OpenShift Container Platform 4.9

存储

在 OpenShift Container Platform 中配置和管理存储

Last Updated: 2023-06-04

OpenShift Container Platform 4.9 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。这些术语可帮助您有效地了解 OpenShift Container Platform 架构。

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 不知道有多少可用的本地存储。
- Pod 无法请求保证的本地存储。
- 本地存储无法保证可以满足需求。
- Pod 可能会因为其他 pod 已使用完本地存储而被驱除。只有在足够的存储重新可用后，新的 pod 才可以使用。

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这个本地存储，但它不提供任何与 I/O 吞吐量和延迟有关的内容。

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。

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/sda1    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 AWS 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：

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

- Fibre Channel
- GCE Persistent Disk
- HostPath
- iSCSI
- 本地卷
- NFS
- OpenStack Manila
- OpenShift Container Storage
- 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。。

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。
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。

重要

卷访问模式是卷功能的描述。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。

例如，NFS 提供 **ReadWriteOnce** 访问模式。如果您需要卷的访问模式为 ROX，则需要在声明中指定 **read-only**。供应商中的错误会在运行时作为挂载错误显示。

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。在对节点进行 drain 操作前，需要首先确定使用这些卷的 pod 已被删除。

表 3.2. 支持的 PV 访问模式

卷插件	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
AWS EBS [2]	▮	-	-
AWS EFS	▮	▮	▮
Azure File	▮	▮	▮
Azure Disk	▮	-	-
Cinder	▮	-	-
Fibre Channel	▮	▮	-
GCE Persistent Disk	▮	-	-
HostPath	▮	-	-
iSCSI	▮	▮	-
本地卷	▮	-	-
NFS	▮	▮	▮
OpenStack Manila	-	-	▮
OpenShift Container Storage	▮	-	▮

卷插件	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
VMware vSphere	▮	-	-

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. 为依赖 AWS EBS 的 pod 使用重新创建的部署策略。

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。

使用以下命令可以查看与 PV 绑定的 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 Container Storage（只限于 Ceph RBD）
- VMware vSphere

注意

Fibre Channel 和 HostPath PV 不支持挂载选项。

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 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

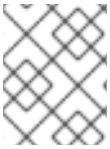
3.4.1. 存储类

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。这个存储类由 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
AWS EBS	▮	▮	▮
AWS EFS			
Azure Disk	▮	▮	▮
Azure File			
Cinder	▮	▮	▮
Fibre Channel	▮		▮
GCP	▮	▮	▮

卷插件	手动置备	动态置备	完全支持
HostPath			
iSCSI	▮		▮
本地卷	▮		▮
NFS			
OpenShift Container Storage	▮	▮	▮
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

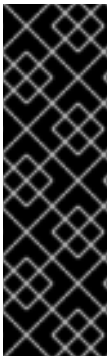
未指定值时将使用默认值 **Filesystem**。

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 AWS Elastic Block Store 卷 (EBS) 。您可以使用 Amazon EC2 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 AWS 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。AWS Elastic Block Store 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 默认使用 in-tree （非 CSI）插件来置备 AWS EBS 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

重要

存储的高可用性功能由底层存储供应商实现。

对于 OpenShift Container Platform，可以从 AWS EBS in-tree 自动迁移到 Container Storage Interface（CSI）驱动程序作为技术预览（TP）功能。启用迁移后，使用现有 in-tree 驱动程序置备的卷会自动迁移到使用 AWS EBS CSI 驱动程序。如需更多信息，请参阅 CSI 自动迁移功能。

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。这决定了所创建存储声明的读写权限。

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 AWS 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。

4.1.4. 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。附加的最大 EBS 卷数将单独计算为 in-tree 和 CSI 卷。

4.1.5. 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

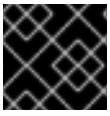
OpenShift Container Platform 支持 Microsoft Azure Disk 卷。您可以使用 Azure 为 OpenShift Container Platform 集群置备永久性存储。我们假设您对 Kubernetes 和 Azure 有一定的了解。Kubernetes 持久性卷框架允许管理员提供带有持久性存储的集群，并使用户可以在不了解底层存储架构的情况下请求这些资源。Azure 磁盘卷可以动态部署。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。持久性卷声明是针对某个项目或者命名空间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

重要

OpenShift Container Platform 默认使用 in-tree（非 CSI）插件来置备 Azure Disk 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。



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

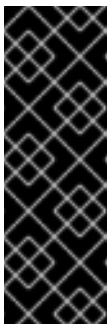
- Microsoft Azure Disk

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。有效选项为 **Premium_LRS**、**Standard_LRS**、**StandardSSD_LRS** 和 **UltraSSD_LRS**。
      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。这决定了所创建存储声明的读写权限。
   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。

4.3. 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。

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:

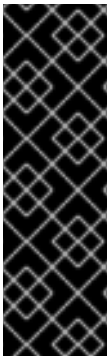
```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name 1
spec:
  containers:
    ...
    volumeMounts:
    - mountPath: "/data" 2
      name: azure-file-share
  volumes:
    - name: azure-file-share
      persistentVolumeClaim:
        claimName: claim1 3
```

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 默认使用 in-tree（非 CSI）插件来置备 Cinder 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

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**。

4.4.1.4. 节点上的 Cinder 卷的最大数量

默认情况下，OpenShift Container Platform 支持最多附加至一个节点的 256 Cinder 卷，并且禁用限制可附加卷的 Cinder predicate。要启用 predicate，将 **MaxCinderVolumeCount** 字符串添加到调度程序策略中的 **predicates** 字段。

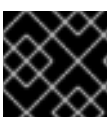
其他资源

- 有关修改调度程序策略的更多信息，请参阅修改调度程序策略。

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。

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

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。

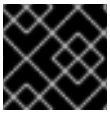
Pod 通过 **flexvolume** 树内插件与 FlexVolume 驱动程序交互。

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 默认使用 in-tree（非 CSI）插件来置备 gcePD 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

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。这决定了所创建存储声明的读写权限。
   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 GCE 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。

4.8. 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

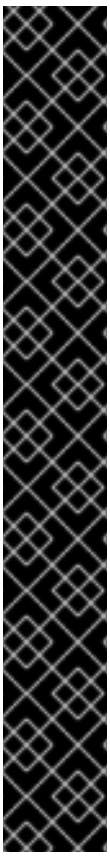
集群管理员必须将 pod 配置为以特权方式运行。这样可访问同一节点上的 pod。

4.8.1. 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/ubi8/ubi
    name: test-container
    command: ['sh', '-c', 'sleep 3600']
    volumeMounts:
    - mountPath: /host
      name: host-slash
  volumes:
   - name: host-slash
     hostPath:
       path: /
       type: ''
```

4.8.2. 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
```

4.8.3. 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。

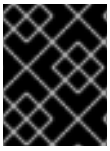
4.9. 使用 ISCSI 的持久性存储

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。

4.9.1. 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'
```

4.9.2. 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。

4.9.3. 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。

4.9.3.1. 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。

4.9.4. 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。

4.9.5. 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。

4.10. 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。

4.10.1. 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 project openshift-local-storage openshift.io/node-selector=''
   ```

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. 运行以下命令以获取 OpenShift Container Platform 的主版本和次版本。下一步中需要 **channel** 值。

   ```
   $ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
       grep -o '[0-9]*[.][0-9]*' | head -1)
   ```

   b. 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: "${OC_VERSION}"
  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。

4.10.2. 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)
      volumeMode: Filesystem (4)
      fsType: xfs (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。

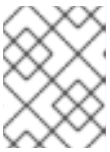
(3) 创建持久性卷对象时使用的存储类的名称。如果不存在，Local Storage Operator 会自动创建存储类。确保使用唯一标识此本地卷的存储类。

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

(5) 第一次挂载本地卷时所创建的文件系统。

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

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

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: "localblock-sc" 3
      volumeMode: Block 4
      devicePaths: 5
        - /path/to/device 6
```

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 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。

6 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。

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。

4.10.3. 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。

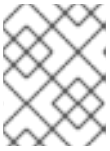
example-pv-filesystem.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv-filesystem
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem (1)
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage (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-storage (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-
storage         3m47s
example-pv1             1Gi        RWO            Delete           Bound       local-storage/pvc1   local-
storage         12h
example-pv2             1Gi        RWO            Delete           Bound       local-storage/pvc2   local-
```

```
storage         12h
example-pv3     1Gi     RWO     Delete     Bound     local-storage/pvc3   local-
storage         12h
```

4.10.4. 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
```

4.10.5. 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: localpvc
    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
```

4.10.6. 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。

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。

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. 在 *Administrator* 视角中，导航到 **Operators → Installed Operators**，再点 **Local Volume Discovery** 选项卡。
   b. 点 **Create Local Volume Discovery**。
   c. 根据您要在所有节点上还是在特定的节点上发现可用磁盘，选择 **All nodes** 或 **Select nodes**。

注意

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

   d. 点击 **Create**。

此时会显示名为 **auto-discover-devices** 的本地卷发现实例。

1. 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。
2. 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."

3. 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: example-storageclass (1)
  volumeMode: Filesystem
  fsType: ext4
  maxDeviceCount: 10
  deviceInclusionSpec:
    deviceTypes: (2)
      - disk
      - part
    deviceMechanicalProperties:
      - NonRotational
    minSize: 10G
    maxSize: 100G
    models:
      - SAMSUNG
      - Crucial_CT525MX3
    vendors:
      - ATA
      - ST2000LM
```

   (1) 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
```

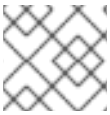
   c. 根据存储类验证本地持久性卷是否被动态置备：

```
$ 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   example-
storageclass   88m
local-pv-2ef7cd2a   100Gi   RWO   Delete   Available   example-
storageclass   82m
local-pv-3fa1c73   100Gi   RWO   Delete   Available   example-
storageclass   48m
```



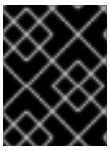
注意

结果会在从节点中删除后删除。必须手动删除符号链接。

4.10.7. 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: "localblock-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。

4.10.8. 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：

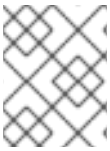
- 安装 Local Storage Operator 时启用对监控的支持。
- 当升级到 OpenShift Container Platform 4.9 或更高版本时，通过将 **operator-metering=true** 标签添加到命名空间来手动启用指标支持。

如需有关指标的更多信息，请参阅管理指标。

4.10.9. 删除 Local Storage Operator 资源

4.10.9.1. 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. 删除节点上的所有符号链接。

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。

   a. 在节点上创建一个调试 Pod：

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

   b. 将您的根目录改为 **/host** ：

   ```
   $ chroot /host
   ```

   c. 前往包含本地卷符号链接的目录。

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

1 用于创建本地卷的存储类的名称。

d. 删除归属于已移除设备的符号链接。

```
$ rm <symlink>
```

4.10.9.2. 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
```

4.11. 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

- 网络文件系统 (NFS)

4.11.1. 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
```

4.11.2. 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。

4.11.3. 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** 。

4.11.3.1. 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。

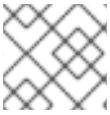
因此，上面的 pod 可以通过，并被启动。但是，如果需要进行组 ID 范围检查，使用自定义 SCC 就是首选的解决方案。可创建一个定义了最小和最大组群 ID 的自定义 SCC，这样就会强制进行组 ID 范围检查，组 ID **5555** 将被允许。

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）。

4.11.3.2. 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）。

4.11.3.3. 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
```

4.11.3.4. 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）。

4.11.4. 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。

4.11.5. 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`

4.12. RED HAT OPENSHIFT CONTAINER STORAGE

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。

Red Hat OpenShift Data Foundation 提供自己的文档库。完整的 Red Hat OpenShift Data Foundation 文档包括在 https://access.redhat.com/documentation/zh-cn/red_hat_openshift_data_foundation/4.9

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。

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
规划	
新的、已知的问题、显著的程序错误修复以及技术预览	OpenShift Data Foundation 4.9 发行注记
支持的工作负载、布局、硬件和软件要求、调整和扩展建议	规划 OpenShift Data Foundation 4.9 部署
部署	

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
使用 Amazon Web Services 部署 Red Hat OpenShift Data Foundation 进行本地或云存储	使用 Amazon Web Services 部署 OpenShift Data Foundation 4.9
将 Red Hat OpenShift Data Foundation 部署到裸机基础架构上的本地存储	使用裸机基础架构部署 OpenShift Data Foundation 4.9
部署 Red Hat OpenShift Data Foundation 以使用外部 Red Hat Ceph Storage 集群	在外部模式中部署 OpenShift Data Foundation 4.9
在现有 Google Cloud 集群上部署和管理 Red Hat OpenShift Data Foundation	使用 Google Cloud 部署和管理 OpenShift Data Foundation 4.9
部署 Red Hat OpenShift Data Foundation 在 IBM Z 基础架构上使用本地存储	使用 IBM Z 部署 OpenShift Data Foundation
在 IBM Power 系统上部署 Red Hat OpenShift Data Foundation	使用 IBM Power 系统部署 OpenShift Data Foundation
在 IBM Cloud 上部署 Red Hat OpenShift Data Foundation	使用 IBM Cloud 部署 OpenShift Data Foundation
在 Red Hat OpenStack Platform (RHOSP)上部署和管理 Red Hat OpenShift Data Foundation	使用 Red Hat OpenStack Platform 部署和管理 OpenShift Data Foundation 4.9
在 Red Hat Virtualization (RHV) 上部署和管理 Red Hat OpenShift Data Foundation	使用 Red Hat Virtualization Platform 部署和管理 OpenShift Data Foundation 4.9
在 VMware vSphere 集群上部署 Red Hat OpenShift Data Foundation	在 VMware vSphere 上部署 OpenShift Data Foundation 4.9
将 Red Hat OpenShift Data Foundation 更新至最新版本	更新 OpenShift Data Foundation
管理	
将存储分配给 Red Hat OpenShift Data Foundation 中的核心服务和托管应用程序，包括快照和克隆	管理并分配资源
使用多云对象网关（NooBaa）在混合云或多云环境中管理存储资源	管理混合和多云资源
为 Red Hat OpenShift Data Foundation 安全替换存储设备	替换设备
安全替换 Red Hat OpenShift Data Foundation 集群中的节点	替换节点

如果您要寻找 Red Hat OpenShift Data Foundation 的相关信息	请参阅以下 Red Hat OpenShift Data Foundation 文档：
在 Red Hat OpenShift Data Foundation 中扩展操作	扩展存储
监控 Red Hat OpenShift Data Foundation 4.9 集群	监控 OpenShift Data Foundation 4.9
错误和问题故障排除	OpenShift Data Foundation 4.9 故障排除
将 OpenShift Container Platform 集群从版本 3 迁移到版本 4	容器迁移工具

4.13. 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。

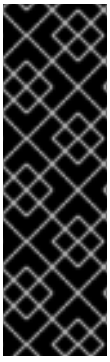
VMware vSphere 卷可以动态置备。OpenShift Container Platform 在 vSphere 中创建磁盘，并将此磁盘附加到正确的镜像。

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

OpenShift Container Platform 默认使用 in-tree （非 CSI）插件来置备 vSphere 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

其他资源

- VMware vSphere

4.13.1. 动态置备 VMware vSphere 卷

动态置备 VMware vSphere 卷是推荐的方法。

4.13.2. 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。

4.13.2.1. 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。

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

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

4.13.3. 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
```

4.13.3.1. 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。

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。

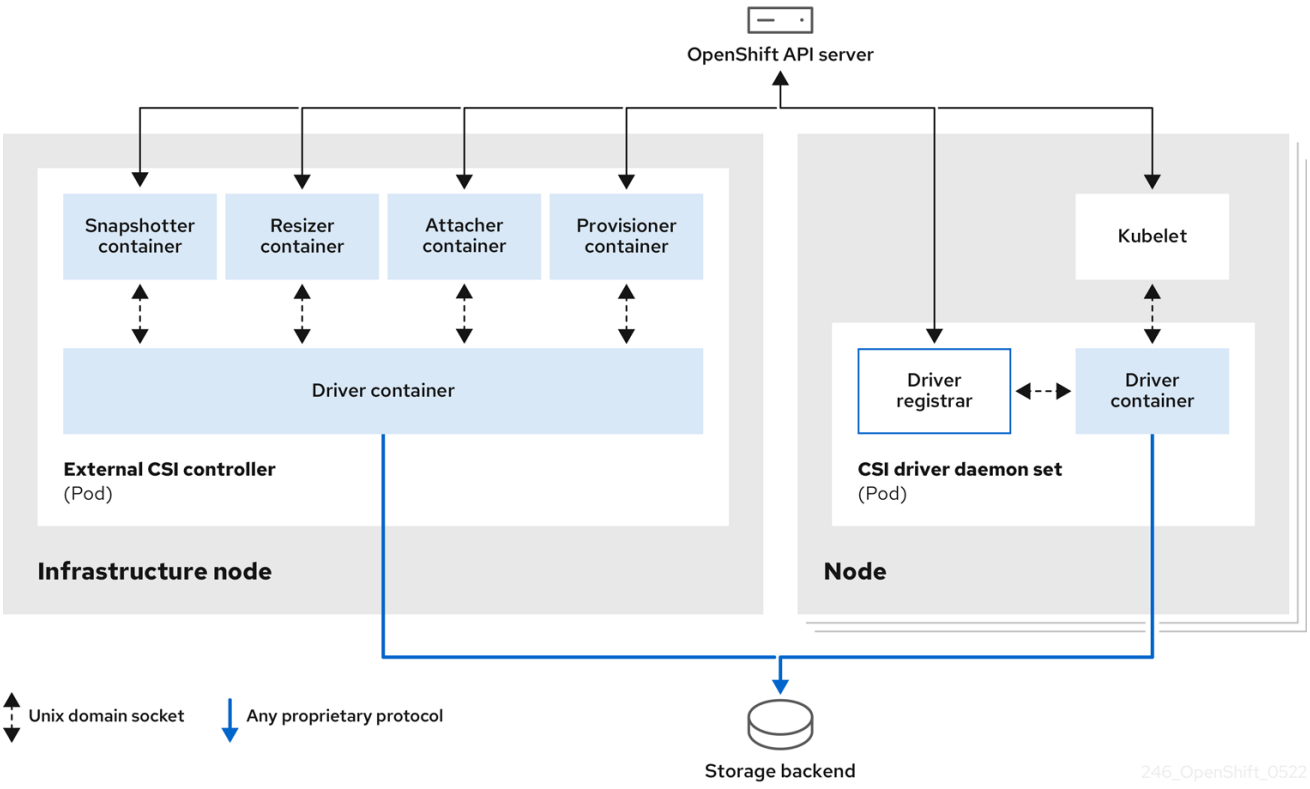
注意

OpenShift Container Platform 4.9 支持 CSI 规格的版本 1.5.0。

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。

下表描述了 OpenShift Container Platform 安装的 CSI 驱动程序及其支持的 CSI 功能，如卷快照、克隆和调整大小。

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
AWS EBS	▮	-	▮
AWS EFS（技术预览）	-	-	-
Google Cloud Platform(GCP)持久磁盘(PD)	▮	-	▮
Microsoft Azure Disk（技术预览）	▮	▮	▮
Microsoft Azure Stack Hub	▮	▮	▮
OpenStack Cinder	▮	▮	▮
OpenShift Container Storage	▮	▮	▮
OpenStack Manila	▮	-	-
Red Hat Virtualization（oVirt）	-	-	▮
VMware vSphere（技术预览）	-	-	-

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
```

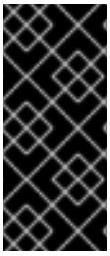
输出示例

```
NAME            STATUS    VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
mysql           Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi
RWO           cinder        3s
```

5.2. 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。

此功能仅可用于受支持的 Container Storage Interface (CSI) 驱动程序。



重要

CSI 内联临时卷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。

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。
- OpenShift Container Platform 不包括任何 CSI 驱动程序。请使用由开源社区或存储供应商提供的 CSI 驱动程序。请根据 CSI 驱动程序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。

5.2.2. 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
```

5.3. 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的概念。

5.3.1. 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 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。请根据 CSI 驱动程序提供的说明进行操作。
- 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。

5.3.2. 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。

5.3.2.1. 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。

5.3.2.2. 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。

5.3.3. 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。

5.3.3.1. 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。

5.3.4. 置备卷快照

置备快照的方法有两种：动态和手动。

5.3.4.1. 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。

5.3.4.2. 手动调配

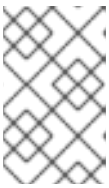
作为集群管理员，您可以手动置备大量 **VolumeSnapshotContent** 对象。它们包括了集群用户可以获得的实际卷的快照详情。

5.3.5. 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。



注意

如果 pod 正在使用，则不要为 PVC 创建卷快照。这样做可能会导致数据崩溃，因为 PVC 没有被静默（暂停）。确保首先停止正在运行的 pod，以确保快照的一致性。

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。

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**。

5.3.6. 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>
   ```

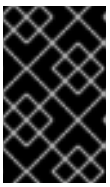
   输出示例

   ```
   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。

5.3.7. 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 * 已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。* 已创建卷快照并可使用。

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**。

5.4. 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。

5.4.1. 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。

5.4.1.1. 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。
- 源和目标的存储类必须相同。
- 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。

5.4.2. 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：

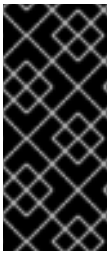
```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-1-clone 1
```

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。

5.5. CSI 自动迁移

OpenShift Container Platform 为支持的树内卷插件提供自动迁移到对应的 Container Storage Interface（CSI）驱动程序。



重要

CSI 自动迁移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。

5.5.1. 概述

启用自动迁移功能后，使用此功能支持的树内存储插件置备的卷会迁移到其对应的 CSI 驱动程序中。

支持以下驱动程序：

- Amazon Web Services（AWS）Elastic Block Storage（EBS）
- OpenStack Cinder
- Azure Disk
- Google Compute Engine Persistent Disk (in-tree) 和 Google Cloud Platform Persistent Disk (CSI)

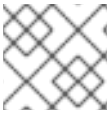
CSI 自动迁移应该可以无缝进行。启用此功能不会改变如何使用所有现有 API 对象（如 **PersistentVolume**、**PersistentVolumeClaim** 和 **StorageClasses**）。

默认情况下禁用自动迁移。

重要

在以后的 OpenShift Container Platform 发行版本中，CSI 自动迁移会默认启用，因此强烈建议您现在测试并报告任何问题。

5.5.2. 启用 CSI 自动迁移

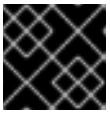


注意

启用 CSI 自动迁移排空，然后按顺序重启集群中的所有节点。这可能需要一些时间。

流程

- 启用功能门（请参阅 *Nodes → working with cluster → Enabling features using feature gates* ）。



重要

在使用功能门开启技术预览功能后，无法关闭它们。因此，集群升级会被阻止。

以下配置示例启用 CSI 自动迁移到此功能支持的所有 CSI 驱动程序：

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
```

```
spec:
  featureSet: TechPreviewNoUpgrade (1)
...
```

1 为 AWS EBS、Cinder 和 Azure Disk 启用自动迁移。

您可以通过设置 **CustomNoUpgrade featureSet** 和 **featuregates** 为所选 CSI 驱动程序指定 CSI 自动迁移：

- CSIMigrationAWS
- CSIMigrationOpenStack
- CSIMigrationAzure
- CSIMigrationGCE

以下配置示例只启用到 AWS EBS CSI 驱动程序：

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
spec:
  featureSet: CustomNoUpgrade
  customNoUpgrade:
    enabled:
      - CSIMigrationAWS (1)
    ...
```

1 只为 AWS EBS 启用自动迁移。

5.5.3. 其他资源

- 使用功能门启用功能

5.6. AWS ELASTIC BLOCK STORE CSI DRIVER OPERATOR

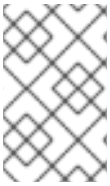
5.6.1. 概述

OpenShift Container Platform 可以使用 AWS Elastic Block Store（EBS）的 Container Storage Interface（CSI）驱动来置备持久性卷（PV）。

在使用 Container Storage Interface（CSI）Operator 和驱动器时，请先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 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。您也可以选择按照使用 AWS Elastic Block Store 的 Persistent Storage 中的内容创建 AWS EBS StorageClass。
- *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 4.9 前卸载 4.5 Operator 和驱动程序。

5.6.2. 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 in-tree （非 CSI）插件来置备 AWS EBS 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

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 AWS Elastic Block Store 的持久性存储。

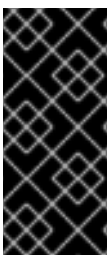
其他资源

- 使用 AWS Elastic Block Store 的持久性存储
- 配置 CSI 卷

5.7. AWS ELASTIC FILE SERVICE CSI DRIVER OPERATOR

5.7.1. 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)。



重要

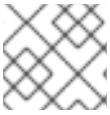
AWS EFS Driver Operator 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

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 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 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。
- *AWS EFS CSI 驱动程序* 允许您创建并挂载 AWS EFS PV。必须手动安装 AWS EFS CSI 驱动程序。



注意

AWS EFS 只支持区域卷，不支持 zonal 卷。

5.7.2. 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。

5.7.3. 安装 AWS EFS CSI Driver Operator

默认情况下，AWS EFS CSI Driver 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：
      - 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。

3. 安装 AWS EFS CSI 驱动程序：

a. 点 **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**。

b. 在 **Instances** 选项卡上，单击 **Create ClusterCSIDriver**。

c. 使用以下 YAML 文件：

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
    name: efs.csi.aws.com
spec:
  managementState: Managed
```

d. 点 **Create**。

e. 等待以下 Conditions 变为 "true" 状态：

- AWSEFSDriverCredentialsRequestControllerAvailable
- AWSEFSDriverNodeServiceControllerAvailable
- AWSEFSDriverControllerServiceControllerAvailable

5.7.4. 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。

安装之后，*AWS EFS CSI Driver Operator* 不会默认创建存储类。但是，您可以手动创建 AWS EFS 存储类。

5.7.4.1. 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**。

5.7.4.2. 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。

5.7.5. 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。

5.7.6. AWS EFS 的动态置备

AWS EFS CSI 驱动程序支持不同的动态置备形式，与其他 CSI 驱动程序不同。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 AWS 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），请参阅上面创建的 **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 :!StorageClass ：

5.7.7. 使用 AWS EFS 创建静态 PV

可以在没有动态置备的情况下将 AWS EFS 卷用作单个 PV。整个卷挂载到 pod。

先决条件

- 创建 AWS 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。

5.7.8. AWS EFS 安全

以下信息对 AWS EFS 安全性非常重要。

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。

5.7.9. AWS EFS 故障排除

以下信息提供了有关如何排除 AWS EFS 问题的指导：

- 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 AWS EFS 之间丢弃数据包造成的。

检查以下内容是否正确（请参阅在 AWS 中创建和配置 EFS 卷访问权限）：

- AWS 防火墙和安全组
- 网络：端口号和 IP 地址

5.7.10. 卸载 AWS EFS CSI Driver Operator

卸载 AWS EFS CSI Driver Operator 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**。

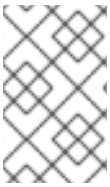
5. 卸载 AWS EFS CSI Operator：

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。

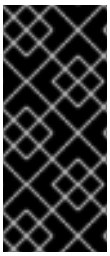
5.7.11. 其他资源

- 配置 CSI 卷

5.8. AZURE DISK CSI DRIVER OPERATOR

5.8.1. 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）。



重要

Azure Disk 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 Operator 和驱动程序时，建议先熟悉 持久性存储和配置 CSI 卷。

要使用此功能创建挂载到 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。

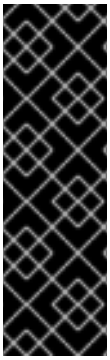
- 启用 *Azure Disk CSI Driver Operator* 后，提供了一个名为 **managed-csi** 的存储类，您可以使用它来创建持久性卷声明（PVC）。Azure Disk CSI Driver Operator 支持动态卷置备，方法是允许按需创建存储卷，使集群管理员无需预置备存储。

- *Azure Disk CSI 驱动程序* 允许您创建并挂载 Azure Disk PV。

5.8.2. 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 in-tree（非 CSI）插件来置备 Azure Disk 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，后续版本的 OpenShift Container Platform 中将最终删除树内插件。

5.8.3. 启用 Azure CSI Driver Operator

要启用 Azure Container Storage Interface（CSI）Driver Operator，您必须使用 **TechPreviewNoUpgrade** 功能集启用功能门。

流程

1. 通过 **TechPreviewNoUpgrade** 功能集启用功能门（请参阅 *Nodes → Enabling features using feature gates*）。

重要

在使用功能门（feature gate）启用技术预览功能后，无法关闭这些技术预览功能，并会防止集群升级。

2. 验证集群操作器存储：

```
$ oc get co storage
```

```
NAME    VERSION                         AVAILABLE   PROGRESSING DEGRADED    SINCE
storage 4.9.0-0.nightly-2021-09-08-162532   True        False       False       4h26m
```

- **AVAILABLE** 应该为 "True"。
- **PROGRESSING** 应该为 "False"。
- **DEGRADED** 应该为 "False"。

3. 验证 **openshift-cluster-csi-drivers** 命名空间中的 pod 状态，以确保它们正在运行：

```
$ oc get pod -n openshift-cluster-csi-drivers
```

```
NAME                                        READY   STATUS  RESTARTS    AGE
azure-disk-csi-driver-controller-5949bf45fd-pm4qb      11/11   Running 0           39m
azure-disk-csi-driver-node-2tcxr                3/3     Running 0           53m
azure-disk-csi-driver-node-2xjzm                3/3     Running 0           53m
```

```
azure-disk-csi-driver-node-6wrgk                  3/3     Running 0          53m
azure-disk-csi-driver-node-frvx2                  3/3     Running 0          53m
azure-disk-csi-driver-node-lf5kb                  3/3     Running 0          53m
azure-disk-csi-driver-node-mqdhh                  3/3     Running 0          53m
azure-disk-csi-driver-operator-7d966fc6c5-x74x5   1/1     Running 0          44m
```

4. 验证是否安装了存储类：

```
$ oc get storageclass
```

```
NAME                 PROVISIONER                RECLAIMPOLICY
VOLUMEBINDINGMODE    ALLOWVOLUMEEXPANSION    AGE
managed-premium (default)   kubernetes.io/azure-disk        Delete
WaitForFirstConsumer    true                    76m
managed-csi                 disk.csi.azure.com              Delete          WaitForFirstConsumer    true
51m (1)
```

(1) Azure 存储类

其他资源

- 使用 Azure Disk 的持久性存储
- 配置 CSI 卷
- 使用功能门启用功能

5.9. AZURE STACK HUB CSI DRIVER OPERATOR

5.9.1. 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。

5.9.2. 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。

5.9.3. 其他资源

- 配置 CSI 卷

5.10. GCP PD CSI DRIVER OPERATOR

5.10.1. 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 和驱动器时，请先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。您还可以选择按照使用 GCE Persistent Disk 的持久性存储 中的内容创建 GCP PD 存储类。
- **GCP PD 驱动程序**：该驱动程序可让您创建并挂载 GCP PD PV。

重要

OpenShift Container Platform 默认使用 in-tree（非 CSI）插件来置备 GCP PD 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

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。

5.10.3. 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 5.2. 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。

5.10.4. 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）。

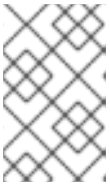
流程

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。

其他资源

- 使用 GCE Persistent Disk 的持久性存储
- 配置 CSI 卷

5.11. OPENSTACK CINDER CSI DRIVER OPERATOR

5.11.1. 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 和驱动器时，请先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。
- *OpenStack Cinder CSI 驱动程序* 允许您创建并挂载 OpenStack Cinder PV。

对于 OpenShift Container Platform，可以从 OpenStack Cinder in-tree 自动迁移到 CSI 驱动程序作为一个技术预览（TP）功能。启用迁移后，使用现有 in-tree 插件置备的卷将自动迁移为使用 OpenStack Cinder CSI 驱动程序。如需更多信息，请参阅 CSI 自动迁移功能。

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。

重要

OpenShift Container Platform 默认使用 in-tree（非 CSI）插件来置备 Cinder 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

5.11.3. 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

5.12. OPENSTACK MANILA CSI DRIVER OPERATOR

5.12.1. 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 和驱动器时，请先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。

5.12.2. 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。

5.12.3. 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

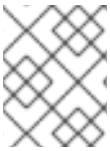
在 Red Hat OpenStack Platform 16.x 和 17.x 中，带有 CephFS 的共享文件系统服务 (Manila) 完全支持通过 Manila CSI 向 OpenShift Container Platform 提供共享。但是，此解决方案不适用于大规模扩展。务必查看 Red Hat OpenStack Platform 的 CephFS NFS Manila-CSI Workload Recommendations 中的重要建议。

5.12.4. 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

5.13. RED HAT VIRTUALIZATION CSI DRIVER OPERATOR

5.13.1. 概述

OpenShift Container Platform 可以使用 Red Hat Virtualization（RHV）的 Container Storage Interface（CSI）驱动程序置备持久性卷（PV）。

在使用 Container Storage Interface（CSI）Operator 和驱动器时，请先熟悉持久性存储和配置 CSI 卷。

要创建挂载到 Red Hat Virtualization（RHV）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 oVirt CSI Driver Operator 和 oVirt CSI 驱动程序。

- *oVirt CSI Driver Operator* 提供了一个默认的 **StorageClass** 对象，您可以使用它来创建持久性卷声明（PVC）。
- *oVirt CSI 驱动程序*允许您创建并挂载 oVirt PV。

5.13.2. 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

oVirt CSI 驱动程序不支持快照。

5.13.3. Red Hat Virtualization (RHV) CSI 驱动程序存储类

OpenShift Container Platform 创建了一个名为 **ovirt-csi-sc** 的 **StorageClass** 类型的默认对象，用于创建动态置备的持久性卷。

要为不同的配置创建额外的存储类，请使用以下示例 YAML 描述的 **StorageClass** 对象创建并保存文件：

ovirt-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: <storage_class_name> (1)
  annotations:
    storageclass.kubernetes.io/is-default-class: "<boolean>" (2)
provisioner: csi.ovirt.org
allowVolumeExpansion: <boolean> (3)
reclaimPolicy: Delete (4)
volumeBindingMode: Immediate (5)
parameters:
  storageDomainName: <rhv-storage-domain-name> (6)
  thinProvisioning: "<boolean>" (7)
  csi.storage.k8s.io/fstype: <file_system_type> (8)
```

(1) 存储类的名称。

(2) 如果存储类是集群中的默认存储类，则设置为 **false**。如果设置为 **true**，现有的默认存储类必须被编辑并设置为 **false**。

(3) **true** 启用动态卷扩展，**false** 会阻止它。建议使用 **true**。

(4) 此存储类的动态置备持久性卷会使用此重新声明策略创建。这个默认策略是 **Delete**。

(5) 指明如何置备和绑定 **PersistentVolumeClaim**。如果没有设置，则使用 **VolumeBindingImmediate**。此字段仅由启用 **VolumeScheduling** 功能的服务器应用。

(6) 要使用的 RHV 存储域名。

(7) 如果为 **true**，则置备磁盘。如果为 **false**，则预先分配磁盘。建议精简置备。

(8) 可选：要创建的文件系统类型。可能的值： **ext4**（默认）或 **xfs**。

5.13.4. 在 RHV 上创建持久性卷

创建 **PersistentVolumeClaim**（PVC） 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。

先决条件

- 登陆到一个正在运行的 OpenShift Container Platform 集群。
- 您在 **ovirt-credentials** secret 中提供了正确的 RHV 凭证。
- 已安装 oVirt CSI 驱动程序。
- 您至少定义了一个存储类。

流程

- 如果您使用 web 控制台在 RHV 上动态创建持久性卷：
  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。
4. 选择正确的 **StorageClass** 对象，默认为 **ovirt-csi-sc**。
5. 输入存储声明的唯一名称。
6. 选择访问模式。目前，RWO（ReadWriteOnce）是唯一受支持的访问模式。
7. 定义存储声明的大小。
8. 选择卷模式：
   Filesystem：作为目录挂载到 pod。这个模式是默认的模式。

   Block：会设备，其中没有任何文件系统
9. 点 **Create** 创建 **PersistentVolumeClaim** 对象并生成 **PersistentVolume** 对象。

- 如果您使用命令行界面（CLI）来动态创建 RHV CSI 卷：

1. 使用以下示例 YAML 描述的 **PersistentVolumeClaim** 对象创建并保存文件：

   pvc-ovirt.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-ovirt
spec:
  storageClassName: ovirt-csi-sc (1)
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <volume size> (2)
  volumeMode: <volume mode> (3)
```

   (1) 所需的存储类的名称。

   (2) GiB 中的卷大小。

   (3) 支持的选项：

   - **Filesystem**：作为目录挂载到 pod。这个模式是默认的模式。
   - **Block**：会设备，其中没有任何文件系统。

2. 运行以下命令，创建上一步中保存的对象：

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

3. 运行以下命令验证卷已创建并就绪：

```
$ oc get pvc pvc-ovirt
```

   pvc-ovirt 显示它的状态为 Bound。

其他资源

- 配置 CSI 卷
- 动态置备

5.14. VMWARE VSPHERE CSI DRIVER OPERATOR

5.14.1. 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）。

重要

vSphere 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 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 在启用此功能后，默认在 **openshift-cluster-csi-drivers** 命名空间中安装 vSphere CSI Driver Operator 和 vSphere CSI 驱动程序。

- **vSphere CSI Driver Operator**：启用后，Operator 提供了一个称为 **thin-csi** 的存储类，可用于创建持久性卷声明（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。
- **vSphere CSI driver**：这个驱动程序可让您创建并挂载 vSphere PV。

重要

OpenShift Container Platform 默认使用 in-tree （非 CSI）插件来置备 vSphere 存储。

在以后的 OpenShift Container Platform 版本中，计划使用现有树内插件置备的卷迁移到对应的 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。

完全迁移后，未来的 OpenShift Container Platform 版本将最终删除树内插件。

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. vSphere 存储策略

vSphere CSI Operator Driver 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
```

5.14.4. 启用 vSphere CSI Driver Operator

要启用 vSphere Container Storage Interface（CSI）Driver Operator,您必须使用 **TechPreviewNoUpgrade** 功能集启用功能门。

流程

1. 通过 **TechPreviewNoUpgrade** 功能集启用功能门（请参阅 *Nodes → Enabling features using feature gates*）。

重要

在使用功能门（feature gate）启用技术预览功能后，无法关闭这些技术预览功能，并会防止集群升级。

2. 验证集群操作器存储：

```
$ oc get co storage
```

```
NAME    VERSION                         AVAILABLE   PROGRESSING DEGRADED    SINCE
storage 4.9.0-0.nightly-2021-09-08-162532   True        False       False       4h26m
```

- **AVAILABLE** 应该为 "True"。
- **PROGRESSING** 应该为 "False"。
- **DEGRADED** 应该为 "False"。

3. 验证 **openshift-cluster-csi-drivers** 命名空间中的 pod 状态，以确保它们正在运行：

```
$ oc get pod -n openshift-cluster-csi-drivers
```

```
NAME                                        READY   STATUS  RESTARTS    AGE
vmware-vsphere-csi-driver-controller-5646dbbf54-cnsx7   9/9     Running 0           4h29m
vmware-vsphere-csi-driver-node-ch22q                    3/3     Running 0           4h37m
vmware-vsphere-csi-driver-node-gfjrb                    3/3     Running 0           4h37m
vmware-vsphere-csi-driver-node-ktlmp                    3/3     Running 0           4h37m
```

```
vmware-vsphere-csi-driver-node-lgksl                3/3     Running 0          4h37m
vmware-vsphere-csi-driver-node-vb4gv                3/3     Running 0          4h37m
vmware-vsphere-csi-driver-operator-7c7fc474c-p544t  1/1     Running 0          4h29m
```

```
NAME                                         READY   STATUS  RESTARTS    AGE
azure-disk-csi-driver-controller-5949bf45fd-pm4qb    11/11   Running 0          39m
azure-disk-csi-driver-node-2tcxr                     3/3     Running 0          53m
azure-disk-csi-driver-node-2xjzm                     3/3     Running 0          53m
azure-disk-csi-driver-node-6wrgk                     3/3     Running 0          53m
azure-disk-csi-driver-node-frvx2                     3/3     Running 0          53m
azure-disk-csi-driver-node-lf5kb                     3/3     Running 0          53m
azure-disk-csi-driver-node-mqdhh                     3/3     Running 0          53m
azure-disk-csi-driver-operator-7d966fc6c5-x74x5      1/1     Running 0          44m
```

1. 验证是否安装了存储类：

```
$ oc get storageclass
```

```
NAME            PROVISIONER                RECLAIMPOLICY    VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION    AGE
thin (default)  kubernetes.io/vsphere-volume    Delete          Immediate              false
5h43m
thin-csi        csi.vsphere.vmware.com          Delete          WaitForFirstConsumer    false
4h38m (1)
```

(1) vSphere 存储类

```
NAME                     PROVISIONER                RECLAIMPOLICY    VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION    AGE
managed-premium (default)   kubernetes.io/azure-disk        Delete          WaitForFirstConsumer    true
76m
managed-csi              disk.csi.azure.com              Delete          WaitForFirstConsumer    true
51m (1)
```

(1) Azure 存储类

5.14.5. 其他资源

- 使用功能门启用功能
- 配置 CSI 卷

第 6 章 扩展持久性卷

6.1. 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
```

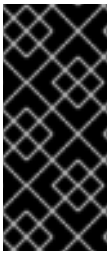
1 将这个属性设置为 **true** 允许在创建后扩展 PVC。

6.2. 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。

OpenShift Container Platform 默认支持 CSI 卷扩展。但是，需要一个特定的 CSI 驱动程序。

OpenShift Container Platform 4.9 支持 CSI 规范版本 1.1.0。



重要

扩展 CSI 卷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。

6.3. 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。

6.4. 使用文件系统扩展持久性卷声明（PVC）

对基于需要调整文件系统大小的卷类型（如 GCE、PD、EBS 和 Cinder）的 PVC 进行扩展分为两个步骤。这个过程包括在云供应商中扩展卷对象，然后在实际节点中扩展文件系统。

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，下面的命令将 ebs **PVC** 扩展到 8 Gi。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ebs
spec:
  storageClass: "storageClassWithFlagSet"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi (1)
```

1 将 **spec.resources.requests** 改为一个较大的值来扩展 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。

6.5. 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，控制器会在没有管理员干预的情况，一直尝试改变大小的请求。

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。这将会在以后重新创建。
3. 为了确保可以把 PVC 绑定到一个带有 **Retain** 设置的 PV，手工编辑 PV，把 **claimRef** 从 PV specs 中删除。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。

第 7 章 动态置备

7.1. 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。

7.2. 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。
AWS Elastic Block Store (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
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。

7.3. 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。这个存储类由 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。

7.3.1. 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: gp2
...
```

(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。

7.3.2. 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
...
```

7.3.3. 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**。

7.3.4. 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。

7.3.5. 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）选择 **io1**、 **gp2**、 **sc1**、 **st1**。默认为 **gp2**。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**。

7.3.6. 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。

7.3.7. 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。

7.3.7.1. 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。

7.3.8. 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**。

7.3.9. VMware vSphere 对象定义

vsphere-storageclass.yaml

vspnere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/vsphere-volume 2
parameters:
  diskformat: thin 3
```

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 的详情，请参阅 VMware vSphere 文档。

3 **diskformat**：**thin**、**zeroedthick** 和 **eagerzeroedthick** 都是有效的磁盘格式。如需有关磁盘格式类型的更多详情，请参阅 vSphere 文档。默认值为 **thin**。

7.4. 更改默认存储类

使用以下流程更改默认存储类。例如，您有两个定义的存储类 **gp2** 和 **standard**，您想要将默认存储类从 **gp2** 改为 **standard**。

1. 列出存储类：

```
$ oc get storageclass
```

输出示例

```
NAME                 TYPE
gp2 (default)        kubernetes.io/aws-ebs 1
standard             kubernetes.io/aws-ebs
```

1 **（默认）** 表示默认存储类。

2. 将默认存储类的 **storageclass.kubernetes.io/is-default-class** 注解的值改为 **false** ：

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 通过将 **storageclass.kubernetes.io/is-default-class** 注解设置为 **true** 来使另一个存储类成为默认值：

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 确认更改：

```
$ oc get storageclass
```

输出示例

```
NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/aws-ebs
```