Red Hat build of MicroShift 4.16

存储

配置和管理集群存储

Last Updated: 2024-06-28

Red Hat build of MicroShift 4.16 存储

配置和管理集群存储

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本文档提供有关为 MicroShift 使用存储的信息。

目录

第 1 章 存储概述 **3**
1.1. 存储类型 3

第 2 章 了解临时存储 **4**
2.1. 概述 4
2.2. 临时存储的类型 4
2.3. 临时存储管理 4
2.4. 监控临时存储 6

第 3 章 通用临时卷 **7**
3.1. 概述 7
3.2. 生命周期和持久性卷声明 7
3.3. 安全性 7
3.4. 持久性卷声明命名 7
3.5. 创建通用临时卷 8

第 4 章 了解持久性存储 **9**
4.1. 持久性存储概述 9
4.2. 其他资源 9
4.3. 卷和声明的生命周期 9
4.4. 持久性卷（PV） 12
4.5. 持久性卷声明 (PVC) 13
4.6. 使用 FSGROUP 减少 POD 超时 15

第 5 章 扩展持久性卷 **16**
5.1. 扩展 CSI 卷 16
5.2. 扩展本地卷 16
5.3. 使用文件系统扩展持久性卷声明（PVC） 16
5.4. 在扩展卷失败时进行恢复 17

第 6 章 使用 LVMS 插件进行动态存储 **18**
6.1. LVMS 系统要求 18
6.2. LVMS 部署 18
6.3. 配置在 LVM 存储中使用的设备大小的限制 19
6.4. 创建 LVMS 配置文件 19
6.5. 基本 LVMS 配置示例 19
6.6. 使用 LVMS 20

第 7 章 使用卷快照 **23**
7.1. 关于 LVM 精简卷 23
7.2. 卷快照类 25
7.3. 关于卷快照 26
7.4. 关于 LVM 卷克隆 32

第 8 章 使用 KUBE STORAGE VERSION MIGRATOR 进行存储迁移 **33**
8.1. 发出存储迁移请求 33

第 1 章 存储概述

MicroShift 支持多种类型的存储，包括内部存储和云提供商。您可以在红帽构建的 MicroShift 集群中管理持久性和非持久性数据的容器存储。

1.1. 存储类型

MicroShift 存储广泛分为两类，即临时存储和持久存储。

1.1.1. 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。若要读取临时存储的详细信息，请单击 了解临时存储。

1.1.2. 持久性存储

容器中部署的有状态应用需要持久存储。MicroShift 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。如需持久性存储详情，请阅读 了解持久性存储。

1.1.3. 动态存储置备

使用动态置备可让您按需创建存储卷，消除预置备存储的需求。有关动态置备如何在红帽构建的 MicroShift 中工作的更多信息，请参阅动态置备。

第 2 章 了解临时存储

临时存储是非结构化的，临时的。它通常与不可变应用程序一起使用。本指南讨论临时存储对 MicroShift 的工作原理。

2.1. 概述

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

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

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

与持久性卷不同，临时存储是非结构化的，空间在节点上运行的所有 pod 之间共享，系统使用其他用途，以及红帽构建的 MicroShift。临时存储框架允许 Pod 指定其临时本地存储需求。它还允许红帽构建的 MicroShift 保护节点不受过度使用本地存储的影响。

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

2.2. 临时存储的类型

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

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

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

2.3. 临时存储管理

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

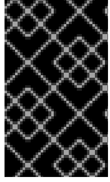
您可以通过指定请求和限值来管理本地临时存储。pod 中的每个容器可以指定以下内容：

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

2.3.1. 临时存储限制和请求单元

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

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



重要

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

2.3.2. 临时存储请求和限值示例

以下示例配置文件显示了一个具有两个容器的 pod：

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

使用配额和限值的临时存储配置示例

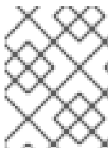
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        ephemeral-storage: "2Gi" (1)
      limits:
        ephemeral-storage: "4Gi" (2)
    volumeMounts:
    - name: ephemeral
      mountPath: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

```
volumes:
  - name: ephemeral
    emptyDir: {}
```

1. 容器请求本地临时存储。
2. 本地临时存储的容器限制。

2.3.3. 临时存储配置会影响 pod 驱除

pod 规格中的设置会在 kubelet 驱除 pod 时产生影响。在容器级别，因为第一个容器设置了资源限值，kubelet 驱除管理器会测量此容器的磁盘用量，并在容器存储使用量超过其限制(4GiB)时驱除 pod。如果总用量超过总体 pod 存储限制(8GiB)，kubelet 驱除管理器也会标记驱除 pod。



注意

此策略严格用于 **emptyDir** 卷，不适用于持久性存储。您可以指定 pod 的 **priorityClass** 来免 pod 驱除。

2.4. 监控临时存储

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

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

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

输出显示 **/var/lib** 中的临时存储使用情况：

输出示例

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

第 3 章 通用临时卷

了解 MicroShift 的临时卷，包括其生命周期、安全性和命名。

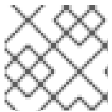
3.1. 概述

通用临时卷是临时卷类型，可以由支持持久性卷和动态置备的所有存储驱动程序提供。通用临时卷与 **emptyDir** 卷类似，它们为从头开始数据提供每个 pod 目录，这通常在置备后为空。

通用临时卷在 pod 规格中内联指定，并遵循 pod 的生命周期。它们与 pod 一起创建和删除。

通用临时卷具有以下功能：

- 存储可以是本地的或者网络附加存储。
- 卷可以有 pod 无法超过的固定大小。
- 根据驱动程序和参数，卷可能有一些初始数据。
- 支持卷上的典型的操作，假设驱动程序支持它们，包括快照、克隆、调整大小和存储容量跟踪。



注意

通用临时卷不支持离线快照和调整大小。

3.2. 生命周期和持久性卷声明

卷声明的参数允许在 pod 的卷源内。支持声明(PVC)的标签、注解和整个字段。当创建这样的 pod 时，临时卷控制器随后会在与 pod 相同的命名空间中创建实际的 PVC 对象（来自 *创建通用临时卷* 流程中显示的模板），并确保在 pod 被删除时删除 PVC。这会以两种方式之一触发卷绑定和置备：

- 另外，如果存储类使用即时卷绑定。
  通过立即绑定，调度程序被强制选择在卷可用后有权访问的节点。
- 当 pod 放入节点时（**WaitForFirstConsumervolume** 绑定模式）。
  建议这个卷绑定选项用于通用临时卷，因为调度程序可以为 pod 选择适当的节点。

就资源限制而言，具有通用临时存储的 pod 是提供该临时存储的 PVC 的所有者。当 pod 被删除时，Kubernetes 垃圾回收器会删除 PVC，然后，后者通常会触发删除卷，因为存储类的默认重新声明策略是删除卷。您可以使用一个带有保留策略的存储类创建 quasi-ephemeral 本地存储：存储会活跃 pod，在这种情况下，您必须确保卷清理单独发生。虽然这些 PVC 存在，它们可以像任何其他 PVC 一样使用。特别是，可以在卷克隆或快照中被引用为数据源。PVC 对象也保存卷的当前状态。

3.3. 安全性

您可以启用通用临时卷功能，以便可以创建 pod 的用户也可以间接创建持久性卷声明 (PVC)。即使这些用户没有直接创建 PVC 的权限，此功能也可以正常工作。集群管理员必须了解这一点。如果这不适用于您的安全模型，请使用准入 Webhook 来拒绝对象，如具有通用临时卷的 pod。

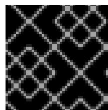
PVC 的一般命名空间配额仍适用，因此即使允许用户使用此新机制，它们也无法使用它来绕过其他策略。

3.4. 持久性卷声明命名

自动创建的持久性卷声明(PVC)由 pod 名称和卷名称的组合命名，中间带有连字符(-)。这种命名惯例还引入了不同 pod 之间以及 pod 和手动创建 PVC 之间潜在的冲突。

例如，**pod-a** 带有卷 **scratch**，**pod** 带有卷 **a-scratch**，它们都使用相同的 PVC 名称 **pod-a-scratch**。

检测到这样的冲突，如果为 pod 创建，PVC 仅用于临时卷。此检查基于所有权关系。现有 PVC 不会被覆盖或修改，但这不会解决冲突。如果没有正确的 PVC，pod 无法启动。



重要

在在同一命名空间中命名 pod 和卷时要小心，以便不会发生命名冲突。

3.5. 创建通用临时卷

流程

1. 创建 **pod** 对象定义，并将其保存到文件中。
2. 在文件中包括通用临时卷信息。

my-example-pod-with-generic-vols.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-app
spec:
  containers:
    - name: my-frontend
      image: busybox:1.28
      volumeMounts:
      - mountPath: "/mnt/storage"
        name: data
      command: [ "sleep", "1000000" ]
  volumes:
    - name: data (1)
      ephemeral:
        volumeClaimTemplate:
          metadata:
            labels:
              type: my-app-ephvol
          spec:
            accessModes: [ "ReadWriteOnce" ]
            storageClassName: "topolvm-provisioner"
            resources:
              requests:
                storage: 1Gi
```

(1) 通用临时卷声明。

第 4 章 了解持久性存储

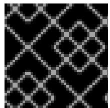
管理存储与管理计算资源不同。MicroShift 使用 Kubernetes 持久性卷 (PV) 框架来允许集群管理员为集群提供持久性存储。开发者可以使用持久性卷声明 (PVC) 来请求 PV 资源而无需具体了解底层存储基础架构。

4.1. 持久性存储概述

PVC 特定于一个命名空间，开发人员创建并使用它作为使用 PV 的方法。PV 资源本身并不限定于任何单一命名空间；它们可以在整个红帽构建的 MicroShift 集群间共享，并从任何命名空间共享。在 PV 绑定到 PVC 后，就不会将 PV 绑定到额外的 PVC。这会影响到一个命名空间的 PV 的影响。

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 对象捕获存储实现的详情，即 LVM、主机文件系统（如 hostpath 或原始块设备）。



重要

存储的高可用性功能由底层的存储架构提供。

与 **PersistentVolumes** 一样，**PersistentVolumeClaims** (PVCs) 是 API 对象，代表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。OpenShift Container Platform 支持的访问模式也可以在红帽构建的 MicroShift 中定义。但是，因为红帽构建的 MicroShift 不支持多节点部署，所以只有 ReadWriteOnce (RWO)。

4.2. 其他资源

- 持久性存储访问模式

4.3. 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。

4.3.1. 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。

4.3.2. 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，红帽构建的 MicroShift 将会把 PVC 绑定到匹配所有其他标准的最小 PV。

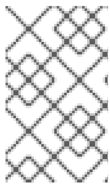
如果匹配的卷不存在，或者相关的置备程序无法创建所需的存储，则请求将会处于未绑定的状态。当出现了匹配的卷时，相应的声明就会与其绑定。例如：在一个集群中有多个手动置备的 50Gi 卷。它们无法和

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。

4.3.3. 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？

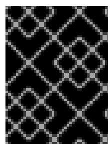
4.3.4. 释放持久性卷

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。

4.3.5. 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**。

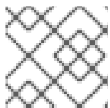
- **Retain** 策略可为那些支持它的卷插件手动重新声明资源。
- **Recycle** 策略在从其请求中释放后，将卷重新放入到未绑定的持久性卷池中。



重要

在红帽构建的 MicroShift 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** 策略删除红帽构建的 MicroShift 以及外部基础架构（如 Amazon Elastic Block Store (Amazon EBS)或 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。

4.3.6. 手动重新声明持久性卷

删除持久性卷声明 (PVC) 时，底层逻辑卷会根据 **reclaimPolicy** 进行处理。

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。

4.3.7. 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。

4.4. 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。

4.4.1. 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。

4.4.2. 支持的访问模式

LVMS 是红帽构建的 MicroShift 唯一支持的 CSI 插件。内置在 OpenShift Container Platform 的 hostPath 和 LV 也支持 RWO。

4.4.3. 阶段

卷可以处于以下几个阶段：

表 4.1. 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。

阶段	描述
Released	以前使用这个卷的声明已被删除，但该资源还没有被集群重新声明。
Failed	卷的自动重新声明失败。

您可以运行以下命令来查看绑定到 PV 的 PVC 名称：

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

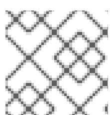
4.4.3.1. 挂载选项

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: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: topolvm-provisioner
mountOptions:
  - uid=1500
  - gid=1500
parameters:
  csi.storage.k8s.io/fstype: xfs
provisioner: topolvm.io
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```



注意

mountOptions 不会被验证。不正确的值将导致挂载失败，并将事件记录到 PVC。

其他资源

- 常用挂载选项

4.5. 持久性卷声明 (PVC)

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

PersistentVolumeClaim 对象定义示例

```
kind: PersistentVolumeClaim
apiVersion: v1
```

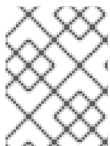
```
metadata:
  name: myclaim 1
spec:
  accessModes:
    - ReadWriteOnce 2
  resources:
    requests:
      storage: 8Gi 3
  storageClassName: gold 4
status:
  ...
```

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

4.5.1. 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。

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。

4.5.2. 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。

4.5.3. 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。

4.5.4. 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）。

4.6. 使用 FSGROUP 减少 POD 超时

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

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

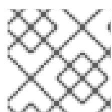
fsGroupChangePolicy 定义在 pod 中公开卷之前更改卷的所有权和权限的行为。此字段仅适用于支持 **fsGroup**- 控制的所有权和权限。此字段有两个可能的值：

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

fsGroupChangePolicy 示例

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

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



注意

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

第 5 章 扩展持久性卷

了解如何在 MicroShift 中扩展持久性卷。

5.1. 扩展 CSI 卷

您可以在存储卷被创建后，使用 Container Storage Interface (CSI) 来扩展它们。

CSI 卷扩展不支持以下内容：

- 在扩展卷失败时进行恢复
- 缩小

先决条件

- 底层 CSI 驱动程序支持调整大小。
- 使用动态置备。
- 控制 **StorageClass** 对象的 **allowVolumeExpansion** 被设置为 **true**。如需更多信息，请参阅"启用卷扩展支持"。

流程

1. 对于持久性卷声明(PVC)，将 **.spec.resources.requests.storage** 设置为所需的新大小。
2. 观察 PVC 的 **status.conditions** 字段来查看调整大小是否完成。红帽构建的 MicroShift 会在扩展过程中为 PVC 添加 **Resizing** 条件，该条件会在扩展完成后删除。

5.2. 扩展本地卷

您可以使用本地存储操作器(LSO)手动扩展持久性卷(PV)和持久性卷声明(PVC)。

流程

1. 扩展底层设备。确定这些设备中提供了适当的容量。
2. 通过编辑 PV 的 **.spec.capacity** 字段来更新对应的 PV 对象以匹配新设备大小。
3. 对于用于将 PVC 绑定到 PVet 的存储类，设置 **allowVolumeExpansion:true**。
4. 对于 PVC，将 **.spec.resources.requests.storage** 设置为与新大小匹配。

根据需要，kubelet 应该自动扩展卷上的底层文件系统，并更新 PVC 的 status 字段来反映新的大小。

5.3. 使用文件系统扩展持久性卷声明（PVC）

根据需要重新定义文件系统大小的卷类型扩展 PVC，如 GCE Persistent Disk 卷(gcePD)、AWS Elastic Block Store EBS (EBS)和 Cinder，分为两个步骤。首先，扩展云供应商中的卷对象。其次，扩展节点上的文件系统。

只有在使用这个卷启动新的 pod 时，才会在该节点中扩展文件系统。

先决条件

- 控制 **StorageClass** 对象必须将 **allowVolumeExpansion** 设置为 **true**。

流程

1. 通过编辑 **spec.resources.requests**来修改 PVC 并请求一个新的大小。例如，以下命令将 **ebs** PVC 扩展至 8 Gi：

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

(1) 将 **spec.resources.requests** 更新至更大的 PVC 来扩展 PVC。

2. 重新定义云供应商对象大小后， PVC 被设置为 **FileSystemResizePending**。输入以下命令检查条件：

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

3. 当云供应商对象完成重新定义大小时，**PersistentVolume** 对象中的 **PersistentVolume.Spec.Capacity** 会显示新请求的大小。此时，您可从 PVC 创建或重新创建新 Pod 来完成文件系统大小调整。当 Pod 运行后，新请求的大小就可用，同时 **FileSystemResizePending** 条件从 PVC 中删除。

5.4. 在扩展卷失败时进行恢复

如果扩展底层存储失败，红帽构建的 MicroShift 管理员可以手动恢复 PVC 的状态，并取消改变大小的请求。否则，控制器会持续重试大小的请求。

流程

1. 把与 PVC 进行绑定的 PV 的 reclaim 策略设为 **Retain**。编辑 PV，把 **persistentVolumeReclaimPolicy** 的值改为 **Retain**。
2. 删除 PVC。
3. 手动编辑 PV 并从 PV specs 中删除 **claimRef** 条目，以确保新创建的 PVC 可以绑定到标记为 **Retain** 的 PV。这会将 PV 标记为 **Available**。
4. 以较小的大小，或底层存储架构可以分配的大小，重新创建 PVC。
5. 将 PVC 的 **volumeName** 值设为 PV 的名称。这使 PVC 只会绑定到置备的 PV。
6. 恢复 PV 上的 reclaim 策略。

第 6 章 使用 LVMS 插件进行动态存储

MicroShift 启用动态存储置备，可立即与逻辑卷管理器存储(LVMS) 容器存储 (CSI) 供应商一起使用。LVMS 插件是 TopoLVM 的 Red Hat downstream 版本，它是一个 CSI 插件，用于管理 Kubernetes 的逻辑卷管理(LVM)逻辑卷(LV)。

LVMS 为带有适当配置的持久性卷声明(PVC)的容器工作负载置备新的 LVM 逻辑卷。每个 PVC 都引用一个存储类，它代表主机节点上的 LVM 卷组(VG)。LV 仅针对调度的 pod 置备。

6.1. LVMS 系统要求

在 MicroShift 中使用 LVMS 需要以下系统规格：

6.1.1. 卷组名称

如果您没有在放置在 **/etc/microshift/** 目录中的 **lvmd.yaml** 文件中配置 LVMS，MicroShift 会尝试通过运行 **vgs** 命令来动态分配默认卷组(VG)。

- MicroShift 在只找到一个 VG 时分配默认 VG。
- 如果存在多个 VG，则名为 **microshift** 的 VG 被分配为默认值。
- 如果名为 **microshift** 的 VG 不存在，则不会部署 LVMS。

如果 MicroShift 主机上没有卷组，则 LVMS 会被禁用。

如果要使用特定的 VG，则必须将 LVMS 配置为选择该 VG。您可以在配置文件中更改 VG 的默认名称。详情请查看本文档的"配置 LVMS"部分。

您可以在配置文件中更改 VG 的默认名称。详情请查看本文档的"配置 LVMS"部分。

MicroShift 启动后，您可以更新 **lvmd.yaml** 以包含或删除 VG。要实现更改，您必须重启 MicroShift。如果删除了 **lvmd.yaml**，MicroShift 会尝试再次查找默认 VG。

6.1.2. 卷大小递增

LVMS 以 1GB (GB) 为单位递增的形式置备存储。存储请求将向上舍入到最接近的 GB。当 VG 的容量小于 1 GB 时，**PersistentVolumeClaim** 会注册一个 **ProvisioningFailed** 事件，例如：

输出示例

```
Warning  ProvisioningFailed    3s (x2 over 5s)  topolvm.cybozu.com_topolvm-controller-858c78d96c-xttzp_0fa83aef-2070-4ae2-bcb9-163f818dcd9f failed to provision volume with
StorageClass "topolvm-provisioner": rpc error: code = ResourceExhausted desc = no enough space
left on VG: free=(BYTES_INT), requested=(BYTES_INT)
```

6.2. LVMS 部署

在 MicroShift 启动后，LVMS 会在 **openshift-storage** 命名空间中自动部署到 openshift-storage 命名空间中的集群。

LVMS 使用 **StorageCapacity** 跟踪来确保在请求的存储大于卷组可用存储时不会调度带有 LVMS PVC 的 pod。有关 **StorageCapacity** 跟踪的更多信息，请参阅 Storage Capacity。

6.3. 配置在 LVM 存储中使用的设备大小的限制

使用 LVM 存储配置可用于置备存储的设备大小的限制如下：

- 您可以置备的总存储大小受底层逻辑卷管理器(LVM)精简池的大小以及过度置备因素的限制。
- 逻辑卷的大小取决于物理扩展(PE)和逻辑扩展(LE)的大小。
    - 您可以在创建物理和虚拟设备的过程中定义 PE 和 LE 的大小。
    - 默认的 PE 和 LE 大小为 4 MB。
    - 如果增加 PE 的大小，LVM 的最大大小由内核限值和您的磁盘空间决定。
    - 使用默认的 PE 和 LE 大小的 Red Hat Enterprise Linux (RHEL) 9 的大小限制为 8 EB。
    - 以下是您可以为每个文件系统类型请求的最小存储大小：
        - 块设备: 8 MiB
        - **XFS**: 300 MiB
        - **ext4**: 32 MiB

6.4. 创建 LVMS 配置文件

当 MicroShift 运行时，会使用 **/etc/microshift/lvmd.yaml** 中的 LVMS 配置（如果提供）。您必须将您创建的任何配置文件放在 **/etc/microshift/** 目录中。

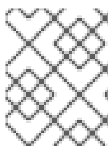
流程

- 要创建 **lvmd.yaml** 配置文件，请运行以下命令：

```
$ sudo cp /etc/microshift/lvmd.yaml.default /etc/microshift/lvmd.yaml
```

6.5. 基本 LVMS 配置示例

MicroShift 支持通过 LVM 配置，并允许您指定自定义卷组、精简卷置备参数和保留未分配的卷组空间。您可以随时编辑您创建的 LVMS 配置文件。编辑该文件后，您必须重启 MicroShift 以部署配置更改。



注意

如果需要执行卷快照，则必须在 **lvmd.conf** 文件中使用精简配置。如果不需要执行卷快照，可以使用 thick 卷。

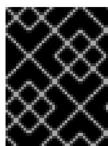
以下 **lvmd.yaml** 示例文件显示了基本的 LVMS 配置：

LVMS 配置示例

```
socket-name: (1)
device-classes: (2)
  - name: "default" (3)
```

```
volume-group: "VGNAMEHERE" 4
spare-gb: 0 5
default: 6
```

1. 字符串.gRPC 的 UNIX 域套接字端点。默认为 '/run/lvmd/lvmd.socket'。
2. 每个 设备类 设置的映射列表。
3. 字符串.**device-class** 的名称。
4. 字符串.**device-class** 创建逻辑卷的组。
5. 64 位无符号整数。存储容量（以 GB 为单位）在卷组中未分配。默认为 **0**。
6. 布尔值.表示默认使用 **device-class**。默认值为 **false**。当它被设置为 **true** 时，必须在 YAML 文件值中输入至少一个值。



重要

竞争条件可防止 LVMS 在同时创建多个 PVC 时准确跟踪分配的空间，并为设备类保留 **spare-gb**。使用单独的卷组和逻辑卷类来防止存储高度动态工作负载相互影响。

6.6. 使用 LVMS

LVMS **StorageClass** 使用一个默认 **StorageClass** 部署。任何没有自动定义的 **.spec.storageClassName** 的 **PersistentVolumeClaim** 对象，都会有一个从默认 **StorageClass** 置备的 **PersistentVolume**。使用以下步骤将逻辑卷置备并挂载到 pod。

流程

- 要将逻辑卷置备并挂载到 pod，请运行以下命令：

```
$ cat <<EOF | oc apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-lv-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1G
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx
    command: ["/usr/bin/sh", "-c"]
    args: ["sleep", "1h"]
```

```
      volumeMounts:
      - mountPath: /mnt
        name: my-volume
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
    volumes:
      - name: my-volume
        persistentVolumeClaim:
          claimName: my-lv-pvc
EOF
```

6.6.1. 设备类

您可以通过在逻辑卷管理器存储(LVMS)配置中添加 **device-classes** 数组来创建自定义设备类。将数组添加到 **/etc/microshift/lvmd.yaml** 配置文件中。必须将单个设备类设置为默认值。您必须重启 MicroShift 才能使配置更改生效。

警告

当仍然连接到那个设备类的持久性卷或 **VolumeSnapshotContent** 对象时，删除设备类会破坏厚和精简置备。

您可以在 **device-classes** 阵列中定义多个设备类。这些类可以是厚和精简卷配置的组合。

混合 device-class 数组示例

```
socket-name: /run/topolvm/lvmd.sock
device-classes:
  - name: ssd
    volume-group: ssd-vg
    spare-gb: 0 (1)
    default: true
  - name: hdd
    volume-group: hdd-vg
    spare-gb: 0
  - name: thin
    spare-gb: 0
    thin-pool:
      name: thin
      overprovision-ratio: 10
    type: thin
    volume-group: ssd
  - name: striped
    volume-group: multi-pv-vg
```

```
spare-gb: 0
stripe: 2
stripe-size: "64"
lvcreate-options: (2)
```

1. 当您将备用容量设置为 **0** 以外的任何值时，可以比预期分配更多空间。

2. 要传递给 **lvcreate** 命令的额外参数，如 **--type=<type>**。MicroShift 和 LVMS 都验证 **lvcreate-options** 值。这些可选值按照 传递给 **lvcreate** 命令。确保此处指定的选项正确。

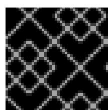
第 7 章 使用卷快照

集群管理员可以使用支持的 MicroShift 逻辑卷管理器存储(LVMS)容器存储(CSI)供应商来使用卷快照来帮助防止数据丢失。需要熟悉 持久性卷。

快照代表集群中特定时间点的存储卷状态。卷快照也可用于置备新卷。快照创建为只读逻辑卷(LV)，位于与原始数据相同的设备上。

集群管理员可以使用 CSI 卷快照完成以下任务：

- 为现有持久性卷声明(PVC)创建快照。
- 将卷快照备份到安全位置。
- 将卷快照恢复为不同的 PVC。
- 删除现有的卷快照。



重要

MicroShift 仅支持逻辑卷管理器存储(LVMS)插件 CSI 驱动程序。

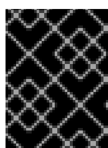
其他资源

- 了解持久性卷
- 配置和管理逻辑卷
- CSI 快照： **VolumeSnapshot** API
- VolumeSnapshot API 规格

7.1. 关于 LVM 精简卷

要使用创建卷快照或卷克隆等高级存储功能，您必须执行以下操作：

- 配置逻辑卷管理器存储(LVMS)供应商和集群。
- 在 RHEL for Edge 主机上置备逻辑卷管理器(LVM)精简池。
- 将 LVM 精简池附加到卷组。



重要

要创建 Container Storage Interface (CSI)快照，您必须在 RHEL for Edge 主机上配置精简卷。CSI 不支持卷缩小。

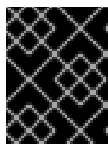
要使 LVMS 管理精简逻辑卷(LV)，必须在 **etc/lvmd.yaml** 配置文件中指定 thin-pool **device-class** 数组。允许多个精简池设备类。

如果使用设备类配置额外的存储池，则还需要存在额外的存储类，才能向用户和工作负载公开存储池。要在 thin-pool 上启用动态置备，集群中必须存在 **StorageClass** 资源。**StorageClass** 资源指定 **topolvm.io/ device-class** 参数中的源 device-class 数组。

为 thin-pool 指定单个设备类的 lvmd.yaml 文件示例

```yaml
socket-name: (1)
device-classes: (2)
  - name: thin (3)
    default: true
    spare-gb: 0 (4)
    thin-pool:
      name: thin
      overprovision-ratio: 10 (5)
    type: thin (6)
    volume-group: ssd (7)
```

1. 字符串.gRPC 的 UNIX 域套接字端点。默认为 **/run/lvmd/lvmd.socket**。
2. 每个 设备类 设置的映射列表。
3. 字符串.**device-class** 的唯一名称。
4. 64 位无符号整数。存储容量（以 GB 为单位）在卷组中未分配。默认为 **0**。
5. 如果您有多个共享同一池的精简置备设备，那么这些设备可以是过度配置的。过度置备需要一个浮点值 1 或更高。
6. 创建卷快照需要精简配置。
7. 字符串.**device-class** 创建逻辑卷的组。



重要

当同时创建多个 PVC 时，一个竞争条件可防止 LVMS 准确跟踪分配的空间，并为设备类保留存储容量。使用单独的卷组和逻辑卷类来防止存储高度动态工作负载相互影响。

其他资源

- 要在主机上创建精简池，请参阅创建和管理精简置备的卷
- 创建精简配置的逻辑卷
- 配置和管理精简置备的卷
- 存储类
- 存储设备类

7.1.1. 存储类

存储类提供选择设备类的工作负载层接口。MicroShift 支持以下存储类参数：

- **csi.storage.k8s.io/fstype** 参数选择文件系统类型。支持 **xfs** 和 **ext4** 文件系统类型。
- **topolvm.io/device-class** 参数是设备类的名称。如果没有提供设备类，则假定使用默认设备类。

多个存储类可以引用同一设备类。您可以为同一后备设备类（如 **xfs** 和 **ext4** 变体）提供不同的参数集合。

MicroShift 默认存储类资源示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" (1)
  name: topolvm-provisioner
parameters:
  "csi.storage.k8s.io/fstype": "xfs" (2)
provisioner: topolvm.io (3)
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer (4)
allowVolumeExpansion: (5)
```

1. 默认存储类的示例。如果 PVC 没有指定存储类，则会假定使用这个类。集群中只能有一个默认存储类。也支持没有分配给此注解的值。

2. 指定要在卷中置备的文件系统。选项为 "xfs" 和 "ext4"。

3. 标识哪些置备程序应该管理这个类。

4. 指定是否在客户端 pod 存在或立即置备卷。选项为 **WaitForFirstConsumer** 和 **Immediate**。建议 **WaitForFirstConsumer**，以确保只为可以调度的 pod 置备存储。

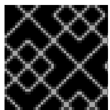
5. 指定从 **StorageClass** 置备的 PVC 是否允许扩展。MicroShift LVMS CSI 插件支持卷扩展，但如果这个值被设置为 **false**，则扩展会被阻止。

其他资源

- 定义存储类
- 存储设备类

7.2. 卷快照类

快照是 LVMS 支持的 CSI 存储功能。要启用动态快照，集群中必须至少有一个 **VolumeSnapshotClass** 配置文件。



重要

您必须启用精简逻辑卷才能生成逻辑卷快照。

VolumeSnapshotClass 配置文件示例

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: topolvm-snapclass
  annotations:
```

```
  snapshot.storage.kubernetes.io/is-default-class: "true" (1)
driver: topolvm.io (2)
deletionPolicy: Delete (3)
```

1. 决定在 **VolumeSnapshot** 没有指定任何 **VolumeSnapshotClass** 配置文件时要使用的 VolumeSnapshotClass 配置文件，这是用户对卷快照的请求。

2. 标识哪些快照置备程序应该由用户为此类管理卷快照的请求。

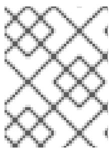
3. 决定在删除绑定 **VolumeSnapshot** 时是否保留或删除 **VolumeSnapshotContent** 对象和后备快照。有效值为 **Retain** 或 **Delete**。

其他资源

- OpenShift CSI 卷快照

7.3. 关于卷快照

您可以将卷快照与逻辑卷管理器(LVM)精简卷一起使用，以帮助防止在 MicroShift 集群中运行的应用程序丢失数据。MicroShift 只支持逻辑卷管理器存储(LVMS)容器存储接口(CSI)供应商。



注意

LVMS 只支持将存储类的 **volumeBindingMode** 设置为 **WaitForFirstConsumer**。此设置意味着在 pod 准备好挂载它前不会置备存储卷。

部署单个 pod 和 PVC 的工作负载示例

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-claim-thin
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: topolvm-provisioner-thin
---
apiVersion: v1
kind: Pod
metadata:
  name: base
spec:
  containers:
  - command:
     - nginx
     - -g
     - 'daemon off;'
    image: registry.redhat.io/rhel8/nginx-
122@sha256:908ebb0dec0d669caaf4145a8a21e04fdf9ebffbba5fd4562ce5ab388bf41ab2
```

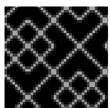
```
      name: test-container
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
          - ALL
      volumeMounts:
      - mountPath: /vol
        name: test-vol
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    volumes:
    - name: test-vol
      persistentVolumeClaim:
        claimName: test-claim-thin
EOF
```

7.3.1. 创建卷快照

要为 MicroShift 存储卷创建快照，您必须首先配置 RHEL for Edge 和集群。在以下示例中，源卷被挂载到的 pod 已被删除。删除 pod 可防止在快照创建过程中写入数据。确保在快照期间不会写入任何数据，这对创建可行的快照至关重要。

先决条件

- 用户对 MicroShift 集群具有 root 访问权限。
- MicroShift 集群正在运行。
- 设备类定义 LVM 精简池。
- **volumeSnapshotClass** 指定 **driver: topolvm.io**。
- 附加到源 PVC 的任何工作负载都会暂停或删除。这有助于避免数据崩溃。



重要

在创建快照时，必须停止对卷的所有写入。如果没有停止写入，您的数据可能会损坏。

流程

1. 使用以下步骤之一防止数据在快照期间写入卷：
    a. 运行以下命令，删除 pod 以确保在快照过程中不会写入卷：

    ```
    $ oc delete my-pod
    ```

    b. 在使用复制控制器管理的 pod 上将副本数扩展到零。将 count 设为零可防止在删除新 pod 时立即创建新 pod。
2. 在对卷的所有写入都停止后，运行类似于下例的命令。插入您自己的配置详情。

快照配置示例

```
# oc apply -f <<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot 1
metadata:
  name: <snapshot_name> 2
spec:
  volumeSnapshotClassName: topolvm-snapclass 3
  source:
    persistentVolumeClaimName: test-claim-thin 4
EOF
```

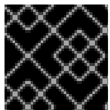
1. 创建 **VolumeSnapshot** 对象。
2. 您为快照指定的名称。
3. 指定 **VolumeSnapshotClass** 对象所需的名称。
4. 指定 **persistentVolumeClaimName** 或 **volumeSnapshotContentName**。在本例中，从名为 **test-claim-thin** 的 PVC 创建快照。

3. 运行以下命令，等待存储驱动程序完成创建快照：

```
$ oc wait volumesnapshot/<snapshot_name> --for=jsonpath\='{.status.readyToUse}=true'
```

后续步骤

1. 当 **volumeSnapshot** 对象处于 **ReadyToUse** 状态时，您可以将其恢复为将来 PVC 的卷。重启 pod 或将副本数向上扩展到所需的数量。
2. 创建卷快照后，您可以将源 PVC 重新挂载为新 pod。



重要

卷快照位于与原始数据相同的设备上。要将卷快照用作备份，请将快照移到安全位置。

7.3.2. 备份卷快照

在 MicroShift 集群上运行的应用程序的数据快照作为只读逻辑卷(LV)创建，位于与原始数据相同的设备上。在复制为持久性卷(PV)并用作备份副本前，您必须手动挂载本地卷。要使用 MicroShift 存储卷的快照作为备份，请在本地主机中找到它，然后将其移到安全位置。

要查找特定的快照并复制它们，请使用以下步骤。

先决条件

- 有对主机的 root 访问权限。
- 您有一个现有的卷快照。

流程

1. 运行以下命令，获取卷快照的名称：

```
$ oc get volumesnapshot -n <namespace> <snapshot_name> -o 'jsonpath=
{.status.volumeSnapshotContentName}'
```

2. 使用以下命令在存储后端上创建的卷的唯一身份，并插入上一步中获取的名称：

```
$ oc get volumesnapshotcontent snapcontent-<retrieved_volume_identity> -o 'jsonpath=
{.status.snapshotHandle}'
```

3. 运行以下命令，使用您在上一步中获得的卷的唯一身份显示快照：

```
$ sudo lvdisplay <retrieved_snapshot_handle>
```

输出示例

```
--- Logical volume ---
LV Path                /dev/rhel/732e45ff-f220-49ce-859e-87ccca26b14c
LV Name                732e45ff-f220-49ce-859e-87ccca26b14c
VG Name                rhel
LV UUID                6Ojwc0-YTfp-nKJ3-F9FO-PvMR-Ic7b-LzNGSx
LV Write Access        read only
LV Creation host, time rhel-92.lab.local, 2023-08-07 14:45:26 -0500
LV Pool name           thinpool
LV Thin origin name    a2d2dcdc-747e-4572-8c83-56cd873d3b07
LV Status              available
# open                 0
LV Size                1.00 GiB
Mapped size            1.04%
Current LE             256
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:11
```

4. 运行以下命令，创建一个用于挂载 LV 的目录：

```
$ sudo mkdir /mnt/snapshot
```

5. 运行以下命令，使用检索的快照句柄的设备名称挂载 LV：

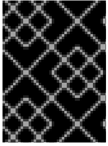
```
$ sudo mount /dev/<retrieved_snapshot_handle> /mnt/snapshot
```

6. 运行以下命令，从挂载的位置复制文件并将其存储在安全位置：

```
$ sudo cp -r /mnt/snapshot <destination>
```

7.3.3. 恢复卷快照

以下工作流演示了快照恢复。在本例中，还提供了验证步骤，以确保写入源持久性卷声明(PVC)的数据被保留并在新的 PVC 上恢复。



重要

快照必须恢复到与快照的源卷相同的 PVC。如果需要更大的 PVC，您可以在成功恢复快照后重新定义 PVC 的大小。

流程

1. 输入以下命令将 **VolumeSnapshot** 对象指定为持久性卷声明中的数据源来恢复快照：

```
$ oc apply -f <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-restore
spec:
  accessModes:
  - ReadWriteOnce
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: my-snap
  resources:
    requests:
      storage: 1Gi
  storageClassName: topolvm-provisioner-thin
---
apiVersion: v1
kind: Pod
metadata:
  name: base
spec:
  containers:
  - command:
      - nginx
     - -g
     - 'daemon off;'
    image: registry.redhat.io/rhel8/nginx-
122@sha256:908ebb0dec0d669caaf4145a8a21e04fdf9ebffbba5fd4562ce5ab388bf41ab2
    name: test-container
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
    volumeMounts:
    - mountPath: /vol
      name: test-vol
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  volumes:
  - name: test-vol
    persistentVolumeClaim:
      claimName: snapshot-restore
EOF
```

验证

1. 等待 pod 进入 **Ready** 状态：

```
$ oc wait --for=condition=Ready pod/base
```

2. 当新 pod 就绪时，验证来自应用程序的数据是否正确。

其他资源

- 恢复卷快照

7.3.4. 删除卷快照

您可以配置红帽如何构建 MicroShift 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>
```

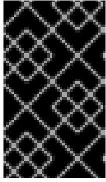
输出示例

```
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}}'
```

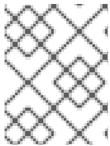
输出示例

```
volumesnapshotclass.snapshot.storage.k8s.io "csi-ocs-rbd-snapclass" deleted
```

删除终结器并删除卷快照。

7.4. 关于 LVM 卷克隆

逻辑卷管理器存储(LVMS)支持逻辑卷管理器(LVM)精简卷的持久性卷声明(PVC)克隆。克隆是现有卷的副本，可以像任何其他卷一样使用。置备时，如果数据源引用同一命名空间中的源 PVC，则创建原始卷的确切副本。创建克隆的 PVC 后，它被视为新对象，并完全与源 PVC 分开。克隆代表源在创建时的数据。



注意

只有在源和目标 PVC 位于同一命名空间中时，才能克隆。要创建 PVC 克隆，您必须在 RHEL for Edge 主机上配置精简卷。

其他资源

- CSI 卷克隆
- LVMS 卷克隆单节点 OpenShift
- 要将主机配置为启用克隆，请参阅关于 LVM 精简卷

第 8 章 使用 KUBE STORAGE VERSION MIGRATOR 进行存储迁移

存储版本迁移用于将集群中的现有对象从当前版本更新至最新版本。MicroShift 中使用 Kube Storage Version Migrator 嵌入式控制器来迁移资源，而无需重新创建这些资源。您或控制器都可以创建一个 **StorageVersionMigration** 自定义资源(CR)，它将通过 Migrator 控制器请求迁移。

8.1. 发出存储迁移请求

存储迁移是将存储的数据更新到最新的存储版本的过程，例如从 **v1beta1** 更新至 **v1beta2**。要更新您的存储版本，请使用以下步骤。

流程

- 您或任何支持 **StorageVersionMigration** API 的控制器都必须触发迁移请求。使用以下示例请求作为参考：

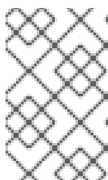
  请求示例

```
apiVersion: migration.k8s.io/v1alpha1
kind: StorageVersionMigration
metadata:
  name: snapshot-v1
spec:
  resource:
    group: snapshot.storage.k8s.io
    resource: volumesnapshotclasses (1)
    version: v1 (2)
```

  (1) 您必须使用资源的复数名称。

  (2) 版本已更新为：

- 迁移的进度被发布到 **StorageVersionMigration** 状态。



注意

- 由于未命名组或资源，所以可能会出现故障。
- 当之前和最新版本之间不兼容时，也可以发生迁移失败。