass.kubernetes.io/is-default-class** 注解的值改为 **false** ：

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-
default-class": "false"}}}'
```

4. 确认更改：

```
$ oc get storageclass
```

输**出示例**

```
NAME                TYPE
gp3                 kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```