OpenShift Container Platform 4.3

存储

在 OpenShift Container Platform 中配置和管理存储

Last Updated: 2020-11-30

OpenShift Container Platform 4.3 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 了解临时存储 **3**
1.1. 概述 3
1.2. 临时存储的类型 3
1.3. 临时存储管理 3
1.4. 监控临时存储 3

第 2 章 了解持久性存储 **5**
2.1. 持久性存储概述 5
2.2. 卷和声明的生命周期 5
2.3. 持久性卷（PV） 8
2.4. 持久性卷声明 (PVC) 12
2.5. 块卷支持 14

第 3 章 配置持久性存储 **18**
3.1. 使用 AWS ELASTIC 文件系统的永久性存储 18
3.2. 使用 AWS ELASTIC BLOCK STORE 的持久性存储 24
3.3. 使用 AZURE 持久性存储 25
3.4. 使用 AZURE FILE 的持久性存储 27
3.5. 使用 CINDER 的持久性存储 29
3.6. 使用容器存储接口 (CSI) 的持久性存储 31
3.7. 使用 FIBRE CHANNEL 持久性存储 34
3.8. 使用 FLEXVOLUME 的持久性存储 35
3.9. 使用 GCE PERSISTENT DISK 的持久性存储 39
3.10. 使用 HOSTPATH 的持久性存储 41
3.11. 使用 ISCSI 的持久性存储 43
3.12. 使用本地卷的持久性存储 45
3.13. 使用 NFS 的持久性存储 55
3.14. OPENSHIFT CONTAINER STORAGE 61
3.15. 使用 VMWARE VSPHERE 卷的持久性存储 61

第 4 章 扩展持久性卷 **66**
4.1. 启用卷扩展支持 66
4.2. 扩展 CSI 卷 66
4.3. 使用支持的驱动程序扩展 FLEXVOLUME 66
4.4. 通过文件系统扩展 PVC 67
4.5. 在扩展卷失败时进行恢复 68

第 5 章 动态置备 **69**
5.1. 关于动态置备 69
5.2. 可用的动态部署插件 69
5.3. 定义 STORAGECLASS 70
5.4. 修改默认的 STORAGECLASS 75

第 1 章 了解临时存储

1.1. 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。

1.2. 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。

1.3. 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。

1.4. 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 2 章 了解持久性存储

2.1. 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”。

2.2. 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。

2.2.1. 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。

2.2.2. 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。

2.2.3. 使用 pod 和声明的 PV

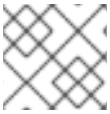
pod 使用声明（claim）作为卷。集群通过检查声明来找到绑定的卷，并为 pod 挂载相应的卷。对于那些支持多个访问模式的卷，您必须指定作为 pod 中的卷需要使用哪种模式。

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。

2.2.4. 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。

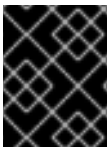
2.2.5. 释放 PersistentVolume

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。

2.2.6. 为 PersistentVolume 重新声明策略

PersistentVolume 的重新声明（reclaim）政策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。

2.2.7. 手动重新声明 PersistentVolume

删除 PersistentVolumeClaim（PVC）后，PersistentVolume（PV）仍然存在，并被视为"released"。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-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。

2.2.8. 更改 PersistentVolume 的重新声明策略

更改 PersistentVolume 的重新声明策略：

1. 列出集群中的 PersistentVolume:

```
$ 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 PersistentVolume 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 PersistentVolume 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。

2.3. 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：

PV 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。

2.3.1. PV 类型

OpenShift Container Platform 支持 **以下** persistentVolume 插件：

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

- iSCSI
- 本地卷
- NFS
- OpenStack Manila
- OpenShift Container Storage
- VMware vSphere

2.3.2. 容量

通常 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。

2.3.3. 访问模式

一个 **PersistentVolume** 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 2.1. 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

卷的 **AccessModes** 只是卷功能的一个描述符。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 2.2. 支持的 PV 访问模式

卷插件	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
AWS EBS [2]	▮	-	-
Azure File	▮	▮	▮
Azure Disk	▮	-	-
Cinder	▮	-	-
Fibre Channel	▮	▮	-
GCE Persistent Disk	▮	-	-
HostPath	▮	-	-
iSCSI	▮	▮	-
本地卷	▮	-	-
NFS	▮	▮	▮
OpenStack Manila	-	-	▮
OpenShift Container Storage	▮	-	▮
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，可以恢复或删除失败的节点，使卷可供其他节点使用。

2. 为依赖 AWS EBS 的 Pod 使用重新创建的部署策略。

2.3.4. 阶段

卷可以处于以下几个阶段：

表 2.3. 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>
```

2.3.4.1. 挂载选项

您可以使用注解（annotation） **volume.beta.kubernetes.io/mount-options** 指定在挂载 PV 时使用的挂载选项。

例如：

挂载选项示例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec (1)
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  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。

2.4. 持久性卷声明 (PVC)

每个 PVC 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：

PVC 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

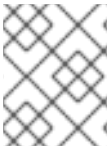
2.4.1. 存储类

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，ClusterStorageOperator 可能会安装一个默认的 StorageClass。这个 StorageClass 由操作员拥有和控制。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 StorageClass。

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 StorageClass 被标记为默认，则只能在 **storageClassName** 被显式指定时才能创建 PVC。因此，应只将一个 StorageClass 设置为默认值。

2.4.2. 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。

2.4.3. 资源

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。

2.4.4. 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

3 要使用的 PVC 名称（需要位于同一命名空间中）

2.5. 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

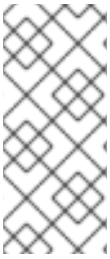
使用原始块卷的 pod 需要配置为允许特权容器。

下表显示了哪些卷插件支持块卷。

表 2.4. 块卷支持

卷插件	手动置备	动态置备	完全支持
AWS EBS	▮	▮	▮
Azure Disk	▮	▮	▮
Azure File			
Cinder	▮	▮	
Fibre Channel	▮		
GCP	▮	▮	▮
HostPath			
iSCSI	▮		▮
本地卷	▮		▮
NFS			

卷插件	手动置备	动态置备	完全支持
OpenShift Container Storage	▮	▮	▮
VMware vSphere	▮	▮	▮



注意

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详情，请参阅 https://access.redhat.com/support/offerings/techpreview/。

2.5.1. 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。

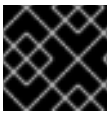
表 2.5. VolumeMode 可以使用的值

值	默认
Filesystem	是
Block	否

表 2.6. 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

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



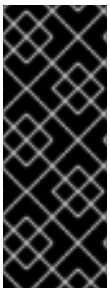
重要

未指定值时将使用默认值 **Filesystem**。

第 3 章 配置持久性存储

3.1. 使用 AWS ELASTIC 文件系统的永久性存储

OpenShift Container Platform 可以使用 Amazon Web Services (AWS) Elastic File System volumes (EFS)。您可以使用 AWS 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。



重要

elastic 文件系统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详情，请参阅 https://access.redhat.com/support/offerings/techpreview/。

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。

3.1.1. 先决条件

- 配置 AWS 安全组来允许来自 EFS 卷安全组的入站 NFS 流量。
- 将 AWS EFS 卷配置为允许来自任何主机的 SSH 流量。

其他参考资源

- Amazon EFS
- 用于 EFS 的 Amazon 安全组

3.1.2. 在 ConfigMap 中保存 EFS 变量

建议使用 ConfigMap 包含 EFS 置备程序所需的所有环境变量。

流程

1. 通过创建一个包含以下内容的 **configmap.yaml** 文件，定义包含环境变量的 OpenShift Container Platform ConfigMap：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: efs-provisioner
data:
  file.system.id: <file-system-id> 1
  aws.region: <aws-region> 2
  provisioner.name: openshift.org/aws-efs 3
  dns.name: "" 4
```

1. 定义 Amazon Web Services (AWS) EFS 文件系统 ID。
2. EFS 文件系统的 AWS 区域，比如 **us-east-1**。
3. 关联的 StorageClass 的置备程序名称。
4. 用来指定 EFS 卷所在的新 DNS 名称的一个可选参数。如果没有提供 DNS 名称，则置备程序将在 **<file-system-id>.efs.<aws-region>.amazonaws.com**中搜索 EFS 卷。

2. 在文件被配置后，运行以下命令在集群中创建该文件：

```
$ oc create -f configmap.yaml -n <namespace>
```

3.1.3. 为 EFS 卷配置授权

EFS 置备程序必须被授权与 AWS 端点沟通，同时观察和更新 OpenShift Container Platform 存储资源。以下是为 EFS 置备程序创建必要权限的流程。

流程

1. 创建一个 **efs-provisioner** 服务帐户：

```
$ oc create serviceaccount efs-provisioner
```

2. 创建定义所需权限的文件 **clusterrole.yaml** ：

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: efs-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
  - apiGroups: ["security.openshift.io"]
    resources: ["securitycontextconstraints"]
    verbs: ["use"]
    resourceNames: ["hostmount-anyuid"]
```

3. 创建一个文件 **clusterrolebinding.yaml**，该文件定义一个集群角色绑定把定义的角色和服务帐户进行绑定：

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: run-efs-provisioner
subjects:
  - kind: ServiceAccount
    name: efs-provisioner
    namespace: default (1)
roleRef:
  kind: ClusterRole
  name: efs-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
```

(1) 运行 EFS provisioner pod 的命名空间。如果 EFS 置备程序不在 **default** 命名空间中运行，则必须更新这个值。

4. 创建一个文件 **role.yaml**，它定义了具有所需权限的角色：

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-efs-provisioner
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
```

5. 创建一个文件 **rolebinding.yaml**，它定义了一个角色绑定将这个角色和服务帐户进行绑定：

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-efs-provisioner
subjects:
  - kind: ServiceAccount
    name: efs-provisioner
    namespace: default (1)
roleRef:
  kind: Role
  name: leader-locking-efs-provisioner
  apiGroup: rbac.authorization.k8s.io
```

(1) 运行 EFS provisioner pod 的命名空间。如果 EFS 置备程序不在 **default** 命名空间中运行，则必须更新这个值。

6. 在 OpenShift Container Platform 集群中创建资源：

```
$ oc create -f clusterrole.yaml,clusterrolebinding.yaml,role.yaml,rolebinding.yaml
```

3.1.4. 创建 EFS StorageClass

在创建 PersistentVolumeClaims 前，OpenShift Container Platform 集群中应已存在一个 StorageClass。以下是为 EFS 置备程序创建 StorageClass 的流程。

流程

1. 通过创建包含以下内容的 **storageclass.yaml** 来定义包含环境变量的 OpenShift Container Platform ConfigMap：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aws-efs
provisioner: openshift.org/aws-efs
parameters:
  gidMin: "2048" (1)
  gidMax: "2147483647" (2)
  gidAllocate: "true" (3)
```

(1) 定义卷分配的最小组群 ID (GID)。这是一个可选参数。默认值为 **2048**。

(2) 定义卷分配的最大 GID。这是一个可选参数。默认值为 **2147483647**。

(3) 指定是否为卷分配 GID。它是一个可选参数。如果为 **false**，则动态置备的卷不会分配 GID，这将允许所有用户对创建的卷进行读和写操作。默认值为 **true**。

2. 在文件被配置后，运行以下命令在集群中创建该文件：

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

3.1.5. 创建 EFS 置备程序

EFS 置备程序是一个 OpenShift Container Platform Pod，它将 EFS 卷作为 NFS 共享挂载。

先决条件

- 创建定义 EFS 环境变量的 ConfigMap。
- 创建包含必要的集群和角色权限的服务帐户。
- 为置备卷创建一个 StorageClass。
- 配置 Amazon Web Services (AWS) 安全组，允许在所有 OpenShift Container Platform 节点上接收进入的 NFS 网络数据。
- 配置 AWS EFS 卷安全组配置，允许来自所有源的 SSH 网络数据。

流程

1. 通过创建包含以下内容的 **provisioner.yaml** 文件定义 EFS 置备程序：

```
kind: Pod
apiVersion: v1
metadata:
  name: efs-provisioner
spec:
  serviceAccount: efs-provisioner
  containers:
```

```
      - name: efs-provisioner
        image: quay.io/external_storage/efs-provisioner:latest
        env:
          - name: PROVISIONER_NAME
            valueFrom:
              configMapKeyRef:
                name: efs-provisioner
                key: provisioner.name
          - name: FILE_SYSTEM_ID
            valueFrom:
              configMapKeyRef:
                name: efs-provisioner
                key: file.system.id
          - name: AWS_REGION
            valueFrom:
              configMapKeyRef:
                name: efs-provisioner
                key: aws.region
          - name: DNS_NAME
            valueFrom:
              configMapKeyRef:
                name: efs-provisioner
                key: dns.name
                optional: true
        volumeMounts:
          - name: pv-volume
            mountPath: /persistentvolumes
      volumes:
        - name: pv-volume
          nfs:
            server: <file-system-id>.efs.<region>.amazonaws.com (1)
            path: / (2)
```

(1) 包含 EFS 卷的 DNS 名称。必须为 Pod 更新这个字段以可以发现 EFS 卷。

(2) EFS 卷的挂载路径。每个持久性卷都是作为 EFS 卷的独立子目录创建的。如果这个 EFS 卷是用于 OpenShift Container Platform 以外的其他项目，那么建议您在 EFS 上为集群手动创建一个独立的 OpenShift Container Platform 子目录，以防止数据被其他项目访问。指定不存在的目录会导致错误。

2. 在文件被配置后，运行以下命令在集群中创建该文件：

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

3.1.6. 创建 EFS persistentVolumeClaim

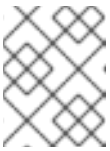
创建 EFS persistentvolumeclaim 的目的是为了使 Pod 可以挂载底层的 EFS 存储。

先决条件

- 创建 EFS provisioner pod。

流程 (UI)

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. 从列表中选择您创建的存储类。
   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。



注意

虽然您必须输入大小，但每个访问 EFS 卷的 Pod 都有无限存储。定义一个值，比如 **1Mi**，它会提醒您存储大小是无限的。

4. 点击 **Create** 创建持久性卷声明，并生成一个持久性卷。

流程 (CLI)

1. 另外，您还可以通过创建一个包含以下内容的 **pvc.yaml** 文件来定义 EFS PersistentVolumeClaim：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: efs-claim 1
  namespace: test-efs
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: openshift.org/aws-efs
  finalizers:
    - kubernetes.io/pvc-protection
spec:
  accessModes:
    - ReadWriteOnce 2
  resources:
    requests:
      storage: 5Gi 3
  storageClassName: aws-efs 4
  volumeMode: Filesystem
```

1. PVC 的唯一名称。
2. 决定所创建 PVC 的读写访问权限的访问模式。
3. 定义 PVC 的大小。
4. EFS 置备程序 StorageClass 的名称。

2. 在文件被配置后，运行以下命令在集群中创建该文件：

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

3.2. 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 AWS 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

存储的高可用性功能由底层存储供应商实现。

其他参考资源

- Amazon EC2

3.2.1. 创建 EBS 存储类

StorageClasses 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/aws-ebs**。
   e. 根据需要为存储类输入附加参数。
4. 点 **Create** 创建存储类。

3.2.2. 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。

3.2.3. 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 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。

3.2.4. 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。

3.3. 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

存储的高可用性功能由底层的存储架构提供。

其他参考资源

- Microsoft Azure Disk

3.3.1. 创建 Azure 存储类

StorageClasses 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。

其他参考资源

- Azure Disk Storage Class

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**.
   e. 根据需要为存储类输入附加参数。
4. 点 **Create** 创建存储类。

3.3.2. 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。

3.3.3. 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。

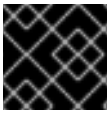
这将可以使用未格式化的 Azure 卷作为持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

3.4. 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。Azure File 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。PersistentVolumeClaim 是针对某一项目或命名空间的，应用程序中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。

其他参考资源

- Azure File

3.4.1. 创建 Azure File 共享 PersistentVolumeClaim

要创建 PersistentVolumeClaim，您必须首先定义包含 Azure 帐户和密钥的 Secret。这个 Secret 用在 PersistentVolume 定义中，应用程序中使用的 PersistentVolumeClaim 将引用它。

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：

```
$ 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. 创建一个引用您创建的 Secret 的 PersistentVolume：

```
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. PersistentVolume 的名称。
2. 此 PersistentVolume 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. 创建一个映射到您创建的 PersistentVolume 的 PersistentVolumeClaim：

```
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. PersistentVolumeClaim 的名称。
2. 此 PersistentVolumeClaim 的大小。
3. 用于置备 PersistentVolume 的 StorageClass 名称。指定 PersistentVolume 定义中使用的 StorageClass。
4. 引用 Azure File 共享的现有 PersistentVolume 的名称。

3.4.2. 在 Pod 中挂载 Azure File 共享

创建 PersistentVolumeClaim 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

- PersistentVolumeClaim 存在，已映射到底层的 Azure File 共享。

流程

- 创建可挂载现有 PersistentVolumeClaim 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。
3. 之前创建的 PersistentVolumeClaim 的名称。

3.5. 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。

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。

3.5.1. 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

3.5.1.1. 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

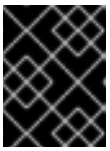
1. 将对象定义保存到文件中。

   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
```

3.5.1.2. 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。

3.5.1.3. 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。

前提条件

- 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 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** 。

3.6. 使用容器存储接口 (CSI) 的持久性存储

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。

重要

OpenShift Container Platform 不附带任何 CSI 驱动程序。建议您使用由开源社区或存储供应商提供的 CSI 驱动程序。

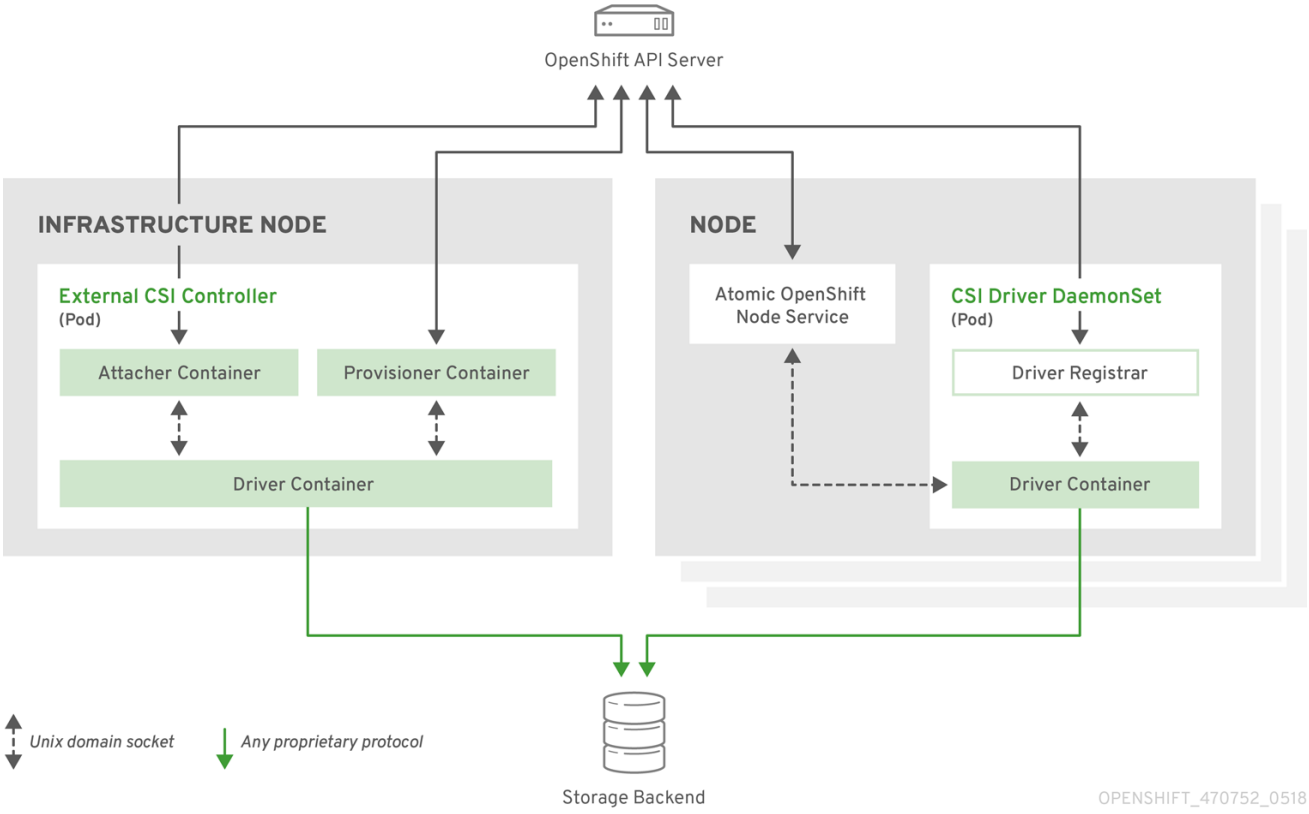
各个驱动程序的安装过程可能会有所不同，请参阅相应的驱动器文档。请根据 CSI 驱动程序提供的说明进行操作。

OpenShift Container Platform 4.3 支持 CSI 规范版本 1.1.0。

3.6.1. 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 DaemonSet。

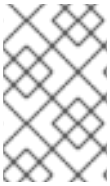
3.6.1.1. 外部 CSI 控制器

外部 CSI 控制器是一个部署，它部署带有以下三个容器的一个或多个 pod：

- 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。

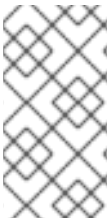
- 一个 CSI 驱动程序容器

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。

3.6.1.2. CSI Driver DaemonSet

CSI 驱动程序 DaemonSet 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上有尽量少的凭证。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）。

3.6.2. 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 StorageClass 及进行配置的参数文档。

创建的 StorageClass 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 StorageClass 的名称。

2 已安装的 CSI 驱动程序名称

3.6.3. 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。
- 为动态置备创建了一个 StorageClass。

流程

- 创建 MySQL 模板：

```
# oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

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

3.7. 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。

其他参考资源

- Fibre Channel

3.7.1. 置备

要使用 persistenceVolume API 置备 Fibre Channel 卷，必须有：

- **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。

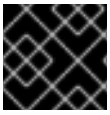
先决条件

- 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:
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] 1
    lun: 2
    fsType: ext4
```

1 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。

3.7.1.1. 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 都被映射到一个单独的 PersistentVolume，不同 PersistentVolume 必须有不同的名称。

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。

3.7.1.2. Fibre Channel 卷安全

用户使用 PersistentVolumeClaim 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。尝试访问其他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。

3.8. 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。

其他参考资源

- 扩展持久性卷

3.8.1. 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** 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。

3.8.2. 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。

3.8.3. 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*。

3.8.4. 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。

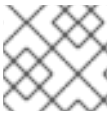
使用 FlexVolume 驱动程序示例定义持久性卷对象

```
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。

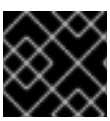
3.9. 使用 GCE PERSISTENT DISK 的持久性存储

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

存储的高可用性功能由底层的存储架构提供。

其他参考资源

- GCE Persistent Disk

3.9.1. 创建 GCE 存储类

StorageClasses 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/gce-pd**。
   e. 根据需要为存储类输入附加参数。
4. 点 **Create** 创建存储类。

3.9.2. 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。

3.9.3. 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。

3.10. 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。

3.10.1. 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 StorageClasses 设置动态置备。

hostPath 卷必须静态置备 。

3.10.2. 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。使用 PersistentVolume 对象定义创建一个 **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。PersistentVolumeClaim 或 Pod 通过这个名称来识别它。

(2) 用于将 PersistentVolumeClaim 请求绑定到此 PersistentVolume。

(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。

2. 从该文件创建 PV：

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

3. 定义持久性卷声明（PVC）。创建包括 PVC 对象定义的一个 **pv.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
```

3.10.3. 在特权 Pod 中挂载 hostPath 共享

创建 PersistentVolumeClaim 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

- 存在的 PersistentVolumeClaim 被映射到底层的 hostPath 共享。

流程

- 创建可挂载现有 PersistentVolumeClaim 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 hostPath 共享的路径。

4 之前创建的 PersistentVolumeClaim 的名称。

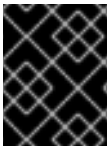
3.11. 使用 ISCSI 的持久性存储

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。

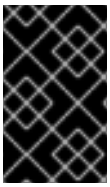
重要

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

OpenShift 假设集群中的所有节点都已配置了 iSCSI initator，即安装了 **iscsi-initiator-utils** 软件包，并在 **/etc/iscsi/initiatorname.iscsi** 中配置了它们的 initiator 的名称。请参阅《存储管理指南》 。

3.11.1. 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。

例 3.1. 持久性卷对象定义

```
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'
```

3.11.2. 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通过指定一个数量（例如，10Gi）来请求持久性存储，并与相等或更大容量的对应卷匹配。

3.11.3. iSCSI 卷安全

用户使用 **PersistentVolumeClaim** 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。

3.11.3.1. Challenge Handshake Authentication Protocol (CHAP) 配置

另外，OpenShift 可使用 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。这个 Secrets 对象必须在所有需要使用引用卷的命名空间中可用。

3.11.4. 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。

使用 **portals** 字段在 pod 规格中指定多路径。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。

3.11.5. 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。

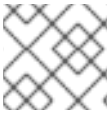
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: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

1 指定 initiator 的名称。

3.12. 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 PVC 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。

3.12.1. 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 **local-storage** 项目：

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

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

   要允许在 master 和基础架构节点上创建本地存储，请输入以下命令为 DaemonSet 添加容限：

```
$ oc patch ds local-storage-local-diskmaker -n local-storage -p '{"spec": {"template": {"spec": {"tolerations":[{"operator": "Exists"}]}}}}'
```

```
$ oc patch ds local-storage-local-provisioner -n local-storage -p '{"spec": {"template": {"spec": {"tolerations":[{"operator": "Exists"}]}}}}'
```

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 **Create Operator Subscription** 页面中，选择 **A specific namespace on the cluster**。从下拉菜单中选择 **local-storage**。
6. 将 **Update Channel** 和 **Approval Strategy** 的值调整为所需的值。
7. 点 **Subscribe**.

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

使用 CLI

1. 通过 CLI 安装 Local Storage Operator。

a. 创建一个对象 YAML 文件来为 Local Storage Operator 定义一个 Namespace、OperatorGroup 和 Subscription。例如 **local-storage.yaml**:

local-storage 示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: local-storage
---
apiVersion: operators.coreos.com/v1alpha2
kind: OperatorGroup
metadata:
  name: local-operator-group
  namespace: local-storage
spec:
  targetNamespaces:
    - local-storage
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: local-storage
spec:
  channel: "{product-version}" (1)
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

(1) 此字段可以被编辑，以匹配选择的 OpenShift Container Platform 的发行版本。

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

```
$ oc apply -f 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. 检查是否创建了所有必需的 Pod:

```
$ oc -n local-storage get pods
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 **local-storage** 项目中是否有 Local Storage Operator:

```
$ oc get csvs -n local-storage
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。

3.12.2. 置备本地卷

无法通过动态置备来创建本地卷。相反，PersistentVolume 必须由 Local Storage Operator 创建。此置备程序会在定义的资源中指定的路径上查找任意设备，包括文件系统和块卷。

先决条件

- 安装了 Local Storage Operator。
- 本地磁盘已附加到 OpenShift Container Platform 节点。

流程

1. 创建本地卷资源。这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 StorageClass 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: "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"
      volumeMode: Filesystem (3)
      fsType: xfs (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。这个示例使用从 **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。

匹配的磁盘。

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

4 第一次挂载本地卷时所创建的文件系统。

5 包含要从中选择的本地存储设备列表的路径。

6 将这个值替换为您的到 LocalVolume 资源的实际本地磁盘文件路径，如 **/dev/xvdg**。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。

例如：Block

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "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"
      volumeMode: Block 3
      devicePaths: 4
        - /path/to/device 5
```

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。这个示例使用从 **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 定义本地卷类型的卷模式，可以是 **Filesystem** 或 **Block**。

4 包含要从中选择的本地存储设备列表的路径。

5 将这个值替换为您的到 LocalVolume 资源的实际本地磁盘文件路径，如 **/dev/xvdg**。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。

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

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

3. 确定已创建置备程序，并且创建了相应的 DaemonSet：

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

```
NAME                                      READY   STATUS    RESTARTS   AGE
pod/local-disks-local-provisioner-h97hj   1/1     Running   0          46m
pod/local-disks-local-provisioner-j4mnn   1/1     Running   0          46m
pod/local-disks-local-provisioner-kbdnx   1/1     Running   0          46m
pod/local-disks-local-diskmaker-ldldw     1/1     Running   0          46m
pod/local-disks-local-diskmaker-lvrv4     1/1     Running   0          46m
pod/local-disks-local-diskmaker-phxdq     1/1     Running   0          46m
pod/local-storage-operator-54564d9988-vxvhx   1/1     Running   0          47m

NAME                             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/local-storage-operator   ClusterIP   172.30.49.90    <none>        60000/TCP   47m

NAME                                     DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/local-disks-local-provisioner   3         3         3       3
3           <none>          46m
daemonset.apps/local-disks-local-diskmaker     3         3         3       3
3           <none>          46m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/local-storage-operator   1/1     1            1           47m

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/local-storage-operator-54564d9988   1         1         1       47m
```

查看需要的和当前的 DaemonSet 进程数。如果需要的数目是 **0**，这表示标签选择器无效。

4. 确认创建了 PersistentVolume：

```
$ oc get pv

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 PersistentVolume 的 **fsType** 或 **volumeMode**，因为这样做可能会导致破坏操作。

3.12.3. 创建本地卷 PersistentVolumeClaim

必须静态创建本地卷作为 PersistentVolumeClaim (PVC)，才能被 Pod 访问。

前提条件

- PersistentVolume 已通过本地卷置备程序创建。

流程

1. 使用对应的 StorageClass 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 StorageClass 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
```

3.12.4. 附加本地声明

本地卷映射到 PersistentVolumeClaim (PVC) 后，可在资源内指定该本地卷。

先决条件

- 同一命名空间中存在 PVC。

流程

1. 在资源规格中包含定义的声明。以下示例在 Pod 内声明 PVC：

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

1. 要挂载的卷的名称。
2. 卷在 Pod 内的挂载路径。
3. 要使用的现有 PVC 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
```

3.12.5. 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: "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。

定义的容限度将传递给生成的 DaemonSet，允许为包含指定污点的节点创建 diskmaker 和 provisioner Pod。

3.12.6. 删除 Local Storage Operator 资源

3.12.6.1. 删除本地卷

有时候，必须要删除本地卷。虽然移除 LocalVolume 资源中的条目并且删除 PersistentVolume 通常已经足够，但若您要重新使用同一设备路径或者使其由不同的 StorageClass 进行管理，则需要额外的步骤。

警告

以下流程涉及以 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。

前提条件

- PersistentVolume 必须处于 **Released** 或 **Available** 状态。

警告

删除仍在使用中的 PersistentVolume 可能会导致数据丢失或崩溃。

流程

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

   ```
   $ oc edit localvolume <name> -n 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 PersistentVolume。

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

3. 删除节点上的所有符号链接。

a. 在节点上创建一个调试 Pod：

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

b. 将您的根目录改为主机：

```
$ chroot /host
```

c. 前往包含本地卷符号链接的目录。

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

1 用于创建本地卷的 StorageClass 名称。

d. 删除归属于已移除设备的符号链接。

```
$ rm <symlink>
```

3.12.6.2. 卸载 Local Storage Operator

要卸载 Local Storage Operator，您必须删除 Operator 以及 **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. 删除项目中的所有本地卷资源：

```
$ oc delete localvolume --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 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。当这些卷不再被使用后，运行以下命令删除它们：

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

4. 删除 **local-storage** 项目：

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

3.13. 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)

3.13.1. 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
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
```

1. 如前面对 PV 提供的一样， **accessModes** 不会强制实现安全控制，而是作为标签来把一个 PV 和一个 PVC 进行匹配。
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
NAME         STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-claim1   Bound     pv0001    5Gi        RWO            gp2            2m
```

3.13.2. 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。

3.13.3. 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。

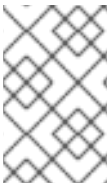
NFS 服务器中的 **/etc/exports** 文件包含可访问的 NFS 目录。目标 NFS 目录有 POSIX 拥有者和组群 ID。OpenShift Container Platform NFS 插件使用相同的 POSIX 所有者权限及在导出的 NFS 目录中找到的权限挂载容器的 NFS 目录。然而，容器实际运行时所使用的 UID 与 NFS 挂载的所有者的 UID 不同。这是所需的行为。

例如，目标 NFS 目录在 NFS 服务器中，如下所示：

```
$ ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs

$ id nfsnobody
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** 。

3.13.3.1. 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 定义的一个 GID 阵列。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）。

3.13.3.2. 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**，以及一个适用于 Pod 定义中所有容器的 Pod 的 **securityContext**。

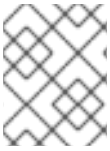
2 **65534** 是 **nfsnobody** 用户。

假设 **default** 项目及 **restricted** SCC，Pod 请求的用户 ID **65534** 不被允许，因此 Pod 将失败。Pod 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）。

3.13.3.3. SELinux

默认情况下，SELinux 不允许从 Pod 写入远程 NFS 服务器。NFS 卷会被正确挂载，但只读。

要允许写入远程 NFS 服务器，请遵循以下步骤。

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
```

3.13.3.4. 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 3 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）。

3.13.4. 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。

3.13.5. 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`

3.14. OPENSHIFT CONTAINER STORAGE

Red Hat OpenShift Container Storage 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 Container Storage 与 OpenShift Container Platform 完全集成，用于部署、管理和监控。

Red Hat OpenShift Container Storage 提供自己的文档库。完整的 Red Hat OpenShift Container Storage 文档包括在 https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/4.3/

如果您要寻找 Red Hat OpenShift Container Storage 的相关信息	请参阅以下 Red Hat OpenShift Container Storage 文档：
新的、已知的问题、显著的程序错误修复以及技术预览	Red Hat OpenShift Container Storage 4.3 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 Red Hat OpenShift Container Storage 4.3 部署
在现有 OpenShift Container Platform 集群中部署 Red Hat OpenShift Container Storage 4.3	部署 Red Hat OpenShift Container Storage 4.3
管理 Red Hat OpenShift Container Storage 4.3 集群	管理 Red Hat OpenShift Container Storage 4.3
监控 Red Hat OpenShift Container Storage 4.3 集群	监控 Red Hat OpenShift Container Storage 4.3
将 OpenShift Container Platform 集群从版本 3 迁移到版本 4	迁移

3.15. 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。

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。

其他参考资源

- VMware vSphere

3.15.1. 动态置备 VMware vSphere 卷

动态置备 VMware vSphere 卷是推荐的方法。

3.15.2. 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 StorageClass 动态置备这些卷。

3.15.2.1. 使用 UI 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

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** StorageClass。
   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**，以创建 PersistentVolumeClaim 并生成一个 PersistentVolume。

3.15.2.2. 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 PersistentVolumeClaim 的唯一名称。

(2) PersistentVolumeClaim 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) PersistentVolumeClaim 的大小。

2. 从文件创建 PersistentVolumeClaim：

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

3.15.3. 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 PersistentVolume。使用 PersistentVolume 对象定义创建一个 **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。PersistentVolumeClaim 或 Pod 通过这个名称来识别它。

(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：

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

4. 创建一个映射到您在前一步中创建的 PersistentVolume 的 PersistentVolumeClaim：创建包括 PVC 对象定义的一个 **pv1.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 PersistentVolumeClaim 的唯一名称。
2. PersistentVolumeClaim 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. PersistentVolumeClaim 的大小。
4. 已存在的 PersistentVolume 的名称。

5. 从文件创建 PersistentVolumeClaim：

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

3.15.3.1. 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 4 章 扩展持久性卷

4.1. 启用卷扩展支持

在扩展持久性卷之前，StorageClass 的 **allowVolumeExpansion** 字段需要设为 **true**。

流程

- 编辑 StorageClass，添加 **allowVolumeExpansion** 属性。下面的例子在 StorageClass 配置的底部添加了一行。

```
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。

4.2. 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 不附带任何 CSI 驱动程序。建议您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。

OpenShift Container Platform 4.3 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详情，请参阅 https://access.redhat.com/support/offerings/techpreview/。

4.3. 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 的 **VolumeExpansion** 被设置为 **true**。

流程

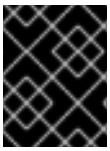
- 要在 FlexVolume 插件中使用 resizing 功能，您必须使用以下方法实现 **ExpandableVolumePlugin** 接口：

  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 master 节点上安装 FlexVolume 插件，所以不支持 FlexVolume 的 control-plane 扩展。

4.4. 通过文件系统扩展 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 的 **VolumeExpansion** 需要设置为 **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**。以下命令用来检查条件：

```
$ oc describe pvc <pvc_name>
```

3. 当云供应商对象完成重新定义大小时，持久性卷对象中的 **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。

4.5. 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 5 章 动态置备

5.1. 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**）或者存储管理员（**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。

5.2. 可用的动态部署插件

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**	
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。
AWS Elastic File System (EFS)		动态置备通过 EFS provisioner pod 实现，而不是通过置备程序插件实现。
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** ServiceAccount 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。

5.3. 定义 STORAGECLASS

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，ClusterStorageOperator 可能会安装一个默认的 StorageClass。这个 StorageClass 由操作员拥有和控制。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 StorageClass。

以下小节介绍了 StorageClass 的基本对象定义，以及每个支持的插件类型的具体示例。

5.3.1. 基本 StorageClass 对象定义

以下介绍了用来配置一个 StorageClass 所需的参数和默认值。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: gp2 (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）StorageClass 的名称。

(4) （可选）StorageClass 的注释

(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。

5.3.2. StorageClass 注解（annotations）

如需把一个 StorageClass 设置为默认在集群范内有效，把以下注解添加到 StorageClass 的元数据中：

```
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"
...
```

这将使任何没有指定特定卷的 PVC 通过默认的 StorageClass 被自动置备。

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。

如需设置一个 StorageClass 的描述，把以下注解添加到 StorageClass 的元数据中：

```
kubernetes.io/description: My StorageClass Description
```

例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My StorageClass Description
...
```

5.3.3. RHOSP Cinder 对象定义

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast 1
  availability: nova 2
  fsType: ext4 3
```

1. 在 Cinder 中创建的卷类型。默认为空。
2. 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。
3. 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**。

5.3.4. AWS Elastic Block Store (EBS) 对象定义

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 1
  iopsPerGB: "10" 2
  encrypted: "true" 3
  kmsKeyId: keyvalue 4
  fsType: ext4 5
```

1. （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 。
2. （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 。
3. （可选）是否加密 EBS 卷。有效值为 **true** 或者 **false**。
4. （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 。
5. （可选）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**。

5.3.5. Azure Disk 对象定义

azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name 1
  storageaccounttype: Standard_LRS 2
  kind: Dedicated 3
```

-

1. Azure 存储帐户名称。这必须与集群位于同一个资源组中。如果指定了存储帐户，则忽略 **location**。如果没有指定存储帐户，则在与集群相同的资源组中创建一个新的存储帐户。如果您要指定 **StorageAccount**，则 **kind** 的值必须是 **Dedicated**。

2. 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。

3. 可能的值有 **Shared**（默认）、**Dedicated** 和 **Managed**。

    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 必须对存储帐户有写入权限。

    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。

5.3.6. Azure File 对象定义

Azure File StorageClass 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：

```
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 ClusterRole 名称。

2. 将 ClusterRole 添加到 ServiceAccount：

```
$ 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：

-

```
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. StorageClass 的名称。PersistentVolumeClaim 使用这个 StorageClass 来置备相关的 PersistentVolume。
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 StorageClass 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。

5.3.6.1. 使用 Azure File 时的注意事项

默认 Azure File StorageClass 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 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 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。

5.3.7. GCE PersistentDisk (gcePD) 对象定义

gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 1
  replication-type: none
```

1. 选择 **pd-standard** 或 **pd-ssd**。默认为 **pd-ssd**。

5.3.8. VMware vSphere 对象定义

vsphere-storageclass.yaml

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

1. 有关在 OpenShift Container Platform 中使用 VMware vSphere 的详情，请参阅 VMware vSphere 文档。
2. **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**。

5.3.9. Red Hat OpenShift Container Storage 对象定义

使用 Red Hat OpenShift Container Storage 时，当从 Operator Hub 部署 Red Hat OpenShift Container Storage 4.3 时，会创建动态卷置备的存储类，如 验证是否创建并列出了存储类所述。

5.4. 修改默认的 STORAGECLASS

如果使用 AWS，请按照以下方法更改默认 StorageClass。这个方法假设您定义了两个 StorageClasses，**gp2** 和 **standard**，您想要将默认存储类从 **gp2** 改为 **standard**。

1. 列出 StorageClass:

```
$ oc get storageclass

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

(1) **（默认）** 指定默认 StorageClass。

2. 为默认 StorageClass 将注解 **storageclass.kubernetes.io/is-default-class** 的值改为 **false** ：

```
$ oc patch storageclass gp2 -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** 来使另外一个 StorageClass 作为默认。

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

4. 确认更改：

```
$ oc get storageclass

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