Red Hat Enterprise Linux 9

配置和管理逻辑卷

在 RHEL 上配置和管理 LVM

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 配置和管理逻辑卷

在 RHEL 上配置和管理 LVM

法律通告

摘要

逻辑卷管理(LVM)在物理存储上创建抽象层来创建逻辑存储卷，它是文件系统、数据库或应用程序可以使用的虚拟块存储设备。物理卷(PV)是分区或整个磁盘。通过使用这些 PV，您可以创建一个卷组(VG)，来从可用的存储为逻辑卷(LV)创建一个磁盘空间池。 您可以通过将物理卷合并到卷组中来创建逻辑卷(LV)。LV 比使用物理存储提供更大的灵活性，而创建的 LV 可以扩展或缩减，而不用重新分区或重新格式化物理设备。 您还可以使用 LVM 执行几个高级操作，如创建精简配置的逻辑卷、原始卷的快照、RAID 卷、缓存卷和条状逻辑卷。

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 点顶部导航栏中的 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您的改进建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 逻辑卷管理概述

逻辑卷管理(LVM)在物理存储上创建抽象层，帮助您创建逻辑存储卷。这比直接使用物理存储的方式具有更大的灵活性。

此外，硬件存储配置在软件中是隐藏的，因此可以在不停止应用程序或卸载文件系统的情况下调整大小和移动。这可降低操作成本。

1.1. LVM 构架

以下是 LVM 组件：

物理卷

物理卷(PV)是指定为 LVM 使用的分区或整个磁盘。如需更多信息，请参阅管理 LVM 物理卷。

卷组

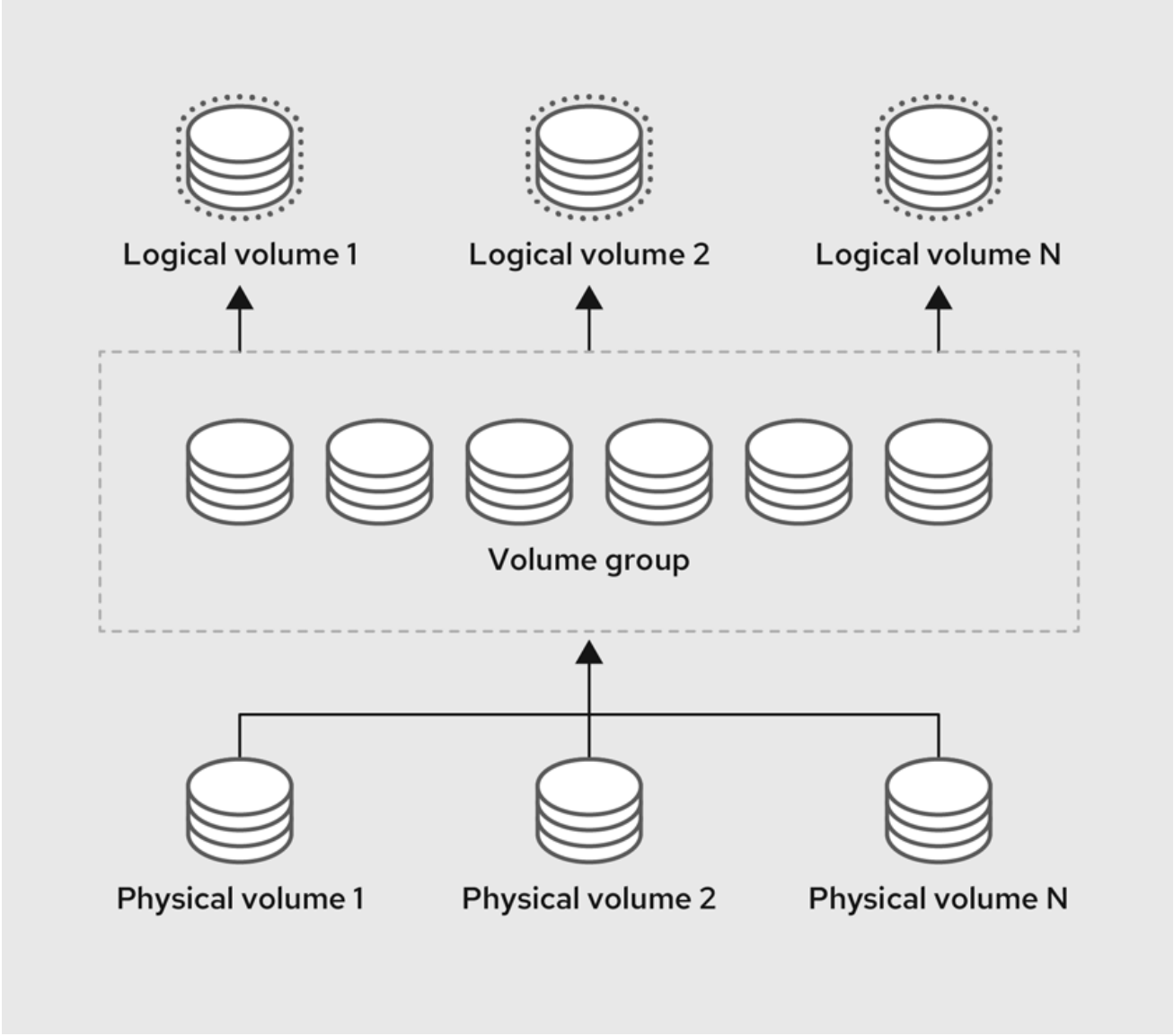
卷组(VG)是物理卷(PV)的集合，它会创建一个磁盘空间池，从中可以分配逻辑卷。如需更多信息，请参阅管理 LVM 卷组。

逻辑卷

逻辑卷代表可挂载的存储设备。如需更多信息，请参阅管理 LVM 逻辑卷。

下图显示了 LVM 的组件：

图 1.1. LVM 逻辑卷组件



1.2. LVM 的优点

与直接使用物理存储相比，逻辑卷具有以下优势：

灵活的容量

使用逻辑卷时，您可以将设备和分区聚合到一个逻辑卷中。借助此功能，文件系统可以扩展到多个设备中，就像它们是一个单一的大型设备一样。

方便设备命名

逻辑卷可以使用用户定义的名称和自定义名称进行管理。

存储卷大小

您可以使用简单的软件命令扩展逻辑卷或减小逻辑卷大小，而无需重新格式化和重新分区基础设备。如需更多信息，请参阅 修改逻辑卷的大小。

在线数据重新定位

要部署更新、更快或更弹性的存储子系统，您可以使用 **pvmove** 命令在系统处于活动状态时移动数据。在磁盘处于使用状态时可以重新分配磁盘。例如，您可以在删除热插拔磁盘前将其清空。
有关如何迁移数据的更多信息，请参阅 **pvmove** 手册页，以及 从卷组中删除物理卷。

条带化卷

您可以创建一个在两个或者多个设备间条带化分布数据的逻辑卷。这可显著提高吞吐量。如需更多信息，请参阅 扩展条状逻辑卷。

RAID 卷

逻辑卷为您对数据配置 RAID 提供了一种便捷的方式。这可防止设备故障并提高性能。如需更多信息，请参阅 配置 RAID 逻辑卷。

卷快照

您可以对数据进行快照（逻辑卷在一个特点时间点上的副本）用于一致性备份或测试更改的影响，而不影响实际数据。如需更多信息，请参阅 逻辑卷的快照。

精简卷

逻辑卷可以使用精简模式置备。这可让您创建大于可用物理空间的逻辑卷。如需更多信息，请参阅 创建和管理精简置备卷（精简卷） 。

缓存卷

缓存逻辑卷使用快速块设备，如 SSD 驱动器，以提高更大、较慢的块设备的性能。如需更多信息，请参阅 启用缓存以提高逻辑卷性能。

其他资源

- 自定义 LVM 报告

第 2 章 使用 RHEL 系统角色管理本地存储

要使用 Ansible 管理 LVM 和本地文件系统(FS)，您可以使用 **storage** 角色，这是 RHEL 9 中提供的 RHEL 系统角色之一。

使用 **存储** 角色可让您自动管理多台机器上的磁盘和逻辑卷上的文件系统，以及从 RHEL 7.7 开始的所有 RHEL 版本。

有关 RHEL 系统角色以及如何应用它们的更多信息，请参阅 RHEL 系统角色简介。

2.1. STORAGE RHEL 系统角色简介

存储 角色可以管理：

- 磁盘上未被分区的文件系统
- 完整的 LVM 卷组，包括其逻辑卷和文件系统
- MD RAID 卷及其文件系统

使用 **storage** 角色，您可以执行以下任务：

- 创建文件系统
- 删除文件系统
- 挂载文件系统
- 卸载文件系统
- 创建 LVM 卷组
- 删除 LVM 卷组
- 创建逻辑卷
- 删除逻辑卷
- 创建 RAID 卷
- 删除 RAID 卷
- 使用 RAID 创建 LVM 卷组
- 使用 RAID 删除 LVM 卷组
- 创建加密的 LVM 卷组
- 使用 RAID 创建 LVM 逻辑卷

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.2. 使用 STORAGE RHEL 系统角色在块设备上创建一个 XFS 文件系统

示例 Ansible playbook 应用 **storage** 角色，来使用默认参数在块设备上创建 XFS 文件系统。

注意

存储 角色只能在未分区、整个磁盘或逻辑卷(LV)上创建文件系统。它不能在分区中创建文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
```

   - 卷名称（示例中为 ***barefs***）目前是任意的。**存储** 角色根据 **disks:** 属性下列出的磁盘设备来识别卷。
   - 您可以省略 **fs_type: xfs** 行，因为 XFS 是 RHEL 9 中的默认文件系统。
   - 要在 LV 上创建文件系统，请在 **disks:** 属性下提供 LVM 设置，包括括起的卷组。详情请参阅 使用 storage RHEL 系统角色管理逻辑卷 。
     不要提供到 LV 设备的路径。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.3. 使用 STORAGE RHEL 系统角色永久挂载一个文件系统

示例 Ansible 应用 **storage** 角色，来立即且永久挂载 XFS 文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - 此 playbook 将文件系统添加到 **/etc/fstab** 文件中，并立即挂载文件系统。
   - 如果 **/dev/sdb** 设备上的文件系统或挂载点目录不存在，则 playbook 会创建它们。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件

- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.4. 使用 STORAGE RHEL 系统角色管理逻辑卷

示例 Ansible playbook 应用 **storage** 角色来在卷组中创建 LVM 逻辑卷。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt/dat
```

   - **myvg** 卷组由以下磁盘组成：**/dev/sda**、**/dev/sdb** 和 **/dev/sdc**。
   - 如果 **myvg** 卷组已存在，则 playbook 会将逻辑卷添加到卷组。
   - 如果 **myvg** 卷组不存在，则 playbook 会创建它。
   - playbook 在 **mylv** 逻辑卷上创建 Ext4 文件系统，并在 **/mnt** 上永久挂载文件系统。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.5. 使用 STORAGE RHEL 系统角色启用在线块丢弃

示例 Ansible playbook 应用 **storage** 角色，来挂载启用了在线块丢弃的 XFS 文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_options: discard
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.6. 使用 STORAGE RHEL 系统角色创建并挂载一个 EXT4 文件系统

示例 Ansible playbook 应用 **storage** 角色，来创建和挂载 Ext4 文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
```

   - playbook 在 **/dev/sdb** 磁盘上创建文件系统。
   - playbook 将文件系统永久挂载到 **/mnt/data** 目录。
   - 文件系统的标签是 **label-name**。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.7. 使用 STORAGE RHEL 系统角色创建并挂载一个 EXT3 文件系统

示例 Ansible playbook 应用 **storage** 角色，来创建和挂载 Ext3 文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- hosts: all
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - playbook 在 **/dev/sdb** 磁盘上创建文件系统。
   - playbook 将文件系统永久挂载到 **/mnt/data** 目录。
   - 文件系统的标签是 **label-name**。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.8. 使用 STORAGE RHEL 系统角色在 LVM 上重新调整现有文件系统的大小

示例 Ansible playbook 应用 **storage** RHEL 系统角色，以调整带有文件系统的 LVM 逻辑卷的大小。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Create LVM pool over three disks
  hosts: managed-node-01.example.com
  tasks:
    - name: Resize LVM logical volume with file system
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_pools:
          - name: myvg
            disks:
              - /dev/sda
              - /dev/sdb
              - /dev/sdc
            volumes:
              - name: mylv1
                size: 10 GiB
                fs_type: ext4
                mount_point: /opt/mount1
              - name: mylv2
                size: 50 GiB
                fs_type: ext4
                mount_point: /opt/mount2
```

   此 playbook 调整以下现有文件系统的大小：

   - 挂载在 **/opt/mount1** 上的 **mylv1** 卷上的 Ext4 文件系统，大小调整为 10 GiB。
   - 挂载在 **/opt/mount2** 上的 **mylv2** 卷上的 Ext4 文件系统，大小调整为 50 GiB。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.9. 使用 STORAGE RHEL 系统角色创建一个交换卷

本节提供了一个 Ansible playbook 示例。此 playbook 应用 **storage** 角色来创建一个交换卷（如果它不存在），或者使用默认参数在块设备上修改交换卷（如果它已存在）。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- name: Create a disk device with swap
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: swap_fs
        type: disk
        disks:
          - /dev/sdb
        size: 15 GiB
        fs_type: swap
```

卷名称（示例中的 ***swap_fs***）目前是任意的。**存储** 角色根据 **disks:** 属性下列出的磁盘设备来识别卷。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.10. 使用 STORAGE RHEL 系统角色配置一个 RAID 卷

使用 **storage** 系统角色，您可以使用 Red Hat Ansible Automation Platform 和 Ansible-Core 在 RHEL 上配置一个 RAID 卷。使用参数创建一个 Ansible playbook，以配置 RAID 卷以满足您的要求。

警告

设备名称在某些情况下可能会改变，例如：当您在系统中添加新磁盘时。因此，为了避免数据丢失，请不要在 playbook 中使用特定的磁盘名称。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure the storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_safe_mode: false
        storage_volumes:
          - name: data
            type: raid
            disks: [sdd, sde, sdf, sdg]
            raid_level: raid0
            raid_chunk_size: 32 KiB
            mount_point: /mnt/data
            state: present
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.11. 使用 STORAGE RHEL 系统角色配置带有 RAID 的 LVM 池

使用 **storage** 系统角色，您可以使用 Red Hat Ansible Automation Platform 在 RHEL 上配置一个带有 RAID 的 LVM 池。您可以使用可用参数建立一个 Ansible playbook，来配置带有 RAID 的 LVM 池。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure LVM pool with RAID
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
```

要使用 RAID 创建一个 LVM 池，您必须使用 **raid_level** 参数指定 RAID 类型。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录
- 管理 RAID

2.12. 使用 STORAGE RHEL 系统角色为 RAID LVM 卷配置条带大小

使用 **storage** 系统角色，您可以使用 Red Hat Ansible Automation Platform 在 RHEL 上为 RAID LVM 卷配置条带大小。您可以使用可用参数建立一个 Ansible playbook，来配置带有 RAID 的 LVM 池。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            raid_level: raid1
            raid_stripe_size: "256 KiB"
            state: present
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录
- 管理 RAID

2.13. 使用 STORAGE RHEL 系统角色在 LVM 上压缩和去重一个 VDO 卷

示例 Ansible playbook 应用 **storage** RHEL 系统角色，以便使用虚拟数据优化器(VDO)启用逻辑卷(LVM)的压缩和去重。

注意

由于 **storage** 系统角色使用 LVM VDO，因此每个池只有一个卷可以使用压缩和去重。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: mylv1
            compression: true
            deduplication: true
            vdo_pool_size: 10 GiB
            size: 30 GiB
            mount_point: /mnt/app/shared
```

在本例中，**compression** 和 **deduplication** 池被设为 true，这指定使用 VDO。下面描述了这些参数的用法：

- **deduplication** 用于去除存储在存储卷上的重复数据。
- compression 用于压缩存储在存储卷上的数据，从而提高存储量。
- vdo_pool_size 指定卷在设备上占用的实际大小。VDO 卷的虚拟大小由 **size** 参数设置。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

2.14. 使用 STORAGE RHEL 系统角色创建一个 LUKS2 加密的卷

您可以通过运行 Ansible playbook，使用 **storage** 角色来创建和配置使用 LUKS 加密的卷。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
---
- name: Create and configure a volume encrypted with LUKS
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: <password>
```

   您还可以将其他加密参数（如 **encryption_key**,**encryption_cipher**,**encryption_key_size** 和 **encryption_luks**）添加到 playbook 文件中。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

验证

1. 查看加密状态：

```
# cryptsetup status sdb

/dev/mapper/sdb is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/sdb
...
```

2. 验证创建的 LUKS 加密的卷：

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards

Data segments:
  0: crypt
        offset: 33554432 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 4096 [bytes]
...
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录
- 使用 LUKS 加密块设备

2.15. 使用 STORAGE RHEL 系统角色将池卷大小表示为百分比

示例 Ansible playbook 应用 **storage** 系统角色，以便您可以将逻辑卷管理器卷(LVM)卷大小表达为池总大小的百分比。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: data
            size: 60%
            mount_point: /opt/mount/data
          - name: web
            size: 30%
            mount_point: /opt/mount/web
          - name: cache
            size: 10%
            mount_point: /opt/cache/mount
```

这个示例将 LVM 卷的大小指定为池大小的百分比，例如：**60%**。另外，您还可以将 LVM 卷的大小指定为人类可读的文件系统的池大小的百分比，例如 **10g** 或 **50 GiB** 。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

第 3 章 管理 LVM 物理卷

物理卷(PV)是 LVM 要使用的分区或整个磁盘。要将设备用于 LVM 逻辑卷，必须将设备初始化为物理卷。

如果您将整个磁盘作为您的物理卷使用，那么磁盘就不能有分区表。对于 DOS 磁盘分区，应该使用 **fdisk** 或 **cfdisk** 命令或对等命令将分区 id 设置为 0x8e。如果您将整个磁盘作为您的物理卷使用，那么磁盘就不能有分区表。任何现有分区表都必须被清除，这样将有效销毁该磁盘上的所有数据。您可以使用 **wipefs -a <PhysicalVolume>`** 命令删除现有分区表，作为 root 用户。

3.1. 物理卷概述

将块设备初始化为物理卷会在接近设备起始的位置放置一个标签。下面描述了 LVM 标签：

- LVM 标签为物理设备提供正确的标识和设备排序。未标记的非 LVM 设备可以在重新引导后更改名称，具体取决于系统在启动过程中发现它们的顺序。LVM 标签在重新引导时具有持久性并在整个集群中可用。
- LVM 标签可将该设备识别为 LVM 物理卷。它包含一个随机唯一标识符，即物理卷的 UUID。它还以字节为单位保存块设备的大小，并记录 LVM 元数据存储在该设备中的位置。
- 默认情况下，LVM 标签是放在第二个 512 字节扇区。您可以在创建物理卷时将标签放在前 4 个扇区的任意一个扇区，从而覆盖此默认设置。如果需要，LVM 卷可与其它使用这些扇区的用户共同存在。

下面描述了 LVM 元数据：

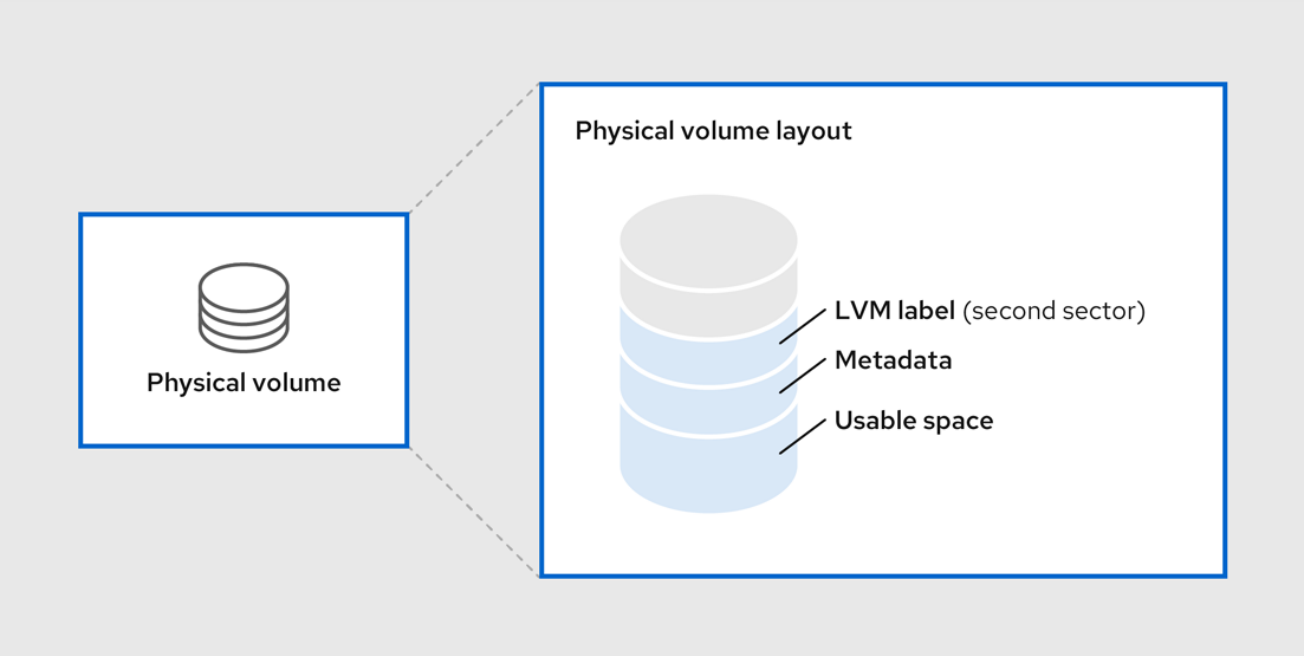
- LVM 元数据包含您系统中 LVM 卷组的配置详情。默认情况下，卷组中的每个物理卷的元数据区域都会保留一个一样的元数据副本。LVM 元数据很小，它以 ASCII 格式保存。
- 目前 LVM 允许您在每个物理卷中存储 0、1 或 2 个元数据副本。默认为 1 个副本。当您在物理卷中配置元数据副本数后，您将无法再更改该号码。第一个副本保存在设备的起始位置，紧随在标签后面。如果有第二个副本，会将其放在设备的末尾。如果您不小心写入了不同于您想要写入的磁盘覆盖了磁盘起始部分，那么您可以使用在设备末尾的元数据的第二个副本恢复元数据。

下图说明了 LVM 物理卷的布局。LVM 标签在第二个扇区，接下来是元数据区域，后面是设备的可用空间。

注意

在 Linux 内核和整个文档中，扇区的大小被视为 512 字节。

图 3.1. 物理卷布局



其他资源

- 一个磁盘上的多个分区

3.2. 一个磁盘上的多个分区

您可以使用 LVM 从磁盘分区中创建物理卷(PV)。

红帽建议您创建一个覆盖整个磁盘的单一分区，将其标记为 LVM 物理卷，理由如下：

方便管理

如果每个真实磁盘只出现一次，那么在系统中追踪硬件就比较容易。特别是当磁盘失败时。

条带化性能

LVM 无法告知两个物理卷位于同一个物理磁盘中。如果您在两个物理卷位于同一物理磁盘时创建了条状逻辑卷，那么条带就可能在同一磁盘的不同分区中。这可能会降低性能，而不是提高性能。

RAID 冗余

LVM 无法确定两个物理卷是否位于同一设备中。如果您在位于同一设备上的两个物理卷上创建 RAID 逻辑卷，则性能和容错可能会丢失。

虽然不建议您这样做，但在某些情况下可能需要将磁盘分成独立的 LVM 物理卷。例如：在有多个磁盘的系统中，当您要将现有系统迁移到 LVM 卷时，可能需要将数据在分区间转移。另外，如果您有一个很大的磁盘，并且因为管理的原因想要有一个以上卷组，那么对磁盘进行分区是很必要的。如果您的磁盘有一个以上的分区，且这些分区在同一卷组中，在创建卷时指定逻辑卷中应包含哪些分区。

请注意，虽然 LVM 支持将非分区磁盘用作物理卷，但建议创建一个全磁盘分区，因为在混合操作系统环境中创建不含分区的 PV 可能会有问题。其他操作系统可能会认为这些设备为空，并覆盖驱动器开头的 PV 标签。

3.3. 创建 LVM 物理卷

这个步骤描述了如何创建和标记 LVM 物理卷（PV）。

在此过程中，将 */dev/vdb1*、*/dev/vdb2* 和 */dev/vdb3* 替换为您系统中的可用存储设备。

先决条件

- 已安装 **lvm2** 软件包。

流程

1. 使用以空格分隔的设备名称作为 **pvcreate** 命令的参数来创建多个物理卷：

```
# pvcreate /dev/vdb1 /dev/vdb2 /dev/vdb3
  Physical volume "/dev/vdb1" successfully created.
  Physical volume "/dev/vdb2" successfully created.
  Physical volume "/dev/vdb3" successfully created.
```

   这会在 */dev/vdb1*、*/dev/vdb2* 和 */dev/vdb3* 上放置一个标签，将它们标记为属于 LVM 的物理卷。

2. 根据您的要求，使用以下命令之一查看创建的物理卷：

   a. **pvdisplay** 命令，它为每个物理卷提供详细的多行输出。它以固定格式显示物理属性，如大小、扩展、卷组和其他选项：

```
# pvdisplay
--- NEW Physical volume ---
  PV Name               /dev/vdb1
  VG Name
  PV Size               1.00 GiB
[..]
--- NEW Physical volume ---
  PV Name               /dev/vdb2
  VG Name
  PV Size               1.00 GiB
[..]
--- NEW Physical volume ---
  PV Name               /dev/vdb3
  VG Name
  PV Size               1.00 GiB
[..]
```

   b. **pvs** 命令提供了可以对其进行格式配置的物理卷信息，每行显示一个物理卷。

```
# pvs
  PV         VG  Fmt    Attr    PSize      PFree
/dev/vdb1        lvm2           1020.00m   0
/dev/vdb2        lvm2           1020.00m   0
/dev/vdb3        lvm2           1020.00m   0
```

   c. **pvscan** 命令扫描系统中所有支持的物理卷 LVM 块设备。您可以在 **lvm.conf** 文件中定义过滤器，以便这个命令避免扫描特定物理卷：

```
# pvscan
  PV /dev/vdb1                    lvm2 [1.00 GiB]
  PV /dev/vdb2                    lvm2 [1.00 GiB]
  PV /dev/vdb3                    lvm2 [1.00 GiB]
```

-

其他资源

- **pvcreate(8)**、**pvdisplay(8)**、**pvs(8)**、**pvscan(8)** 和 **lvm(8)** 手册页

3.4. 删除 LVM 物理卷

如果 LVM 不再需要某个设备，您可以使用 **pvremove** 命令删除 LVM 标签。执行 **pvremove** 命令会将空物理卷上的 LVM 元数据归零。

流程

1. 删除物理卷：

```
# pvremove /dev/vdb3
Labels on physical volume "/dev/vdb3" successfully wiped.
```

2. 查看现有物理卷并验证是否已删除所需卷：

```
# pvs
  PV         VG   Fmt  Attr PSize    PFree
/dev/vdb1        lvm2       1020.00m  0
/dev/vdb2        lvm2       1020.00m  0
```

如果您要删除的物理卷当前是卷组的一部分，则必须使用 **vgreduce** 命令将其从卷组中删除。如需更多信息，请参阅从卷组中删除物理卷

其他资源

- **pvremove(8)** 手册页

3.5. 其他资源

- 使用 parted 在磁盘中创建分区表.
- **parted(8)** 手册页。

第 4 章 管理 LVM 卷组

卷组(VG)是物理卷(PV)的集合，它会创建一个磁盘空间池，从中可以分配逻辑卷。

在卷组中，可用于分配的磁盘空间被分成固定大小的单元，我们称之为扩展。一个扩展就是可被分配的最小空间单位。在物理卷中，扩展被称为物理扩展。

逻辑卷被分配成与物理卷扩展大小相同的逻辑扩展。因此，扩展大小对卷组中的所有逻辑卷都一样。卷组将逻辑扩展与物理扩展匹配。

4.1. 创建 LVM 卷组

您可以使用 */dev/vdb1* 和 */dev/vdb2* 物理卷(PV)创建 LVM 卷组(VG) myvg。默认情况下，当使用物理卷创建卷组时，其磁盘空间被分成 4MB 扩展。这个扩展大小是最小数量，可以通过它来增加或减小逻辑卷的大小。可以使用 **vgcreate** 命令的 **-s** 参数修改扩展大小，大数量的扩展对逻辑卷的 I/O 性能没有影响。您可以使用 **vgcreate** 命令的 **-p** 和 **-l** 参数限制卷组可以拥有的物理或者逻辑卷的数量。

先决条件

- 已安装 **lvm2** 软件包。
- 创建一个或多个物理卷。有关创建物理卷的更多信息，请参阅 创建 LVM 物理卷。

流程

1. 使用以下任一方法创建 *myvg* VG：
   - 不指定任何选项：

     ```
     # vgcreate myvg /dev/vdb1 /dev/vdb2
      Volume group "myvg" successfully created.
     ```

   - 使用 **-s** 参数指定卷组扩展大小：

     ```
     # vgcreate -s 2 /dev/myvg /dev/vdb1 /dev/vdb2
     Volume group "myvg" successfully created.
     ```

   - 通过使用 **-p** 和 **-l** 参数限制 VG 可以拥有的物理或者逻辑卷的数量：

     ```
     # vgcreate -l 1 /dev/myvg /dev/vdb1 /dev/vdb2
     Volume group "myvg" successfully created.
     ```

2. 根据您的要求，使用以下命令之一查看创建的卷组：
   - **vgs** 命令以可配置的形式提供卷组信息，每个卷组显示一行：

     ```
     # vgs
       VG    #PV #LV #SN  Attr  VSize   VFree
      myvg   2    0   0   wz-n  159.99g 159.99g
     ```

   - **vgdisplay** 命令以固定格式显示卷组属性，如大小、区块、物理卷数量和其他选项。以下示例显示了卷组 *myvg* 的 **vgdisplay** 命令的输出。要显示所有现有卷组，请不要指定卷组：

     ```
     # vgdisplay myvg
     ```

```
  --- Volume group ---
  VG Name               myvg
  System ID
  Format                lvm2
  Metadata Areas        4
  Metadata Sequence No  6
  VG Access             read/write
[..]
```

- **vgscan** 命令为卷组扫描系统中所有受支持的 LVM 块设备：

```
# vgscan
  Found volume group "myvg" using metadata type lvm2
```

3. 可选：通过添加一个或多个可用物理卷来提高卷组容量：

```
# vgextend myvg /dev/vdb3
Physical volume "/dev/vdb3" successfully created.
Volume group "myvg" successfully extended
```

4. 可选：重命名现有卷组：

```
# vgrename myvg myvg1
Volume group "myvg" successfully renamed to "myvg1"
```

其他资源

- **vgcreate (8)**, **vgextend (8)**, **vgdisplay (8)**, **vgs (8)**, **vgscan (8)**, **vgrename (8)** 和 **lvm (8)** 手册页

4.2. 合并 LVM 卷组

要将两个卷组合并为一个卷组，请使用 **vgmerge** 命令。如果这两个卷的物理扩展大小相等，且两个卷组的物理卷和逻辑卷的描述符合目的卷组的限制，您可以将一个不活跃的"源"卷与一个活跃或者不活跃的"目标"卷合并。

流程

- 将不活跃卷组 *数据库* 合并到活跃或者不活跃的卷组 *myvg* 中，提供详细的运行时信息：

```
# vgmerge -v myvg databases
```

其他资源

- **vgmerge(8)** 手册页

4.3. 从卷组中删除物理卷

要从卷组(VG)中删除未使用的物理卷(PV)，请使用 **vgreduce** 命令。**vgreduce** 命令通过删除一个或多个空物理卷来缩小卷组的容量。这样就可以使不同的卷组自由使用那些物理卷，或者将其从系统中删除。

流程

1. 如果物理卷仍在使用，则将数据从同一卷组中迁移到另一个物理卷中：

```
# pvmove /dev/vdb3
  /dev/vdb3: Moved: 2.0%
 ...
  /dev/vdb3: Moved: 79.2%
 ...
  /dev/vdb3: Moved: 100.0%
```

2. 如果现有卷组中的其他物理卷上没有足够的空闲扩展：

   a. 从 */dev/vdb4* 创建一个新物理卷：

```
# pvcreate /dev/vdb4
  Physical volume "/dev/vdb4" successfully created
```

   b. 将新创建的物理卷添加到 *myvg* 卷组：

```
# vgextend myvg /dev/vdb4
  Volume group "myvg" successfully extended
```

   c. 将数据从 */dev/vdb3* 移到 */dev/vdb4* 中 ：

```
# pvmove /dev/vdb3 /dev/vdb4
  /dev/vdb3: Moved: 33.33%
  /dev/vdb3: Moved: 100.00%
```

3. 从卷组中删除物理卷 */dev/vdb3*:

```
# vgreduce myvg /dev/vdb3
Removed "/dev/vdb3" from volume group "myvg"
```

验证

- 验证 */dev/vdb3* 物理卷是否已从 *myvg* 卷组中删除：

```
# pvs
  PV         VG    Fmt   Attr   PSize       PFree      Used
  /dev/vdb1 myvg  lvm2   a--    1020.00m    0          1020.00m
  /dev/vdb2 myvg  lvm2   a--    1020.00m    0          1020.00m
  /dev/vdb3       lvm2   a--    1020.00m   1008.00m    12.00m
```

其他资源

- **vgreduce(8)**, **pvmove(8)**, 和 **pvs(8)** man pages

4.4. 分割 LVM 卷组

如果在物理卷中有足够的空闲空间，就可在不添加新磁盘的情况下创建新的卷组。

在初始设置中，卷组 *myvg* 由 */dev/vdb1*、*/dev/vdb2* 和 */dev/vdb3* 组成。完成此步骤后，卷组 *myvg* 将包含 */dev/vdb1* 和 */dev/vdb2*，第二个卷组 *yourvg* 将包含 */dev/vdb3*。

先决条件

- 卷组中有足够的空间。使用 **vgscan** 命令确定卷组中当前有多少可用空间。
- 根据现有物理卷中的可用容量，使用 **pvmove** 命令将所有使用的物理区块移动到其他物理卷。如需更多信息，请参阅从卷组中删除物理卷。

流程

1. 将现有卷组 *myvg* 拆分到新卷组 *yourvg* ：

```
# vgsplit myvg yourvg /dev/vdb3
  Volume group "yourvg" successfully split from "myvg"
```

注意

如果您使用现有卷组创建了逻辑卷，请使用以下命令取消激活逻辑卷：

```
# lvchange -a n /dev/myvg/mylv
```

有关创建逻辑卷的更多信息，请参阅 管理 LVM 逻辑卷。

2. 查看两个卷组的属性：

```
# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  myvg     2   1   0 wz--n- 34.30G 10.80G
  yourvg   1   0   0 wz--n- 17.15G 17.15G
```

验证

- 验证新创建的卷组 *yourvg* 是否由 */dev/vdb3* 物理卷组成：

```
# pvs
  PV         VG     Fmt  Attr PSize    PFree    Used
  /dev/vdb1  myvg   lvm2 a--  1020.00m 0        1020.00m
  /dev/vdb2  myvg   lvm2 a--  1020.00m 0        1020.00m
  /dev/vdb3  yourvg lvm2 a--  1020.00m 1008.00m   12.00m
```

其他资源

- **vgsplit(8)**、 **vgs(8)** 和 **pvs(8)** man page

4.5. 将卷组移动到另一个系统中

您可以使用以下命令将整个 LVM 卷组(VG)移到另一个系统中：

vgexport

在现有系统上使用这个命令使系统无法访问不活跃的 VG。一旦 VG 无法访问，您就可以分离其物理卷(PV)。

vgimport

在其他系统上使用此命令使在旧系统中不活跃的 VG 可在新系统中访问。

先决条件

- 没有用户正在访问您要移动的卷组中活动的卷上的文件。

流程

1. 卸载 *mylv* 逻辑卷：

```
# umount /dev/mnt/mylv
```

2. 取消激活卷组中的所有逻辑卷，这可防止卷组上任何进一步的活动：

```
# vgchange -an myvg
vgchange -- volume group "myvg" successfully deactivated
```

3. 导出卷组，以防止其被您要从中删除它的系统访问。

```
# vgexport myvg
vgexport -- volume group "myvg" successfully exported
```

4. 查看导出的卷组：

```
# pvscan
  PV /dev/sda1    is in exported VG myvg [17.15 GB / 7.15 GB free]
  PV /dev/sdc1    is in exported VG myvg [17.15 GB / 15.15 GB free]
  PV /dev/sdd1   is in exported VG myvg [17.15 GB / 15.15 GB free]
  ...
```

5. 关闭您的系统,拔出组成卷组的磁盘,并将其连接到新系统。

6. 将磁盘插入新系统，并导入卷组使其可以被新系统访问：

```
# vgimport myvg
```

注意

您可以使用 **vgimport** 命令的 **--force** 参数导入缺少物理卷的卷组，然后运行 **vgreduce --removemissing** 命令。

7. 激活卷组：

```
# vgchange -ay myvg
```

8. 挂载文件系统使其可用：

```
# mkdir -p /mnt/myvg/users
# mount /dev/myvg/users /mnt/myvg/users
```

其他资源

- **vgimport(8)**, **vgexport(8)**, 和 **vgchange(8)** 手册页

4.6. 删除 LVM 卷组

您可以使用 **vgremove** 命令删除现有卷组。

先决条件

- 卷组没有包含逻辑卷。要从卷组中删除逻辑卷，请参阅删除 LVM 逻辑卷。

流程

1. 如果卷组存在于集群的环境中，在所有节点上停止卷组的锁定空间。在除了您要删除的节点外的所有节点上使用以下命令：

```
# vgchange --lockstop vg-name
```

   等待锁定停止。

2. 删除卷组：

```
# vgremove vg-name
  Volume group "vg-name" successfully removed
```

其他资源

- **vgremove(8)** 手册页

第 5 章 管理 LVM 逻辑卷

逻辑卷是文件系统、数据库或应用可以使用的虚拟块存储设备。要创建 LVM 逻辑卷，物理卷(PV)合并为一个卷组(VG)。这会创建一个磁盘空间池，用于分配 LVM 逻辑卷（LV）。

5.1. 逻辑卷概述

管理员可以在不损坏数据的情况下增大或缩小逻辑卷，这与标准磁盘分区不同。如果卷组中的物理卷位于不同的驱动器或者 RAID 阵列中，那么管理员也可以跨存储设备分配逻辑卷。

如果您缩小逻辑卷到比卷中数据所需的容量小的容量时，则可能会丢失数据。此外，某些文件系统无法缩小。为确保最大的灵活性，创建逻辑卷以满足您当前的需求，并使超额存储容量保持未分配。您可以根据需要安全地扩展逻辑卷使用未分配空间。

重要

在 AMD、Intel、ARM 和 IBM Power Systems 服务器中,引导装载程序无法读取 LVM 卷。您必须为您的 **/boot** 分区创建一个标准的非 LVM 磁盘分区。在 IBM Z 中， **zipl** 引导装载程序通过线性映射在 LVM 逻辑卷中支持 **/boot**。默认情况下，安装过程总是在 LVM 卷中创建 / 和 swap 分区，物理卷中有一个单独的 **/boot** 分区。

以下是不同类型的逻辑卷：

线性卷

线性卷将来自一个或多个物理卷的空间集合到一个逻辑卷中。例如：如果您有两个 60GB 的磁盘，您可以创建一个 120GB 的逻辑卷。物理存储是连在一起的。

条带化逻辑卷

当您向 LVM 逻辑卷写入数据时，文件系统会在基本物理卷之间部署数据。您可以通过创建一个条状逻辑卷来控制将数据写入物理卷的方法。对于大量连续的读取和写入，这样可以提高数据输入/输出的效率。

条带化通过以 round-robin 模式向预定数目的物理卷写入数据来提高性能。使用条带，I/O 可以并行执行。在某些情况下，这可能会为条带中的每个额外物理卷增加近线性能。

RAID 逻辑卷

LVM 支持 RAID 0、1、4、5、6 和 10。RAID 逻辑卷不是集群感知的。当您创建 RAID 逻辑卷时，LVM 会创建一个元数据子卷，它是阵列中的每个数据或奇偶校验子卷的大小的一个区块。

精简配置的逻辑卷（精简卷）

使用精简配置的逻辑卷，您可以创建大于可用物理存储的逻辑卷。通过创建精简配置的卷集合，系统可以分配您使用的内容，而不是分配请求的完整存储量

快照卷

LVM 快照功能提供在特定时间创建设备的虚拟镜像且不会造成服务中断的功能。在提取快照后，当对原始设备进行修改时，快照功能会生成有变化的数据区域的副本，以便重建该设备的状态。

精简配置的快照卷

使用精简配置的快照卷，可以有更多虚拟设备存储在同一个数据卷中。精简置备的快照很有用，因为您不会复制在给定时间要捕获的所有数据。

缓存卷

LVM 支持在较慢的块设备中使用快速块设备（比如 SSD 驱动器）作为写入或者写入缓存。用户可以创建缓存逻辑卷来提高其现有逻辑卷的性能，或者创建由小而快速的设备组成的新缓存逻辑卷，再加上一个大型、较慢的设备。

5.2. 创建 LVM 逻辑卷

先决条件

- 已安装 **lvm2** 软件包。
- 已创建卷组。如需更多信息，请参阅创建 LVM 卷组。

流程

1. 创建逻辑卷：

```
# lvcreate -n mylv -L 500M myvg
Logical volume "mylv" successfully created.
```

使用 **-n** 选项将 LV 名称设置为 *mylv*，并使用 **-L** 选项以 Mb 为单位设置 LV 的大小，但可以使用任何其他单元。默认情况下 LV 类型是线性的，但用户可以使用 **--type** 选项指定所需的类型。

重要

如果 VG 没有足够数量的可用物理扩展用于请求的大小和类型，该命令将失败。

2. 根据您的要求，使用以下命令之一查看创建的逻辑卷：

   a. **lvs** 命令提供了可以对其进行格式配置的逻辑卷信息，每行显示一个逻辑卷。

```
# lvs
  LV   VG   Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg -wi-ao----   500.00m
```

   b. **lvdisplay** 命令以固定格式显示逻辑卷属性，如大小、布局和映射：

```
# lvdisplay -v /dev/myvg/mylv
  --- Logical volume ---
  LV Path                /dev/myvg/mylv
  LV Name                mylv
  VG Name                myvg
  LV UUID                YTnAk6-kMIT-c4pG-HBFZ-Bx7t-ePMk-7YjhaM
  LV Write Access        read/write
[..]
```

   c. **lvscan** 命令扫描系统中所有逻辑卷并列出它们：

```
# lvscan
  ACTIVE             '/dev/myvg/mylv' [500.00 MiB] inherit
```

3. 在逻辑卷中创建文件系统。以下命令在逻辑卷中创建 **xfs** 文件系统：

```
# mkfs.xfs /dev/myvg/mylv
meta-data=/dev/myvg/mylv      isize=512    agcount=4, agsize=32000 blks
         =                    sectsz=512   attr=2, projid32bit=1
         =                    crc=1        finobt=1, sparse=1, rmapbt=0
         =                    reflink=1
```

```
data     =                       bsize=4096   blocks=128000, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
Discarding blocks...Done.
```

4. 挂载逻辑卷并报告文件系统磁盘空间使用情况：

```
# mount /dev/myvg/mylv /mnt

# df -h
Filesystem             1K-blocks  Used  Available Use% Mounted on

/dev/mapper/myvg-mylv   506528    29388  477140    6%   /mnt
```

其他资源

- **lvcreate(8)**, **lvdisplay(8)**, **lvs(8)**, **lvscan(8)**, **lvm(8)** 和 **mkfs.xfs(8)** man page

5.3. 创建 RAID0 条带化逻辑卷

RAID0 逻辑卷以条的大小为单位，将逻辑卷数据分散到多个数据子卷中。下面的步骤创建了一个名为 *mylv* 的 LVM RAID0 逻辑卷，该逻辑卷在磁盘间条状分布数据。

先决条件

1. 您已创建了三个或者多个物理卷。有关创建物理卷的更多信息，请参阅 创建 LVM 物理卷。
2. 您已创建了卷组。如需更多信息，请参阅创建 LVM 卷组。

流程

1. 从现有卷组中创建 RAID0 逻辑卷。以下命令从卷组 *myvg* 中创建 RAID0 卷 *mylv*，大小为 *2G*，有三个条带，条带大小为 *4kB* ：

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv my_vg
  Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
  Logical volume "mylv" created.
```

2. 在 RAID0 逻辑卷中创建文件系统。以下命令在逻辑卷中创建 ext4 文件系统：

```
# mkfs.ext4 /dev/my_vg/mylv
```

3. 挂载逻辑卷并报告文件系统磁盘空间使用情况：

```
# mount /dev/my_vg/mylv /mnt

# df
Filesystem             1K-blocks     Used  Available  Use% Mounted on
/dev/mapper/my_vg-mylv   2002684     6168  1875072    1%   /mnt
```

验证

- 查看创建的 RAID0 剥离的逻辑卷：

```
# lvs -a -o +devices,segtype my_vg
  LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert Devices Type
  mylv my_vg rwi-a-r--- 2.00g mylv_rimage_0(0),mylv_rimage_1(0),mylv_rimage_2(0) raid0
  [mylv_rimage_0] my_vg iwi-aor--- 684.00m /dev/sdf1(0) linear
  [mylv_rimage_1] my_vg iwi-aor--- 684.00m /dev/sdg1(0) linear
  [mylv_rimage_2] my_vg iwi-aor--- 684.00m /dev/sdh1(0) linear
```

5.4. 重命名 LVM 逻辑卷

这个步骤描述了如何将现有逻辑卷 *mylv* 重命名为 *mylv1*。

流程

1. 如果逻辑卷当前已挂载，卸载该卷：

```
# umount /mnt
```

使用挂载点替换 */mnt*。

2. 重命名现有逻辑卷：

```
# lvrename myvg mylv mylv1
Renamed "mylv" to "mylv1" in volume group "myvg"
```

您还可以通过指定设备的完整路径来重命名逻辑卷：

```
# lvrename /dev/myvg/mylv /dev/myvg/mylv1
```

其他资源

- **lvrename(8)** 手册页

5.5. 从逻辑卷中删除磁盘

这个步骤描述了如何从现有逻辑卷中删除磁盘，替换磁盘或者将磁盘用作不同卷的一部分。

要删除磁盘，您必须首先将 LVM 物理卷中的扩展移动到不同的磁盘或者一组磁盘中。

流程

1. 在使用 LV 时查看物理卷的已用和可用空间：

```
# pvs -o+pv_used
  PV         VG    Fmt    Attr   PSize      PFree     Used
 /dev/vdb1 myvg  lvm2   a--    1020.00m    0        1020.00m
 /dev/vdb2 myvg  lvm2   a--    1020.00m    0        1020.00m
 /dev/vdb3 myvg  lvm2   a--    1020.00m   1008.00m   12.00m
```

2. 将数据移到其他物理卷中：

a. 如果现有卷组中的其他物理卷中有足够的可用扩展，请使用以下命令移动数据：

```
# pvmove /dev/vdb3
  /dev/vdb3: Moved: 2.0%
 ...
  /dev/vdb3: Moved: 79.2%
 ...
  /dev/vdb3: Moved: 100.0%
```

b. 如果现有卷组中的其他物理卷上没有足够的可用扩展，请使用以下命令来添加新物理卷，使用新创建的物理卷扩展卷组，并将数据移动到此物理卷中：

```
# pvcreate /dev/vdb4
  Physical volume "/dev/vdb4" successfully created

# vgextend myvg /dev/vdb4
  Volume group "myvg" successfully extended

# pvmove /dev/vdb3 /dev/vdb4
  /dev/vdb3: Moved: 33.33%
  /dev/vdb3: Moved: 100.00%
```

3. 删除物理卷：

```
# vgreduce myvg /dev/vdb3
Removed "/dev/vdb3" from volume group "myvg"
```

如果逻辑卷包含失败的物理卷，您就无法使用该逻辑卷。要从卷组中删除缺少的物理卷，如果缺少的物理卷上没有分配逻辑卷，您可以使用 **vgreduce** 命令的 **--removemissing** 参数：

```
# vgreduce --removemissing myvg
```

其他资源

- **pvmove(8)**, **vgextend(8)**, **vereduce(8)**, 和 **pvs(8)** man 页

5.6. 删除 LVM 逻辑卷

此流程描述了如何从卷组 *myvg* 中删除现有逻辑卷 */dev/myvg/mylv1*。

流程

1. 如果逻辑卷当前已挂载，卸载该卷：

```
# umount /mnt
```

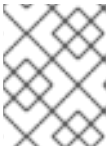
2. 如果在集群环境中存在逻辑卷，则在所有其激活的节点上取消激活逻辑卷。对每个这样的节点运行以下命令：

```
# lvchange --activate n vg-name/lv-name
```

3. 使用 **lvremove** 实用程序删除逻辑卷：

```
# lvremove /dev/myvg/mylv1

Do you really want to remove active logical volume "mylv1"? [y/n]: y
Logical volume "mylv1" successfully removed
```



注意

在这种情况下，逻辑卷还没有被取消激活。如果您在删除逻辑卷前明确取消激活了逻辑卷，则无法看到验证您是否要删除活跃逻辑卷的提示信息。

其他资源

- **lvremove(8)** 手册页

5.7. 使用 RHEL 系统角色管理 LVM 逻辑卷

使用 **storage** 角色执行以下任务：

- 在由多个磁盘组成的卷组中创建 LVM 逻辑卷。
- 在逻辑卷中创建一个带给定标签的 ext4 文件系统。
- 永久挂载 ext4 文件系统。

先决条件

- 包含 **storage** 角色的 Ansible playbook

5.7.1. 使用 storage RHEL 系统角色管理逻辑卷

示例 Ansible playbook 应用 **storage** 角色来在卷组中创建 LVM 逻辑卷。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
```

```
    volumes:
      - name: mylv
        size: 2G
        fs_type: ext4
        mount_point: /mnt/dat
```

- **myvg** 卷组由以下磁盘组成：**/dev/sda**、**/dev/sdb** 和 **/dev/sdc**。
- 如果 **myvg** 卷组已存在，则 playbook 会将逻辑卷添加到卷组。
- 如果 **myvg** 卷组不存在，则 playbook 会创建它。
- playbook 在 **mylv** 逻辑卷上创建 Ext4 文件系统，并在 **/mnt** 上永久挂载文件系统。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

5.7.2. 其他资源

- 有关 **storage** 角色的更多信息，请参阅 使用 RHEL 系统角色管理本地存储。

5.8. 删除 LVM 卷组

您可以使用 **vgremove** 命令删除现有卷组。

先决条件

- 卷组没有包含逻辑卷。要从卷组中删除逻辑卷，请参阅删除 LVM 逻辑卷。

流程

1. 如果卷组存在于集群的环境中，在所有节点上停止卷组的锁定空间。在除了您要删除的节点外的所有节点上使用以下命令：

```
# vgchange --lockstop vg-name
```

等待锁定停止。

2. 删除卷组：

```
# vgremove vg-name
  Volume group "vg-name" successfully removed
```

其他资源

- **vgremove(8)** 手册页

第 6 章 修改逻辑卷的大小

创建逻辑卷后，您可以修改卷的大小。

6.1. 扩展逻辑卷和文件系统

您可以使用 **lvextend** 命令扩展逻辑卷(LV)。您可以指定您要扩展多少 LV，或者您希望扩展后 LV 有大小。使用 **lvextend** 命令的 **-r** 选项，增长底层文件系统以及 LV。

警告

您还可以使用 **lvresize** 命令扩展逻辑卷，但这个命令不能保证不会意外缩小。

先决条件

- 您有一个现有逻辑卷(LV)，其中包含一个文件系统。使用 **df -Th** 命令确定文件系统类型和大小。有关创建逻辑卷和文件系统的更多信息，请参阅创建 LVM 逻辑卷。
- 卷组中有足够的空间来扩展 LV 和文件系统。使用 **vgs -o name,vgfree** 命令确定可用空间。有关创建卷组的更多信息，请参阅 创建 LVM 卷组。

流程

1. 可选：如果卷组没有足够的空间增大 LV，请向卷组中添加新物理卷：

```
# vgextend myvg /dev/vdb3
Physical volume "/dev/vdb3" successfully created.
Volume group "myvg" successfully extended.
```

2. 扩展 LV 和文件系统：

注意

使用没有 **-r** 参数的 **lvextend** 命令仅能扩展 LV。要扩展底层 XFS 文件系统，请参阅 增加 XFS 文件系统的大小，对于 GFS2 文件系统，请参阅 增大 GFS2 文件系统，对于 ext4 文件系统，请参阅 调整 ext4 文件系统大小。

注意

使用 **-L** 选项将 LV 扩展到新大小，使用 **-l** 选项指定扩展的数目，具体取决于您要增大的逻辑卷的大小。

```
# lvextend -r -L 3G /dev/myvg/mylv
fsck from util-linux 2.32.1
/dev/mapper/myvg-mylv: clean, 11/131072 files, 26156/524288 blocks
  Size of logical volume myvg/mylv changed from 2.00 GiB (512 extents) to 3.00 GiB (768 extents).
  Logical volume myvg/mylv successfully resized.
```

```
resize2fs 1.45.6 (20-Mar-2020)
Resizing the filesystem on /dev/mapper/myvg-mylv to 786432 (4k) blocks.
The filesystem on /dev/mapper/myvg-mylv is now 786432 (4k) blocks long.
```

您还可以扩展 *mylv* 逻辑卷，来填充 *myvg* 卷组中所有未分配的空间：

```
# lvextend -l +100%FREE /dev/myvg/mylv
 Size of logical volume myvg/mylv changed from 10.00 GiB (2560 extents) to 6.35 TiB (1665465 extents).
 Logical volume myvg/mylv successfully resized.
```

验证

- 验证文件系统和 LV 是否已增长：

```
# df -Th
Filesystem           Type      Size  Used Avail Use% Mounted on
devtmpfs             devtmpfs  1.9G     0  1.9G   0% /dev
tmpfs                tmpfs     1.9G     0  1.9G   0% /dev/shm
tmpfs                tmpfs     1.9G  8.6M  1.9G   1% /run
tmpfs                tmpfs     1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/mapper/rhel-root xfs        45G  3.7G   42G   9% /
/dev/vda1            xfs      1014M  369M  646M  37% /boot
tmpfs                tmpfs     374M     0  374M   0% /run/user/0
/dev/mapper/myvg-mylv xfs       2.0G   47M  2.0G   3% /mnt/mnt1
```

其他资源

- **vgextend(8)**, **lvextend(8)**, 和 **xfs_growfs(8)** 手册页

6.2. 减少逻辑卷和文件系统

您可以使用 **lvreduce** 命令和 **resizefs** 选项来减少逻辑卷及其文件系统。

如果您要缩小的逻辑卷包含一个文件系统，为了防止数据丢失，必须确定该文件系统没有使用将被缩小的逻辑卷中的空间。因此，当逻辑卷包含文件系统时，请使用 **lvreduce** 命令的 **--resizefs** 选项。

当您使用 **--resizefs** 时，**lvreduce** 会在缩小逻辑卷前尝试缩小文件系统。如果缩小文件系统因为其满了或者不支持缩小而失败，则 **lvreduce** 命令会失败，且不会尝试减少逻辑卷。

警告

在大多数情况下，**lvreduce** 命令会警告可能的数据丢失，并要求进行确认。但是，您不应该依赖于这些确认提示来防止数据丢失，因为在某些情况下，您不会看到这些提示信息，比如当逻辑卷不活跃或者没有使用 **--resizefs** 选项时。

请注意，使用 **lvreduce** 命令的 **--test** 选项并不表示操作是否安全，因为此选项不会检查文件系统或测试文件系统大小调整。

先决条件

- 逻辑卷的文件系统支持缩小。使用 **df -Th** 命令确定文件系统类型和大小。

注意

例如：GFS2 和 XFS 文件系统不支持缩小。

- 底层文件系统不使用正在缩小的 LV 中的空间。

流程

1. 使用以下选项之一缩小 *myvg* 卷组中的 *mylv* 逻辑卷及其文件系统：
   - 将 LV 及其文件系统减少到所需值：

```
# lvreduce --resizefs -L 500M myvg/mylv
File system ext4 found on myvg/mylv.
File system size (2.00 GiB) is larger than the requested size (500.00 MiB).
File system reduce is required using resize2fs.
...
Logical volume myvg/mylv successfully resized.
```

   - 从逻辑卷和文件系统减少 64MB：

```
# lvreduce --resizefs -L -64M myvg/mylv
File system ext4 found on myvg/mylv.
File system size (500.00 MiB) is larger than the requested size (436.00 MiB).
File system reduce is required using resize2fs.
...
Logical volume myvg/mylv successfully resized
```

其他资源

- **lvreduce(8)** man 页

6.3. 扩展条状逻辑卷

您可以使用 **lvextend** 命令和所需的值来扩展条状逻辑卷(LV)。

先决条件

1. 您在组成卷组(VG)的底层物理卷(PV)上有足够的可用空间来支持条带。

流程

1. **可选：**显示卷组：

```
# vgs
  VG     #PV #LV #SN Attr   VSize   VFree
  myvg     2   1   0 wz--n- 271.31G 271.31G
```

2. **可选：**使用卷组中的整个空间来创建一个条带：

```
# lvcreate -n stripe1 -L 271.31G -i 2 myvg
  Using default stripesize 64.00 KB
  Rounding up size to full physical extent 271.31 GiB
```

3. **可选：**通过添加新物理卷来扩展 *myvg* 卷组：

```
# vgextend myvg /dev/sdc1
  Volume group "myvg" successfully extended
```

重复此步骤，根据您的条带类型和已用空间量来添加足够的物理卷。例如，对于使用整个卷组的双向条带，您需要至少添加两个物理卷。

4. 扩展作为 *myvg* VG 一部分的条状逻辑卷 *stripe1* ：

```
# lvextend myvg/stripe1 -L 542G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 542.00 GB
  Logical volume stripe1 successfully resized
```

您还可以扩展 *stripe1* 逻辑卷，来填充 *myvg* 卷组中所有未分配的空间：

```
# lvextend -l+100%FREE myvg/stripe1
  Size of logical volume myvg/stripe1 changed from 1020.00 MiB (255 extents) to <2.00 GiB
(511 extents).
  Logical volume myvg/stripe1 successfully resized.
```

验证

- 验证扩展的条状 LV 的新大小：

```
# lvs
  LV      VG    Attr     LSize      Pool   Origin Data%  Move Log Copy%  Convert
  stripe1 myvg  wi-ao---- 542.00 GB
```

第 7 章 自定义 LVM 报告

LVM 提供了广泛的配置和命令行选项来生成自定义报告，并过滤报告输出。您可以对输出进行排序，指定单元、使用选择条件，并更新 **lvm.conf** 文件以自定义 LVM 报告。

7.1. 控制 LVM 显示的格式

无论您使用 **pvs**、**lvs** 或 **vgs**，这些命令都确定显示的默认字段集和排序顺序。您可以通过执行以下命令控制这些命令的输出。

流程

- 使用 **-o** 选项更改 LVM 显示中的默认字段：

```
# pvs -o pv_name,pv_size,pv_free
  PV         PSize  PFree
  /dev/vdb1  17.14G 17.14G
  /dev/vdb2  17.14G 17.09G
  /dev/vdb3  17.14G 17.14G
```

- 使用 **-O** 选项对 LVM 显示进行排序：

```
# pvs -o pv_name,pv_size,pv_free -O pv_free
  PV         PSize  PFree
  /dev/vdb2 17.14G 17.09G
  /dev/vdb1  17.14G 17.14G
  /dev/vdb3  17.14G 17.14G
```

- 使用 **-O** 参数和 **-** 字符显示反向排序：

```
# pvs -o pv_name,pv_size,pv_free -O -pv_free
  PV         PSize  PFree
  /dev/vdb1  17.14G 17.14G
  /dev/vdb3  17.14G 17.14G
  /dev/vdb2  17.14G 17.09G
```

其他资源

- **lvmreport (7)**, **lvs (8)**, **vgs (8)**, 和 **pvs (8)** 手册页
- 为 LVM 报告显示指定单位
- 自定义 LVM 配置文件

7.2. 为 LVM 报告显示指定单位

您可以通过指定 report 命令的 **--units** 参数来查看以基数 2 或 10 为单位的 LVM 设备的大小。

基数 2 单位

默认单位以 2 的幂显示，即 1024 的倍数。您可以使用人类可读的(**r**) **<** 和 **>** 舍入指示符, 字节 ((**b**), 扇区 (**s**)，千字节(**k**), 兆字节 (**m**), G 字节 (**g**),TB 字节(**t**), PB 字节 (**p**), EB 字节(**e**) 和人类可读的(**h**)指定它。

未指定 **--units** 时，默认显示为 **r**。您可以通过在 **/etc/lvm/lvm.conf** 文件的全局部分中设置 units 参数来覆盖默认设置。

基数 10 单位

您可以通过大写单位规格(**R**、**B**、**S**、**K**、**M**、**G**、**T**、**P**、**E**、**H**)来指定要以 1000 倍显示的单位。

流程

- 为基数 2GB 单位指定 LVM 的单位：

```
# pvs --units g /dev/vdb
  PV         VG    Fmt  Attr PSize   PFree
  /dev/vdb   myvg  lvm2 a--  931.00g 930.00g

# vgs --units g myvg
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 931.00g 931.00g

# lvs --units g myvg
  LV    VG   Attr     LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg  wi-a---- 1.OOg
```

- 在输出中使用带有 **<** 或 **>** 前缀的 **r** 选项表示 LVM 的实际大小：

```
# vgs --units g myvg
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 931.00g 930.00g

# vgs --units r myvg
  VG   #PV #LV #SN Attr VSize    VFree
  myvg   1   1   0 wz-n <931.00g <930.00

# vgs myvg
  VG   #PV #LV #SN Attr VSize    VFree
  myvg   1   1   0 wz-n <931.00g <930.00g
```

r 单元的工作方式与 **h**（人类可读的）类似，但报告的值有一个 **<** 或 **>** 前缀，来表示实际大小略微大于或小于显示的大小。LVM 将十进制值进行舍入，导致报告了不精确的大小。

它还显示为何 **--units g** 或其他 **--units** 不总是显示准确的大小。它还显示 **r** 的主要目的，即 **<** 表示显示的大小不准确。在本例中，值不准确，因为 VG 大小不是一个精确的 GB 倍数，.01 也不是分数的一个准确表示。

- 为基数 10 GB 单位的 LVM 指定单位：

```
# pvs --units G /dev/vdb
  PV         VG   Fmt  Attr  PSize   PFree
  /dev/vdb   myvg lvm2 a--   999.65G 998.58G

# vgs --units G myvg
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 999.65G 998.58G
```

```
# lvs --units G myvg
  LV   VG   Attr     LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg wi-a---- 1.07G
```

- 指定扇区(**s**)，定义为 512 字节或自定义单位。以下示例将 **pvs** 命令的输出显示为几个扇区：

```
# pvs --units s
  PV         VG     Fmt  Attr PSize       PFree
  /dev/vdb   myvg   lvm2 a--  1952440320S 1950343168S
```

- 指定 MB (**m**)。以下示例显示了 **pvs** 命令的输出，单位为 4 MB：

```
# pvs --units 4m
  PV         VG     Fmt  Attr PSize      PFree
  /dev/vdb   myvg   lvm2 a--  238335.00U 238079.00U
```

7.3. 自定义 LVM 配置文件

通过编辑 **lvm.conf** 文件，您可以根据您的特定存储和系统要求自定义 LVM。例如，您可以使用 **lvm.conf** 修改过滤器设置、配置卷组自动激活、管理精简池或自动扩展快照。

流程

1. 显示默认的 **lvm.conf** 文件：

```
# lvmconfig --typeconfig default --withcomments
```

默认情况下，**lvm.conf** 文件只包含显示可能设置的注释。

2. 通过取消 **lvm.conf** 中设置的注释，根据您的要求自定义 **lvm.conf** 文件。以下设置侧重于更改某些命令的默认显示：

- 在 **lvm.conf** 文件中，调整 **lvs_cols** 参数来仅打印指定的字段：

```
{
  ...
  lvs_cols="lv_name,vg_name,lv_attr"
  ...
}
```

使用这个选项而不是 **lvs -o lv_name,vg_name,lv_attr** 命令，来避免不必要地频繁使用 **-o** 选项。

- 在 **lvm.conf** 文件中，使用 **compact_output=1** 设置避免打印 **pvs**、**vgs** 和 **lvs** 命令的空字段：

```
{
  ...
  compact_output = 1
  ...
}
```

3. 修改 **lvm.conf** 文件后查看默认值：

```
# lvmconfig --typeconfig diff
```

其他资源

- **lvm.conf(5)** 手册页

7.4. 定义 LVM 选择标准

选择条件是 **<field> <operator> <value>** 形式的一组语句，其使用比较运算符来定义具体字段的值。然后，处理或显示与选择条件匹配的对象。通过逻辑和分组运算符对语句进行合并。要定义选择条件，请使用 **-S** 或 **--select** 选项，后跟一个或多个语句。

有些 LVM 命令支持 **-S** 选项，来根据某些属性选择要处理哪些对象。这些对象可以是物理卷(PV)、卷组(VG)或逻辑卷(LV)。

-S 选项的工作原理是描述要处理的对象，而不是命名每个对象。这在处理许多对象时很有用，很难单独查找和命名每个对象，或者当搜索有复杂特征集的对象时。选择选项也可以用作快捷方式，来避免输入多个名称。

使用 **lvs -S help** 命令查看字段和可能的操作符的完整集合。使用任何报告或处理命令替换 **lvs**，以查看该命令的详情。

将选择条件与 LVM 报告和处理命令一起使用，以仅显示或处理满足所选条件的对象：

- 报告命令包括 **pvs**,**vgs**,**lvs**,**pvdisplay**,**vgdisplay**,**lvdisplay** 和 **dmsetup info -c**。
- 处理命令包括 **pvchange**,**vgchange**,**lvchange**,**vgimport**,**vgexport**,**vgremove** 和 **lvremove**。

流程

- 使用 **pvs** 命令的选择条件的示例：

```
# pvs
  PV          VG   Fmt  Attr PSize    PFree
  /dev/nvme2n1       lvm2 ---     1.00g   1.00g
  /dev/vdb1     myvg lvm2 a--  1020.00m 396.00m
  /dev/vdb2     myvg lvm2 a--  1020.00m 896.00m
```

```
# pvs -S name=~nvme
  PV          Fmt  Attr PSize PFree
  /dev/nvme2n1 lvm2 ---  1.00g 1.00g
```

```
# pvs -S vg_name=myvg
  PV         VG   Fmt  Attr PSize    PFree
  /dev/vdb1  myvg lvm2 a--  1020.00m 396.00m
  /dev/vdb2  myvg lvm2 a--  1020.00m 896.00m
```

- 使用 **lvs** 命令的选择条件的示例：

```
# lvs
  LV    VG   Attr       LSize   Cpy%Sync
  mylv  myvg -wi-a----- 200.00m
  lvol0 myvg -wi-a----- 100.00m
```

```
  lvol1 myvg -wi-a----- 100.00m
  lvol2 myvg -wi------- 100.00m
  rr    myvg rwi-a-r--- 120.00m 100.00
```

```
# lvs -S 'size > 100m && size < 200m'
  LV   VG   Attr       LSize   Cpy%Sync
  rr   myvg rwi-a-r--- 120.00m 100.00
```

```
# lvs -S name=~lvol[02]
  LV    VG   Attr       LSize
  lvol0 myvg -wi-a----- 100.00m
  lvol2 myvg -wi------- 100.00m
```

```
# lvs -S segtype=raid1
  LV   VG   Attr       LSize   Cpy%Sync
  rr   myvg rwi-a-r--- 120.00m 100.00
```

- 更高级的示例：

```
# lvchange --addtag mytag -S active=1
  Logical volume myvg/mylv changed.
  Logical volume myvg/lvol0 changed.
  Logical volume myvg/lvol1 changed.
  Logical volume myvg/rr changed.
```

```
# lvs -a -o lv_name,vg_name,attr,size,pool_lv,origin,role -S 'name!~_pmspare'
  LV          VG      Attr       LSize Pool Origin Role
  thin1       example Vwi-a-tz-- 2.00g tp          public,origin,thinorigin
  thin1s      example Vwi---tz-- 2.00g tp   thin1  public,snapshot,thinsnapshot
  thin2       example Vwi-a-tz-- 3.00g tp          public
  tp          example twi-aotz-- 1.00g             private
  [tp_tdata]  example Twi-ao---- 1.00g             private,thin,pool,data
  [tp_tmeta]  example ewi-ao---- 4.00m             private,thin,pool,metadata
```

```
# lvchange --setactivationskip n -S 'role=thinsnapshot && origin=thin1'
  Logical volume myvg/thin1s changed.
```

```
# lvs -a -S 'name=~_tmeta && role=metadata && size <= 4m'
  LV         VG      Attr       LSize
  [tp_tmeta] myvg    ewi-ao---- 4.00m
```

其他资源

- **lvmreport (7)** 手册页

第 8 章 在共享存储上配置 LVM

共享存储是可同时被多个节点访问的存储。您可以使用 LVM 管理共享存储。共享存储通常用于集群和高可用性设置，共享存储如何在系统上显示有两种常见场景：

- LVM 设备被附加到主机，并传给客户机虚拟机使用。在这种情况下，设备永远不会被主机使用，只由客户机虚拟机使用。
- 机器附加到存储区域网络(SAN)，例如使用光纤通道，且 SAN LUN 对多台机器可见：

8.1. 为虚拟机磁盘配置 LVM

要防止虚拟机存储暴露给主机，您可以配置 LVM 设备访问和 LVM **系统 ID**。您可以通过从主机中排除有问题的设备来实现此目的，这样可确保主机上的 LVM 看不到或不使用传给客户虚拟机的设备。您可以通过在 VG 中设置 LVM **系统 ID** 以匹配客户虚拟机，来防止在主机上意外使用虚拟机的 VG 。

流程

1. 在 **lvm.conf** 文件中，检查是否 **system.devices** 文件已启用：

```
use_devicesfile=1
```

2. 从主机的设备文件中排除有问题的设备：

```
$ lvmdevices --deldev <device>
```

3. 可选：您可以进一步保护 LVM 设备：
   a. 在主机和虚拟机中的 **lvm.conf** 文件中设置 LVM **系统 ID** 功能：

```
system_id_source = "uname"
```

   b. 设置 VG 的**系统 ID** 以匹配虚拟机 **系统 ID**。这样确保只有客户虚拟机能够激活 VG：

```
$ vgchange --systemid <VM_system_id> <VM_vg_name>
```

8.2. 将 LVM 配置为在一台机器上使用 SAN 磁盘

要防止 SAN LUN 被错误的机器使用，请从所有机器上的设备文件中排除 LUN，但要使用它们的机器除外。

您还可以通过在所有机器上配置 **系统 ID** ，并在 VG 中设置 **系统 ID** 以匹配使用它的机器，来保护 VG 不被错误的机器使用。

流程

1. 在 **lvm.conf** 文件中，检查是否 **system.devices** 文件已启用：

```
use_devicesfile=1
```

2. 从主机的设备文件中排除有问题的设备：

```
$ lvmdevices --deldev <device>
```

3. 在 **lvm.conf** 文件中设置 LVM **系统 ID** 功能：

```
system_id_source = "uname"
```

4. 设置 VG **的系统 ID**，来匹配使用此 VG 的机器的 **系统 ID** ：

```
$ vgchange --systemid <system_id> <vg_name>
```

8.3. 将 LVM 配置为使用 SAN 磁盘进行故障转移

您可以配置在机器间移动的 LUN，例如用于故障转移目的。您可以通过配置 LVM 设备文件，在可能使用设备的所有机器上的设备文件中包括 LUN，并在每台机器上配置 LVM **系统 ID** 来设置 LVM。

以下流程描述了初始 LVM 配置，来为故障转移完成设置 LVM，并在机器之间移动 VG，您需要配置 **pacemaker** 和 LVM 激活资源代理，该代理将自动修改 VG 的系统 ID，以匹配可以使用 VG 的机器的系统 ID。如需更多信息，请参阅 配置和管理高可用性集群。

流程

1. 在 **lvm.conf** 文件中，检查是否 **system.devices** 文件已启用：

```
use_devicesfile=1
```

2. 在主机的设备文件中包括有问题的设备：

```
$ lvmdevices --adddev <device>
```

3. 在所有机器的 **lvm.conf** 文件中设置 LVM **系统 ID** 功能：

```
system_id_source = "uname"
```

8.4. 配置 LVM，以在多台机器间共享 SAN 磁盘

使用 **lvmlockd** 守护进程和锁管理器（如 **dlm** 或 **sanlock**），您可以启用从多台机器访问 SAN 磁盘上的共享 VG。具体命令可能会因使用的锁管理器和操作系统而异。下面的流程描述了将 LVM 配置为在多个机器间共享 SAN 磁盘所需步骤的概述。

警告

在使用 **pacemaker** 时，必须使用 配置和管理高可用性集群 中显示的 pacemaker 步骤来配置和启动系统。

流程

1. 在 **lvm.conf** 文件中，检查是否 **system.devices** 文件已启用：

```
use_devicesfile=1
```

2. 对于将使用共享 LUN 的每台机器，在机器设备文件中添加 LUN：

```
$ lvmdevices --adddev <device>
```

3. 配置 **lvm.conf** 文件，以便在所有机器上使用 **lvmlockd** 守护进程：

```
use_lvmlockd=1
```

4. 在所有机器上启动 **lvmlockd** 守护进程文件。
5. 启动锁管理器以与 **lvmlockd** 一起使用，如所有机器上的 **dlm** 或 **sanlock**。
6. 使用命令 **vgcreate --shared** 创建一个新的共享 VG。
7. 在所有机器上使用 **vgchange --lockstart** 和 **vgchange --lockstop** 命令启动和停止对现有共享 VG 的访问。

其他资源

- **lvmlockd(8)** 手册页

8.5. 使用 STORAGE RHEL 系统角色创建共享的 LVM 设备

如果您希望多个系统同时访问同一个存储，则您可以使用 **storage** RHEL 系统角色创建共享的 LVM 设备。

这可带来以下显著优点：

- 资源共享
- 管理存储资源方面的灵活性
- 存储管理任务的简化

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。
- **lvmlockd** 已配置。如需更多信息，请参阅 配置 LVM，以在多台机器之间共享 SAN 磁盘。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Create shared LVM device
  hosts: managed-node-01.example.com
  become: true
```

```
tasks:
  - name: Create shared LVM device
    ansible.builtin.include_role:
      name: rhel-system-roles.storage
    vars:
      storage_pools:
        - name: vg1
          disks: /dev/vdb
          type: lvm
          shared: true
          state: present
          volumes:
            - name: lv1
              size: 4g
              mount_point: /opt/test1
      storage_safe_mode: false
      storage_use_partitions: true
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录

第 9 章 配置 RAID 逻辑卷

您可以使用逻辑卷管理器(LVM)创建和管理独立磁盘的冗余阵列(RAID)卷。

9.1. RAID 逻辑卷

逻辑卷管理器 (LVM) 支持独立磁盘阵列 (RAID) 级别 0、1、4、5、6 和 10。LVM RAID 卷有以下特征：

- LVM 会创建和管理利用多设备 (MD) 内核驱动程序的 RAID 逻辑卷。
- 您可以从阵列中临时分割 RAID1 镜像，并在之后将其合并到阵列中。
- LVM RAID 卷支持快照。

其他特征包括：

集群

RAID 逻辑卷不是集群感知的。
您可以只在一台机器中创建和激活 RAID 逻辑卷，但不能在多台机器中同时激活它们。

子卷

当您创建 RAID 逻辑卷（LV）时，LVM 会创建一个元数据子卷，它是阵列中的每个数据或奇偶校验子卷的大小的一个区块。
例如，创建一个双向 RAID1 阵列会导致两个元数据子卷（**lv_rmeta_0** 和 **lv_rmeta_1**）以及两个数据子卷（**lv_rimage_0** 和 **lv_rimage_1**）。同样，创建三向条带和一个隐式奇偶校验设备，RAID4 会产生四个元数据子卷(**lv_rmeta_0**, **lv_rmeta_1**, **lv_rmeta_2**, 和 **lv_rmeta_3**)，四个数据子卷(**lv_rimage_0**, **lv_rimage_1**, **lv_rimage_2** 和 **lv_rimage_3**)。

完整性

当 RAID 设备失败或者发生软崩溃时，可能会丢失数据。数据存储中的软崩溃意味着，从存储设备中检索的数据与写入到那个设备中的数据不同。在 RAID LV 中添加完整性可减少或防止软崩溃。如需更多信息，请参阅创建带有 DM 完整性的 RAID LV。

9.2. RAID 级别和线性支持

以下是 RAID 支持的配置，包括级别 0、1、4、5、6、10 和线性：

0 级

RAID 级别 0，通常称为条带化的数据映射技术。这意味着，要写入阵列的数据被分成条块，并在阵列的成员磁盘中写入，这样可以在成本低的情况下提供高的 I/O 性能，但不提供冗余。
RAID 级别 0 的实现只在成员设备间条状分布到阵列中最小设备的大小。就是说，如果您有多个设备，它们的大小稍有不同，那么每个设备的大小都被视为与最小设备的大小相同。因此，级别 0 阵列的常见存储容量是所有磁盘的总容量。如果成员磁盘具有不同的大小，RAID0 将使用可用区使用这些磁盘的所有空间。

1 级

RAID 级别 1 或称为镜像（mirroring），通过将相同数据写入阵列的每个磁盘来提供冗余，在每个磁盘上保留"镜像"副本。因为其简单且数据高度可用，RAID 1 仍然被广泛使用。级别 1 需要两个或者多个磁盘，它提供了很好的数据可靠性，提高了需要读取的应用程序的性能，但是成本相对高。
为了实现数据可靠性，需要向阵列中的所有磁盘写入相同的信息，所以 RAID 1 的成本会很高。与基于奇偶校验的其他级别（如级别 5）相比，空间的利用效率较低。然而，对空间利用率的牺牲提供了高性能：基于奇偶校验的 RAID 级别会消耗大量 CPU 资源以便获得奇偶校验，而 RAID 级别 1 只是一次向

多个 RAID 成员中写入同样数据，其对 CPU 的消耗较小。因此，在使用软件 RAID 的系统中，或系统中有其他操作需要大量使用 CPU 资源时，RAID 1 可能会比使用基于奇偶校验的 RAID 级别的性能更好。

级别 1 阵列的存储容量等于硬件 RAID 中最小镜像硬盘或者软件 RAID 中最小镜像分区的容量相同。级别 1 所提供的冗余性是所有 RAID 级别中最高的，因为阵列只需要在有一个成员可以正常工作的情况下就可以提供数据。

级别 4

级别 4 使用单一磁盘驱动器中的奇偶校验来保护数据。奇偶校验信息根据阵列中其余成员磁盘的内容计算。然后当阵列中的一个磁盘失败时，这个信息就可以被用来重建数据。然后，在出现问题的磁盘被替换前，使用被重建的数据就可以满足 I/O 的请求。在磁盘被替换后，可以在上面重新生成数据。
因为专用奇偶校验磁盘代表所有写交易到 RAID 阵列的固有瓶颈，因此在没有写回缓存等技术的情况下，级别 4 很少被使用。或者在特定情况下，系统管理员有意设计具有这个瓶颈的软件 RAID 设备，比如当阵列使用数据填充后没有写入事务的数组。因此，Anaconda 中并没有提供 RAID 4 这个选项。但是，如果需要，用户可以手动创建它。

硬件 RAID 4 的存储容量等于分区数量减一乘以最小成员分区的容量。RAID 4 阵列的性能是非对称的，即读的性能会好于写的性能。这是因为，写操作会在生成奇偶校验时消耗额外的 CPU 和主内存带宽，然后在将实际数据写入磁盘时也会消耗额外的总线带宽，因为您不仅写数据，而且还写奇偶校验。读操作只需要读取数据而不是奇偶校验，除非该阵列处于降级状态。因此，在正常操作条件下，对于相同数量的数据传输，读操作会对驱动器和计算机总线产生较少的流量。

5 级

这是最常见的 RAID 类型。通过在一个阵列的所有成员磁盘中分布奇偶校验，RAID 5 解除了级别 4 中原有的写入瓶颈。唯一性能瓶颈是奇偶校验计算过程本身。现代 CPU 可以非常快速地计算奇偶校验。但是，如果您在 RAID 5 阵列中有大量磁盘，以便在所有设备间合并数据传输速度非常高，则奇偶校验计算可能会成为瓶颈。
5 级具有非对称性能，读性能显著提高。RAID 5 的存储容量的计算方法与级别 4 的计算方法是一样的。

级别 6

如果数据的冗余性和保护性比性能更重要，且无法接受 RAID 1 的空间利用率低的问题，则通常会选择使用级别 6。级别 6 使用一个复杂的奇偶校验方式，可以在阵列中出现任意两个磁盘失败的情况下进行恢复。因为使用的奇偶校验方式比较复杂，软件 RAID 设备会对 CPU 造成较大负担，同时对写操作造成更大的负担。因此，与级别 4 和 5 相比，级别 6 的性能不对称性更严重。
RAID 6 阵列的总容量与 RAID 5 和 4 类似，但您必须从额外奇偶校验存储空间的设备数中减去 2 个设备（而不是 1 个）。

级别 10

这个 RAID 级别将级别 0 的性能优势与级别 1 的冗余合并。它还可减少在具有多于两个设备的 1 级阵列中发现的一些空间。对于 10 级，可以创建一个 3 个驱动器阵列，来仅存储每块数据的 2 个副本，然后允许整个阵列的大小为最小设备的 1.5 倍，而不是只等于最小设备（这与 3 设备 1 级阵列类似）。与 RAID 级别 6 相比，计算奇偶校验对 CPU 的消耗较少，但空间效率较低。
在安装过程中，不支持创建 RAID 10。您可在安装后手动创建。

线性 RAID

线性 RAID 是创建更大的虚拟驱动器的一组驱动器。
在线性 RAID 中，块会被从一个成员驱动器中按顺序分配，只有在第一个完全填充时才会进入下一个驱动器。这个分组方法不会提供性能优势，因为 I/O 操作不太可能在不同成员间同时进行。线性 RAID 也不提供冗余性，并会降低可靠性。如果有任何一个成员驱动器失败，则无法使用整个阵列，数据可能会丢失。该容量是所有成员磁盘的总量。

9.3. LVM RAID 片段类型

要创建 RAID 逻辑卷，您可以使用 **lvcreate** 命令的 **--type** 参数指定 RAID 类型。对于大多数用户，指定五个可用主类型之一 **raid1**、**raid4**、**raid5**、**raid6** 和 **raid10** 应足够。

下表描述了可能的 RAID 片段类型。

表 9.1. LVM RAID 片段类型

片段类型	描述
raid1	RAID1 镜像。当您指定 **-m** 参数而不指定条带时，这是 **lvcreate** 命令的 **--type** 参数的默认值。
raid4	RAID4 专用奇偶校验磁盘。
raid5_la	• RAID5 左非对称。 • 轮转奇偶校验 0 并分配数据。
raid5_ra	• RAID5 右非对称。 • 轮转奇偶校验 N 并分配数据.
raid5_ls	• RAID5 左对称。 • 它与 **raid5** 相同。 • 使用数据重启轮转奇偶校验 0。
raid5_rs	• RAID5 右对称。 • 使用数据重启轮转奇偶校验 N。
raid6_zr	• RAID6 零重启。 • 它与 **raid6** 相同。 • 数据重启时的旋转奇偶校验零（从左到右）。
raid6_nr	• RAID6 N 重启。 • 数据重启时的旋转奇偶校验 N （从左到右）。

片段类型	描述
raid6_nc	• RAID6 N 继续。 • 数据持续的旋转奇偶校验 N （从左到右）。
raid10	• 条带镜像。如果您指定了 **-m** 参数以及大于 1 的条带数，则这是 **lvcreate** 命令的 **--type** 参数的默认值。 • 镜像集合的条带。
raid0/raid0_meta	条带。RAID0 以条带大小的单位在多个数据子卷间分布逻辑卷数据。这可以提高性能。如果任何数据子卷失败，逻辑卷数据将会丢失。

9.4. 创建 RAID 逻辑卷

您可以根据您为 **-m** 参数指定的值，来创建具有不同副本数的 RAID1 阵列。同样，您可以使用 **-i** 参数为 RAID 0、4、5、6 和 10 逻辑卷指定条带数。您还可以使用 **-I** 参数指定条带大小。下面的步骤描述了创建不同类型的 RAID 逻辑卷的不同方法。

流程

- 创建一个双向 RAID。以下命令在卷组 *my_vg* 中创建一个名为 *my_lv* 的双向 RAID1 阵列，大小为 *1G* ：

```
# lvcreate --type raid1 -m 1 -L 1G -n my_lv my_vg
Logical volume "my_lv" created.
```

- 使用条带创建 RAID5 阵列。以下命令在卷组 *my_vg* 中创建具有三个条带的 RAID5 阵列和一个隐式奇偶校验驱动器，名为 *my_lv*，大小为 *1G*。请注意，您可以指定与 LVM 条带卷类似的条带数。自动添加正确奇偶校验驱动器数。

```
# lvcreate --type raid5 -i 3 -L 1G -n my_lv my_vg
```

- 使用条带创建 RAID6 阵列。以下命令在卷组 *my_vg* 中创建具有三个 3 个条带的 RAID6 阵列，以及名为 *my_lv* 的两个隐式奇偶校验驱动器，大小为 *1G* ：

```
# lvcreate --type raid6 -i 3 -L 1G -n my_lv my_vg
```

验证

- 显示 LVM 设备 *my_vg/my_lv*，它是一个双向 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices _my_vg_
  LV              Copy%  Devices
  my_lv             6.25    my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]          /dev/sde1(0)
```

```
[my_lv_rimage_1]        /dev/sdf1(1)
[my_lv_rmeta_0]         /dev/sde1(256)
[my_lv_rmeta_1]         /dev/sdf1(0)
```

其他资源

- **lvcreate (8)** 和 **lvmraid (7)** man page

9.5. 创建 RAID0 条带化逻辑卷

RAID0 逻辑卷以条的大小为单位，将逻辑卷数据分散到多个数据子卷中。下面的步骤创建了一个名为 *mylv* 的 LVM RAID0 逻辑卷，该逻辑卷在磁盘间条状分布数据。

先决条件

1. 您已创建了三个或者多个物理卷。有关创建物理卷的更多信息，请参阅 创建 LVM 物理卷。
2. 您已创建了卷组。如需更多信息，请参阅创建 LVM 卷组。

流程

1. 从现有卷组中创建 RAID0 逻辑卷。以下命令从卷组 *myvg* 中创建 RAID0 卷 *mylv*，大小为 *2G*，有三个条带，条带大小为 *4kB* ：

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv my_vg
  Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
  Logical volume "mylv" created.
```

2. 在 RAID0 逻辑卷中创建文件系统。以下命令在逻辑卷中创建 ext4 文件系统：

```
# mkfs.ext4 /dev/my_vg/mylv
```

3. 挂载逻辑卷并报告文件系统磁盘空间使用情况：

```
# mount /dev/my_vg/mylv /mnt

# df
Filesystem              1K-blocks     Used  Available  Use% Mounted on
/dev/mapper/my_vg-mylv   2002684     6168  1875072    1%   /mnt
```

验证

- 查看创建的 RAID0 剥离的逻辑卷：

```
# lvs -a -o +devices,segtype my_vg
  LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert Devices Type
  mylv my_vg rwi-a-r--- 2.00g mylv_rimage_0(0),mylv_rimage_1(0),mylv_rimage_2(0) raid0
  [mylv_rimage_0] my_vg iwi-aor--- 684.00m /dev/sdf1(0) linear
  [mylv_rimage_1] my_vg iwi-aor--- 684.00m /dev/sdg1(0) linear
  [mylv_rimage_2] my_vg iwi-aor--- 684.00m /dev/sdh1(0) linear
```

9.6. 使用 STORAGE RHEL 系统角色为 RAID LVM 卷配置条带大小

使用 **storage** 系统角色，您可以使用 Red Hat Ansible Automation Platform 在 RHEL 上为 RAID LVM 卷配置条带大小。您可以使用可用参数建立一个 Ansible playbook，来配置带有 RAID 的 LVM 池。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            raid_level: raid1
            raid_stripe_size: "256 KiB"
            state: present
```

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 文件
- **/usr/share/doc/rhel-system-roles/storage/** 目录
- 管理 RAID

9.7. 创建 RAID0 的参数

您可以使用 **lvcreate --type raid0[*meta*] --stripes *_Stripes* --stripesize *StripeSize VolumeGroup [PhysicalVolumePath]*** 命令创建 RAID0 条状逻辑卷。

下表描述了不同的参数，您可以在创建 RAID0 条状逻辑卷时使用它们。

表 9.2. 创建 RAID0 条状逻辑卷的参数

参数	描述
--type raid0[_meta]	指定 **raid0** 创建一个没有元数据卷的 RAID0 卷。指定**raid0_meta** 创建一个具有元数据卷的 RAID0 卷。因为 RAID0 是不方便的，因此它不会存储任何镜像数据块作为 RAID1/10，或者计算并存储任何奇偶校验块，因为 RAID4/5/6。因此，它不需要元数据卷来保持有关镜像或奇偶校验块重新同步进程的状态。在从 RAID0 转换到 RAID4/5/6/10 时，元数据卷是强制的。指定 **raid0_meta** 预先分配这些元数据卷，以防止相应的分配失败。
--stripes *Stripes*	指定在其中分割逻辑卷的设备数。
--stripesize *StripeSize*	以 KB 为单位指定每个条的大小。这是在移动到下一个设备前写入一个设备的数据量。
VolumeGroup	指定要使用的卷组。
PhysicalVolumePath	指定要使用的设备。如果没有指定，LVM 将选择 *Stripes* 选项指定的设备数，每个条带一个。

9.8. 软数据崩溃

数据存储中的软崩溃意味着，从存储设备中检索的数据与写入到那个设备中的数据不同。错误的数据可以在存储设备中无限期存在。在检索并尝试使用此数据之前，您可能不会发现此损坏数据。

根据配置类型，一个独立磁盘的冗余阵列 (RAID) 逻辑卷 (LV) 可防止设备失败时数据丢失。如果由 RAID 阵列组成的设备失败，可以从 RAID LV 一部分的其他设备中恢复数据。但是 RAID 配置不能保证数据本身的完整性。软崩溃、静默崩溃、软错误和静默错误用来描述，即使系统和软件仍继续按预期工作，但数据已损坏的情况的术语。

设备映射程序(DM)完整性与 RAID 级别 1、4、5、6 和 10 一起使用，用于缓解或防止软崩溃导致数据丢失。RAID 层可确保数据没有破坏的副本可以修复软崩溃错误。完整性层位于每个 RAID 镜像之上，而额外的子 LV 存储每个 RAID 镜像的完整性元数据或数据校验和。当您从带有完整性的 RAID LV 中检索数据时，完整性数据校验和会分析崩溃的数据。如果检测到崩溃，完整性层会返回一个错误消息，RAID 层会从另一个 RAID 镜像检索到非破坏的数据副本。RAID 层会在损坏的数据中自动重写非破坏的数据，以修复软崩溃。

当创建一个具有 DM 完整性或在现有 RAID LV 中添加完整性的 RAID LV 时，请考虑以下点：

- 完整性元数据需要额外的存储空间。对于每个 RAID 镜像，每个 500MB 数据都需要 4MB 的额外存储空间，因为校验和被添加到数据中。
- 添加 DM 完整性会因为访问数时延迟而影响到性能，有些 RAID 的配置会比其他 RAID 配置受到的影响更大。RAID1 配置通常比 RAID5 或其变体提供更好的性能。
- RAID 完整性块的大小也会影响性能。配置更大的 RAID 完整块可提供更好的性能。但是，一个较小的 RAID 完整性块可以提供更好的兼容性。

- 完整性有两种模式：**位图（bitmap）**或**日志（journal）**。**bitmap** 完整性模式通常比**journal** 模式提供更好的性能。

提示

如果您遇到性能问题，请使用带有完整性的 RAID1，或者测试特定 RAID 配置的性能以确保它满足您的要求。

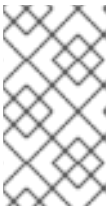
9.9. 创建带有 DM 完整性的 RAID LV

当您创建带有设备映射器 (DM) 完整性的 RAID LV 或者在现有 RAID LV 中添加完整性时，它会降低因为软崩溃而丢失数据的风险。在使用 LV 前，等待完整性同步和 RAID 元数据完成。否则，在后台进行的初始化可能会影响 LV 的性能。

流程

1. 创建具有 DM 完整性的 RAID LV。以下示例在 *my_vg* 卷组中创建一个名为 *test-lv* 的 RAID LV，可用大小为 *256M* 和 RAID 级别 *1*：

```
# lvcreate --type raid1 --raidintegrity y -L 256M -n test-lv my_vg
Creating integrity metadata LV test-lv_rimage_0_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_0_imeta" created.
Creating integrity metadata LV test-lv_rimage_1_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_1_imeta" created.
Logical volume "test-lv" created.
```



注意

要在现有 RAID LV 中添加 DM 完整性，请使用以下命令：

```
# lvconvert --raidintegrity y my_vg/test-lv
```

在 RAID LV 中添加完整性限制了您可以在那个 RAID LV 上执行的一些操作。

2. 可选：在执行某些操作前删除完整性。

```
# lvconvert --raidintegrity n my_vg/test-lv
Logical volume my_vg/test-lv has removed integrity.
```

验证

- 查看有关添加的 DM 完整性的信息：
  - 查看在 *my_vg* 卷组中创建的 test-lv RAID LV 的信息：

```
# lvs -a my_vg
  LV                  VG      Attr       LSize   Origin                Cpy%Sync
  test-lv             my_vg rwi-a-r--- 256.00m                          2.10
  [test-lv_rimage_0]        my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 93.75
  [test-lv_rimage_0_imeta]  my_vg ewi-ao----   8.00m
```

```
  [test-lv_rimage_0_iorig]  my_vg -wi-ao---- 256.00m
  [test-lv_rimage_1]        my_vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 85.94
 [...]
```

下面描述了此输出的不同选项：

g 属性

它是 Attr 列下的属性列表，表示 RAID 镜像使用完整性。完整性将校验和存储在 **_imeta** RAID LV 中。

Cpy%Sync 列

它指示顶层 RAID LV 和每个 RAID 镜像的同步进度。

RAID 镜像

它通过 **raid_image_N** 在 LV 列中指示。

LV 列

它确保对顶层 RAID LV 和每个 RAID 镜像显示 100% 同步进度。

- 显示每个 RAID LV 的类型：

```
# lvs -a my-vg -o+segtype
  LV                   VG      Attr       LSize   Origin               Cpy%Sync Type
  test-lv              my_vg rwi-a-r--- 256.00m                        87.96    raid1
  [test-lv_rimage_0]       my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 100.00
integrity
  [test-lv_rimage_0_imeta] my_vg ewi-ao----   8.00m                              linear
  [test-lv_rimage_0_iorig] my_vg -wi-ao---- 256.00m                              linear
  [test-lv_rimage_1]       my_vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 100.00
integrity
 [...]
```

- 有一个增量的计数器，它计算在每个 RAID 镜像上检测到的不匹配数。查看 *my_vg/test-lv* 下的 **rimage_0** 检测到的数据不匹配：

```
# lvs -o+integritymismatches my_vg/test-lv_rimage_0
  LV              VG      Attr       LSize   Origin               Cpy%Sync IntegMismatches
  [test-lv_rimage_0] my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig]    100.00
0
```

在这个示例中，完整性还没有检测到任何不匹配的数据，因此 **IntegMismatches** 计数器会显示 0 (0)。

- 查看 **/var/log/messages** 日志文件中的数据完整性信息，如下例所示：

例 9.1. 内核消息日志中的 dm-integrity 不匹配示例

```
device-mapper: integrity: dm-12:校验和在扇区 0x24e7 时失败
```

例 9.2. 来自内核消息日志的 dm-integrity 数据更正示例

```
md/raid1:mdX: 读取错误修正(8 扇区位在 dm-16 上 的 9448)
```

其他资源

- **lvcreate (8)** 和 **lvmraid (7)** man page

9.10. 最小和最大 I/O 速率选项

当您创建 RAID 逻辑卷时，使用 sync 操作初始化逻辑卷所需的后台 I/O 可能会加快对 LVM 设备的其它 I/O 操作，比如对卷组元数据的更新，特别是在创建很多 RAID 逻辑卷时。这会导致其它 LVM 操作速度下降。

您可以通过实施恢复节流来控制 RAID 逻辑卷初始化的速率。要控制执行 **sync** 操作的速率，请使用 **lvcreate** 命令的 **--minrecoveryrate** 和**-maxrecoveryrate** 选项为这些操作设置最小和最大 I/O 速率。

您可以按如下方式指定这些选项：

--maxrecoveryrate Rate[bBsSkKmMgG]

为 RAID 逻辑卷设置最大恢复率，使其不会加快 I/O 操作。将比率指定为阵列中每个设备的每秒数量。如果没有提供后缀，它会假设 kiB/sec/device。将恢复率设置为 0 表示它将不被绑定。

--minrecoveryrate Rate[bBsSkKmMgG]

为 RAID 逻辑卷设置最小恢复率，以确保 sync 操作的 I/O 达到最低吞吐量，即使存在大量 I/O。将比率指定为阵列中每个设备的每秒数量。如果您没有提供后缀，它会假设 kiB/sec/device。

例如，使用 **lvcreate --type raid10 -i *2* -m *1* -L *10G* --maxrecoveryrate *128* -n *my_lv my_vg*** 命令来创建一个双向 RAID10 阵列 *my_lv*，它位于卷组 *my_vg* 中，大小为 *3* 个条带，大小为*10G*，最大恢复率为 *128* kiB/sec/device。您还可以为 RAID 清理操作指定最小和最大恢复率。

9.11. 将线性设备转换为 RAID 逻辑卷

您可以将现有的线性逻辑卷转换为 RAID 逻辑卷。要执行此操作，请使用 **lvconvert** 命令的 **--type** 参数。

RAID 逻辑卷由元数据和数据子卷对组成。当您将线性设备转换为 RAID1 阵列时，它会创建一个新的元数据子卷，并将其与线性卷所在的同一物理卷中的原始逻辑卷相关联。其他镜像添加到 metadata/data 子卷对中。如果无法将与原始逻辑卷配对的元数据镜像放在同一个物理卷上，则 **lvconvert** 将失败。

流程

1. 查看需要转换的逻辑卷设备：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV     Copy%  Devices
  my_lv         /dev/sde1(0)
```

2. 将线性逻辑卷转换为 RAID 设备。以下命令将卷组 __my_vg 中的线性逻辑卷 *my_lv* 转换为双向 RAID1 阵列：

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
  Are you sure you want to convert linear LV my_vg/my_lv to raid1 with 2 images enhancing
resilience? [y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

验证

- 确定逻辑卷是否转换为 RAID 设备：

```
# lvs -a -o name,copy_percent,devices my_vg
```

```
  LV              Copy%  Devices
  my_lv           6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(256)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

其他资源

- **lvconvert (8)** 手册页

9.12. 将 LVM RAID1 逻辑卷转换为 LVM 线性逻辑卷

您可以将现有的 RAID1 LVM 逻辑卷转换为 LVM 线性逻辑卷。要执行此操作，请使用 **lvconvert** 命令并指定 **-m0** 参数。这会删除所有 RAID 数据子卷以及构成 RAID 阵列的所有 RAID 元数据子卷，保留顶层 RAID1 镜像作为线性逻辑卷。

流程

1. 显示现有 LVM RAID1 逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

2. 将现有的 RAID1 LVM 逻辑卷转换为 LVM 线性逻辑卷。以下命令将 LVM RAID1 逻辑卷 *my_vg/my_lv* 转换为 LVM 线性设备：

```
# lvconvert -m0 my_vg/my_lv
  Are you sure you want to convert raid1 LV my_vg/my_lv to type linear losing all resilience?
[y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

当您将 LVM RAID1 逻辑卷转换成 LVM 线性卷，,您可以指定要删除的物理卷。在以下示例中，**lvconvert** 命令指定要删除 */dev/sde1*，保留 */dev/sdf1* 作为组成线性设备的物理卷：

```
# lvconvert -m0 my_vg/my_lv /dev/sde1
```

验证

- 验证 RAID1 逻辑卷是否转换为 LVM 线性设备：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV    Copy%  Devices
  my_lv        /dev/sdf1(1)
```

其他资源

- **lvconvert (8)** 手册页

9.13. 将镜像 LVM 设备转换为 RAID1 逻辑卷

您可以将片段类型为 mirror 的现有镜像 LVM 设备转换为 RAID1 LVM 设备。要执行此操作，请使用带有 **--type** raid1 参数的 **lvconvert** 命令。这会将名为 **mimage** 的镜像子卷重命名为名为 **rimage** 的 RAID 子卷。

另外，它还会删除镜像日志，并为与对应的数据子卷在同一物理卷上的数据子卷创建名为 **rmeta** 的元数据子卷。

流程

1. 查看镜像逻辑卷 *my_vg/my_lv* 的布局：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           15.20 my_lv_mimage_0(0),my_lv_mimage_1(0)
  [my_lv_mimage_0]       /dev/sde1(0)
  [my_lv_mimage_1]       /dev/sdf1(0)
  [my_lv_mlog]           /dev/sdd1(0)
```

2. 将镜像逻辑卷 *my_vg/my_lv* 转换为 RAID1 逻辑卷：

```
# lvconvert --type raid1 my_vg/my_lv
Are you sure you want to convert mirror LV my_vg/my_lv to raid1 type? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

验证

- 验证镜像逻辑卷是否转换为 RAID1 逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(0)
  [my_lv_rmeta_0]        /dev/sde1(125)
  [my_lv_rmeta_1]        /dev/sdf1(125)
```

其他资源

- **lvconvert (8)** 手册页

9.14. 重新调整 RAID 逻辑卷大小的命令

您可以使用以下方法重新调整 RAID 逻辑卷的大小：

- 您可以使用 **lvresize** 或 **lvextend** 命令增加任何类型 RAID 逻辑卷的大小。这不会改变 RAID 镜像的数量。对于条带 RAID 逻辑卷，当您创建条带 RAID 逻辑卷时，会应用同样的条带舍入限制。
- 您可以使用 **lvresize** 或 **lvreduce** 命令减少任何类型的 RAID 逻辑卷的大小。这不会改变 RAID 镜像的数量。与 **lvextend** 命令一样，当您创建条带 RAID 逻辑卷时，会应用同样的条带舍入限制。
- 您可以使用 **lvconvert** 命令的 **--stripes *N*** 参数更改条状 RAID 逻辑卷（如 RAID4、RAID5、RAID6 或 RAID10）上的条带数。这会通过添加或删除条带的容量来增加或减少 RAID 逻辑卷的大

小。请注意，raid10 卷只能添加条带。这个功能是 RAID 重塑功能的一部分，您可以使用此功能更改 RAID 逻辑卷的属性，同时保持相同的 RAID 级别。

9.15. 更改现有 RAID1 设备中的镜像数

您可以更改现有 RAID1 阵列中的镜像数量，类似于更改 LVM 镜像实现中的镜像数量。

当您使用 **lvconvert** 命令将镜像添加到 RAID1 逻辑卷时，您可以执行以下操作：

- 指定生成的设备的镜像总数，
- 要添加到该设备的镜像数量，以及
- 可以指定新元数据/数据镜像对所在的物理卷。

流程

1. 显示 LVM 设备 *my_vg/my_lv*，它是一个双向 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(256)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

元数据子卷（名为 **rmeta**）始终与它们的数据子卷 **rimage** 存在于同一物理设备上。元数据/数据子卷对不会与 RAID 阵列中另一元数据/数据子卷对创建在同一物理卷上（除非您指定了 **--alloc anywhere**）。

2. 将双向 RAID1 逻辑卷 *my_vg/my_lv* 转换为三向 RAID1 逻辑卷：

```
# lvconvert -m 2 my_vg/my_lv
Are you sure you want to convert raid1 LV my_vg/my_lv to 3 images enhancing resilience?
[y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

以下是更改现有 RAID1 设备中的镜像数的几个示例：

- 您还可以在 RAID 中添加镜像时要使用的物理卷。以下命令通过指定物理卷 */dev/sdd1* 用于阵列，将双向 RAID1 逻辑卷 *my_vg/my_lv* 转换为三向 RAID1 逻辑卷：

```
# lvconvert -m 2 my_vg/my_lv /dev/sdd1
```

- 将三向 RAID1 逻辑卷转换成双向 RAID1 逻辑卷：

```
# lvconvert -m1 my_vg/my_lv
Are you sure you want to convert raid1 LV my_vg/my_lv to 2 images reducing resilience?
[y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

- 通过指定物理卷 */dev/sde1*，其中包含要删除的镜像，将三向 RAID1 逻辑卷转换成双向 RAID1 逻辑卷：

```
# lvconvert -m1 my_vg/my_lv /dev/sde1
```

另外，当您删除镜像及其关联的元数据子卷时，任何高数字镜像都会被切换以填充插槽。如果您从包含 **lv_rimage_0** 、**lv_rimage_1** 和 **lv_rimage_2** 的三向 RAID1 阵列中删除**lv_rimage_1**，则会产生一个由**lv_rimage_0** 和 **lv_rimage_1** 组成的 RAID1 阵列。**lv_rimage_2** 将会被重命名，并接管空插槽，成为 **lv_rimage_1**。

验证

- 在更改现有 RAID1 设备中的镜像数后查看 RAID1 设备：

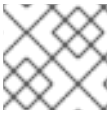
```
# lvs -a -o name,copy_percent,devices my_vg
  LV Cpy%Sync Devices
  my_lv 100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0] /dev/sdd1(1)
  [my_lv_rimage_1] /dev/sde1(1)
  [my_lv_rimage_2] /dev/sdf1(1)
  [my_lv_rmeta_0] /dev/sdd1(0)
  [my_lv_rmeta_1] /dev/sde1(0)
  [my_lv_rmeta_2] /dev/sdf1(0)
```

其他资源

- **lvconvert (8)** 手册页

9.16. 将 RAID 镜像分离为一个独立的逻辑卷

您可以分离 RAID 逻辑卷的镜像形成新的逻辑卷。和您从现有 RAID1 逻辑卷中删除 RAID 镜像一样，当您从设备的中间部分删除 RAID 数据子卷（及其关联的元数据子卷）时，会使用数字高的镜像来填充空的位置。因此，构成 RAID 阵列的逻辑卷上的索引号将是一个完整的整数序列。



注意

如果 RAID1 阵列还没有同步，您就无法分离 RAID 镜像。

流程

1. 显示 LVM 设备 *my_vg/my_lv*，它是一个双向 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           12.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

2. 将 RAID 镜像分成一个单独的逻辑卷：

- 以下示例将一个双向 RAID1 逻辑卷 *my_lv* 分成两个线性逻辑卷 *my_lv* 和 *new*：

```
# lvconvert --splitmirror 1 -n new my_vg/my_lv
Are you sure you want to split raid1 LV my_vg/my_lv losing all resilience? [y/n]: y
```

- 以下示例将一个三向 RAID1 逻辑卷 *my_lv* 分成一个双向 RAID1 逻辑卷 *my_lv* 和线性逻辑卷 *new* ：

```
# lvconvert --splitmirror 1 -n new my_vg/my_lv
```

验证

- 在分离 RAID 逻辑卷的镜像后查看逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV      Copy%  Devices
  my_lv          /dev/sde1(1)
  new            /dev/sdf1(1)
```

其他资源

- **lvconvert (8)** 手册页

9.17. 分割和合并 RAID 镜像

您可以使用 lvconvert 命令的 --trackchanges 参数（使用 **--trackchanges** 参数和 **lvconvert** 命令的 **--splitmirrors** 参数），临时分离 RAID1 阵列的镜像以进行只读使用。这可让您以后将镜像合并到阵列中，同时只重新同步那些自镜像被分割后更改的阵列的部分。

当您使用 **--trackchanges** 参数分离 RAID 镜像时，您可以指定要分离哪个镜像，但您不能更改要分离的卷的名称。另外，得到的卷有以下限制：

- 创建的新卷为只读。
- 不能调整新卷的大小。
- 不能重命名剩余的数组。
- 不能调整剩余的数组大小。
- 您可以独立激活新卷和剩余的阵列。

您可以合并分离的镜像。当您合并镜像时，只有自镜像分割后更改的阵列部分会被重新同步。

流程

1. 创建 RAID 逻辑卷：

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. 可选：查看创建的 RAID 逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
```

```
  [my_lv_rmeta_0]       /dev/sdb1(0)
  [my_lv_rmeta_1]       /dev/sdc1(0)
  [my_lv_rmeta_2]       /dev/sdd1(0)
```

3. 从创建的 RAID 逻辑卷中分割镜像，并跟踪对剩余的阵列的更改：

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
  my_lv_rimage_2 split from my_lv for read-only purposes.
  Use 'lvconvert --merge my_vg/my_lv_rimage_2' to merge back into my_lv
```

4. 可选：在分割镜像后查看逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sdc1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

5. 将卷合并回阵列中：

```
# lvconvert --merge my_vg/my_lv_rimage_1
  my_vg/my_lv_rimage_1 successfully merged back into my_vg/my_lv
```

验证

- 查看合并的逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sdc1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

其他资源

- **lvconvert (8)** 手册页

9.18. 设置 RAID 失败策略

根据 **/etc/lvm/lvm.conf** 文件中的 **raid_fault_policy** 字段首选项，LVM RAID 会自动处理设备故障。您可以根据要求，将 **raid_fault_policy** 字段设置为以下参数之一：

warn

您可以使用此参数，通过使用系统日志手动修复失败的设备并显示警告。
默认情况下，**raid_fault_policy** 字段的值在 **lvm.conf** 中是 **warn**。如果有足够的设备可以正常工作，RAID 逻辑卷将继续操作。

allocate

您可以使用此参数自动替换失败的设备。

9.18.1. 将 RAID 故障策略设置为 allocate

您可以将 **raid_fault_policy** 字段设置为 **/etc/lvm/lvm.conf** 文件中的 **allocate** 参数。使用这个首选项，系统会尝试使用卷组中的备用设备替换失败的设备。如果没有备用设备，系统日志会包含此信息。

流程

1. 查看 RAID 逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg

  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

2. 如果 */dev/sdb* 设备失败，请查看 RAID 逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg

  /dev/sdb: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and assumed devices.
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       [unknown](1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [...]
```

如果 */dev/sdb* 设备失败，您还可以查看系统日志中的错误信息。

3. 在 **lvm.conf** 文件中将 **raid_fault_policy** 字段设置为**allocate**：

```
# vi /etc/lvm/lvm.conf
raid_fault_policy = "allocate"
```

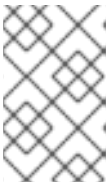
注意

如果将 **raid_fault_policy** 设置为 **allocate**，但没有备用设备，则分配会失败，逻辑卷保留原样。如果分配失败，您可以使用 **lvconvert --repair** 命令修复和替换失败的设备。如需更多信息，请参阅 在逻辑卷中替换失败的 RAID 设备。

验证

- 验证失败的设备现在是否可以使用卷组中的新设备进行替换：

```
# lvs -a -o name,copy_percent,devices my_vg
  Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
  LV            Copy%  Devices
  lv            100.00 lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
  [lv_rimage_0]        /dev/sdh1(1)
  [lv_rimage_1]        /dev/sdc1(1)
  [lv_rimage_2]        /dev/sdd1(1)
  [lv_rmeta_0]         /dev/sdh1(0)
  [lv_rmeta_1]         /dev/sdc1(0)
  [lv_rmeta_2]         /dev/sdd1(0)
```



注意

即使失败的设备现在被替换了，但显示仍然表示 LVM 没有找到失败的设备，因为设备还没有从卷组中删除。您可以通过执行 **vgreduce --removemissing *my_vg*** 命令从卷组中删除失败的设备。

其他资源

- **lvm.conf(5)** 手册页

9.18.2. 将 RAID 故障策略设置为 warn

您可以在 **lvm.conf** 文件中将 **raid_fault_policy** 字段设置为 **warn** 参数。有了这个首选项，系统会在系统日志中添加了一条表示失败的设备的警告。根据警告，您可以确定后续步骤。

流程

1. 查看 RAID 逻辑卷：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

2. 在 lvm.conf 文件中将 raid_fault_policy 字段设置为 warn ：

```
# vi /etc/lvm/lvm.conf
 # This configuration option has an automatic default value.
 raid_fault_policy = "warn"
```

3. 如果 */dev/sdb* 设备失败，查看系统日志以显示错误信息：

```
# grep lvm /var/log/messages

Apr 14 18:48:59 virt-506 kernel: sd 25:0:0:0: rejecting I/O to offline device
Apr 14 18:48:59 virt-506 kernel: I/O error, dev sdb, sector 8200 op 0x1:(WRITE) flags
0x20800 phys_seg 0 prio class 2
```

```
[...]
Apr 14 18:48:59 virt-506 dmeventd[91060]: WARNING: VG my_vg is missing PV 9R2TVV-
bwfn-Bdyj-Gucu-1p4F-qJ2Q-82kCAF (last written to /dev/sdb).
Apr 14 18:48:59 virt-506 dmeventd[91060]: WARNING: Couldn't find device with uuid
9R2TVV-bwfn-Bdyj-Gucu-1p4F-qJ2Q-82kCAF.
Apr 14 18:48:59 virt-506 dmeventd[91060]: Use 'lvconvert --repair my_vg/ly_lv' to replace
failed device.
```

如果 */dev/sdb* 设备失败，系统日志会显示错误消息。在这种情况下，LVM 将不会自动尝试通过替换其中一个镜像修复 RAID 设备。相反，如果设备失败，您可以使用 **lvconvert** 命令的 **--repair** 参数替换该设备。如需更多信息，请参阅 在逻辑卷中替换失败的 RAID 设备。

其他资源

- **lvm.conf(5)** 手册页

9.19. 在逻辑卷中替换 RAID 设备

您可以根据以下场景替换逻辑卷中的 RAID 设备：

- 替换正常工作的 RAID 设备。
- 替换逻辑卷中失败的 RAID 设备。

9.19.1. 替换正常工作的 RAID 设备

您可以使用 **lvconvert** 命令的 **--replace** 参数替换逻辑卷中正常工作的 RAID 设备。

警告

如果 RAID 设备失败，以下命令无法正常工作。

先决条件

- RAID 设备没有失败。

流程

1. 创建 RAID1 阵列：

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. 检查创建的 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
```

```
[my_lv_rimage_1]       /dev/sdb2(1)
[my_lv_rimage_2]       /dev/sdc1(1)
[my_lv_rmeta_0]        /dev/sdb1(0)
[my_lv_rmeta_1]        /dev/sdb2(0)
[my_lv_rmeta_2]        /dev/sdc1(0)
```

3. 根据您的要求，使用以下任一方法替换 RAID 设备：

   a. 通过指定要替换的物理卷来替换 RAID1 设备：

   ```
   # lvconvert --replace /dev/sdb2 my_vg/my_lv
   ```

   b. 通过指定要用于替换的物理卷来替换 RAID1 设备：

   ```
   # lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
   ```

   c. 通过指定多个替换参数来一次替换多个 RAID 设备：

   ```
   # lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
   ```

验证

1. 在指定要替换的物理卷后检查 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           37.50 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc2(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc2(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
```

2. 在指定用于替换的物理卷后检查 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           28.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

3. 一次替换多个 RAID 设备后检查 RAID1 阵列：

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           60.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rimage_2]       /dev/sde1(1)
```

```
[my_lv_rmeta_0]       /dev/sda1(0)
[my_lv_rmeta_1]       /dev/sdd1(0)
[my_lv_rmeta_2]       /dev/sde1(0)
```

其他资源

- **lvconvert(8)** 手册页

9.19.2. 在逻辑卷中替换失败的 RAID 设备

RAID 与传统的 LVM 镜像不同。如果是 LVM 镜像，请删除失败的设备。否则，当 RAID 阵列继续使用失败的设备运行时，镜像逻辑卷将挂起。对于 RAID1 以外的 RAID 级别，删除设备意味着转换到较低的 RAID 级别，例如从 RAID6 转换到 RAID5，或者从 RAID4 或 RAID5 转换到 RAID0。

您可以使用 **lvconvert** 命令的 **--repair** 参数替换 RAID 逻辑卷中作为物理卷的故障设备，而不是删除失败的设备并分配一个替换设备。

先决条件

- 卷组包含一个物理卷，它有足够的可用容量替换失败的设备。
  如果卷组中没有足够空闲扩展的物理卷，请使用 **vgextend** 工具添加一个新的、足够大的物理卷。

流程

1. 查看 RAID 逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
  LV              Cpy%Sync Devices
  my_lv             100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdc1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdc1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

2. 在 */dev/sdc* 设备失败后查看 RAID 逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and assumed devices.
  LV              Cpy%Sync Devices
  my_lv             100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         [unknown](1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          [unknown](0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

3. 替换失败的设备：

```
# lvconvert --repair my_vg/my_lv
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and assumed devices.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
Faulty devices in my_vg/my_lv successfully replaced.
```

4. 可选：手动指定替换失败设备的物理卷：

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 使用替换检查逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg

  /dev/sdc: open failed: No such device or address
  /dev/sdc1: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  LV              Cpy%Sync Devices
  my_lv           43.79    my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdb1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdb1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

在您从卷组中删除失败的设备前，LVM 工具仍然指示 LVM 无法找到失败的设备。

6. 从卷组中删除失败的设备：

```
# vgreduce --removemissing my_vg
```

验证

1. 删除失败的设备后查看可用的物理卷：

```
# pvscan
PV /dev/sde1 VG rhel_virt-506 lvm2 [<7.00 GiB / 0 free]
PV /dev/sdb1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
```

2. 在替换失败的设备后检查逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdb1(1)
```

```
[my_lv_rimage_2]        /dev/sdd1(1)
[my_lv_rmeta_0]         /dev/sde1(0)
[my_lv_rmeta_1]         /dev/sdb1(0)
[my_lv_rmeta_2]         /dev/sdd1(0)
```

其他资源

- **lvconvert (8)** 和 **vgreduce (8)** 手册页

9.20. 检查 RAID 逻辑卷中数据的一致性

LVM 提供对 RAID 逻辑卷的清理支持。RAID 清理是读取阵列中的所有数据和奇偶校验块的过程，并检查它们是否是分配的。**lvchange --syncaction repair** 命令对阵列启动后台同步操作。以下属性提供有关数据一致性的详情：

- **raid_sync_action** 字段显示当前 RAID 逻辑卷正在执行的同步操作。可以是以下值之一：

  idle

  完成所有 **sync** 操作（什么都不做）。

  resync

  在不干净的机器关闭后初始化或重新同步阵列。

  recover

  替换阵列中的设备。

  check

  查找阵列的不一致。

  repair

  查找并修复不一致.

- **raid_mismatch_count** 字段显示 **check** 操作过程中发现的差异数。
- **Cpy%Sync** 字段显示 **sync** 操作的进度。
- **lv_attr** 字段提供额外的指示。这个字段中的第 9 位显示逻辑卷的健康状况，它支持以下指示：

  m 或 mismatches

  表示 RAID 逻辑卷中存在差异。您可以在清理操作检测到 RAID 部分中的不一致后看到此字符。

  r 或 refresh

  表示 RAID 阵列中失败的设备，即使 LVM 可以读取设备标签，并认为设备可以正常工作。刷新逻辑卷通知内核该设备现在可用；如果您怀疑设备失败，则替换该设备。

流程

1. 可选：限制清理过程使用的 I/O 带宽。当您执行 RAID 清理操作时，**sync** 操作所需的后台 I/O 可能会将其他 I/O 分离到 LVM 设备，如对卷组元数据的更新。这可能导致其它 LVM 操作速度下降。
   您可以使用节流功能控制清理操作的速度。您可以将 **--maxrecoveryrate Rate[bBsSkKmMgG]** 或 **--minrecoveryrate Rate[bBsSkKmMgG]** 与 **lvchange --syncaction** 命令一起使用来设置恢复率。如需更多信息，请参阅 最小和最大 I/O 速率选项。

指定*比率*，格式为"数量/每秒/阵列中的每个设备"。如果没有后缀，选项会假定为 kiB/每秒/每个设备。

2. 显示阵列中未修复的差异的数量，没有修复它们：

```
# lvchange --syncaction check my_vg/my_lv
```

此命令对阵列启动后台同步操作。

3. 可选：查看 **var/log/syslog** 文件以了解内核消息。

4. 修正阵列中的差异：

```
# lvchange --syncaction repair my_vg/my_lv
```

这个命令修复或者替换 RAID 逻辑卷中失败的设备。您可以在执行此命令后查看 **var/log/syslog** 文件以了解内核消息。

验证

1. 显示有关清理操作的信息：

```
# lvs -o +raid_sync_action,raid_mismatch_count my_vg/my_lv
LV    VG    Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
SyncAction Mismatches
my_lv my_vg rwi-a-r--- 500.00m                    100.00           idle        0
```

其他资源

- **lvchange (8)** 和 **lvmraid (7)** 手册页
- 最小和最大 I/O 速率选项

9.21. 将 RAID 逻辑卷转换为另一个 RAID 级别

LVM 支持 RAID 接管，这意味着将 RAID 逻辑卷从一个 RAID 级别转换到另一个 RAID 级别，例如从 RAID 5 转换到 RAID 6。您可以更改 RAID 级别以增加或减少对设备故障的恢复能力。

流程

1. 创建 RAID 逻辑卷：

```
# lvcreate --type raid5 -i 3 -L 500M -n my_lv my_vg
Using default stripesize 64.00 KiB.
Rounding size 500.00 MiB (125 extents) up to stripe boundary size 504.00 MiB (126 extents).
Logical volume "my_lv" created.
```

2. 查看 RAID 逻辑卷：

```
# lvs -a -o +devices,segtype
  LV            VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices                                                 Type
  my_lv         my_vg       rwi-a-r--- 504.00m                                100.00
```

```
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0) raid5
  [my_lv_rimage_0] my_vg         iwi-aor--- 168.00m
/dev/sda(1)                                          linear
```

3. 将 RAID 逻辑卷转换为另一个 RAID 级别：

```
# lvconvert --type raid6 my_vg/my_lv
Using default stripesize 64.00 KiB.
Replaced LV type raid6 (same as raid6_zr) with possible type raid6_ls_6.
Repeat this command to convert to raid6 after an interim conversion has finished.
Are you sure you want to convert raid5 LV my_vg/my_lv to raid6_ls_6 type? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

4. 可选：如果这个命令提示重复转换，请运行：

```
# lvconvert --type raid6 my_vg/my_lv
```

验证

1. 查看具有转换的 RAID 级别的 RAID 逻辑卷：

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices                                                                  Type
  my_lv           my_vg       rwi-a-r--- 504.00m                               100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0),my_lv_rimage_
4(0) raid6
  [my_lv_rimage_0] my_vg         iwi-aor--- 172.00m
/dev/sda(1)                                                      linear
```

其他资源

- **lvconvert (8)** 和 **lvmraid (8)** 手册页

9.22. 对 RAID1 逻辑卷的 I/O 操作

您可以使用 **lvchange** 命令的 **--writemostly** 和 **--writebehind** 参数来控制对 RAID1 逻辑卷中设备的 I/O 操作。以下是使用这些参数的格式：

--[raid]writemostly PhysicalVolume[:{t|y|n}]

将 RAID1 逻辑卷中的设备标记为 **write-mostly**，并避免对这些驱动器的所有读操作（除非有必要）。设置此参数会使驱动器中的 I/O 操作数量保持最小。使用 **lvchange --writemostly *dev/sdb my_vg/ly_lv*** 命令设置此参数。
您可以使用以下方法设置 **writemostly** 属性：

:y

默认情况下，对于逻辑卷中指定的物理卷， **writemostly** 属性的值是 yes。

:n

要删除 **writemostly** 标志，请将 **:n** 附加到物理卷中。

:t

要切换 **writemostly** 属性的值，请指定 **--writemostly** 参数。您可以在单个命令中多次使用这个参数，一次为逻辑卷中的所有物理卷切换 **writemostly** 属性。

--[raid]writebehind IOCount

指定标记为 **writemostly** 的待处理写的最大数。这些是适用于 RAID1 逻辑卷中设备的写操作的数量。在超出此参数值后，在 RAID 阵列通知所有写入作完成前，对组成的设备的所有写操作都会同步完成。您可以使用 **lvchange --writebehind *100 my_vg/ly_lv*** 命令设置此参数。将 **writemostly** 属性的值设置为零来清除首选项。使用这个设置，系统会任意选择值。

9.23. 重塑 RAID 卷

RAID 重塑意味着更改 RAID 逻辑卷的属性，而不更改 RAID 级别。您可以更改的一些属性包括 RAID 布局、条带大小和条带数。

流程

1. 创建 RAID 逻辑卷：

```
# lvcreate --type raid5 -i 2 -L 500M -n my_lv my_vg

Using default stripesize 64.00 KiB.
Rounding size 500.00 MiB (125 extents) up to stripe boundary size 504.00 MiB (126 extents).
Logical volume "my_lv" created.
```

2. 查看 RAID 逻辑卷：

```
# lvs -a -o +devices

LV              VG    Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
my_lv           my_vg rwi-a-r--- 504.00m                             100.00           my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0] my_vg iwi-aor--- 252.00m                                              /dev/sda(1)
[my_lv_rimage_1] my_vg iwi-aor--- 252.00m                                              /dev/sdb(1)
[my_lv_rimage_2] my_vg iwi-aor--- 252.00m                                              /dev/sdc(1)
[my_lv_rmeta_0]  my_vg ewi-aor---   4.00m                                              /dev/sda(0)
[my_lv_rmeta_1]  my_vg ewi-aor---   4.00m                                              /dev/sdb(0)
[my_lv_rmeta_2]  my_vg ewi-aor---   4.00m                                              /dev/sdc(0)
```

3. 可选：查看 RAID 逻辑卷的 **stripes** 镜像和**stripesize** ：

```
# lvs -o stripes my_vg/my_lv
  #Str
    3
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  64.00k
```

4. 根据您的要求，使用以下方法修改 RAID 逻辑卷的属性：

a. 修改 RAID 逻辑卷的 **stripes** 镜像：

```
# lvconvert --stripes 3 my_vg/my_lv
Using default stripesize 64.00 KiB.
WARNING: Adding stripes to active logical volume my_vg/my_lv will grow it from 126 to 189 extents!
Run "lvresize -l126 my_vg/my_lv" to shrink it or use the additional capacity.
Are you sure you want to add 1 images to raid5 LV my_vg/my_lv? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

b. 修改 RAID 逻辑卷的 **stripesize** ：

```
# lvconvert --stripesize 128k my_vg/my_lv
  Converting stripesize 64.00 KiB of raid5 LV my_vg/my_lv to 128.00 KiB.
Are you sure you want to convert raid5 LV my_vg/my_lv? [y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

c. 修改 **maxrecoveryrate** 和 **minrecoveryrate** 属性：

```
# lvchange --maxrecoveryrate 4M my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

```
# lvchange --minrecoveryrate 1M my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

d. 修改 **syncaction** 属性：

```
# lvchange --syncaction check my_vg/my_lv
```

e. 修改 **writemostly** 和 **writebehind** 属性：

```
# lvchange --writemostly /dev/sdb my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

```
# lvchange --writebehind 100 my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

验证

1. 查看 RAID 逻辑卷的 **stripes** 镜像和**stripesize** ：

```
# lvs -o stripes my_vg/my_lv
  #Str
    4
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  128.00k
```

2. 修改 **maxrecoveryrate** 属性后查看 RAID 逻辑卷：

```
# lvs -a -o +raid_max_recovery_rate
```

```
  LV              VG       Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert MaxSync
  my_lv           my_vg    rwi-a-r---  10.00g                                    100.00          4096
  [my_lv_rimage_0] my_vg    iwi-aor---  10.00g
  [...]
```

3. 修改 **minrecoveryrate** 属性后查看 RAID 逻辑卷：

```
# lvs -a -o +raid_min_recovery_rate
  LV              VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert MinSync
  my_lv           my_vg  rwi-a-r---  10.00g                                    100.00          1024
  [my_lv_rimage_0] my_vg  iwi-aor---  10.00g
  [...]
```

4. 修改 **syncaction** 属性后查看 RAID 逻辑卷：

```
# lvs -a
  LV              VG      Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
  my_lv           my_vg   rwi-a-r---  10.00g                                    2.66
  [my_lv_rimage_0] my_vg   iwi-aor---  10.00g
  [...]
```

其他资源

- **lvconvert (8)** 和 **lvmraid (8)** 手册页

9.24. 在 RAID 逻辑卷中更改区域大小

当您创建 RAID 逻辑卷时，**/etc/lvm/lvm.conf** 文件中的 **raid_region_size** 参数代表 RAID 逻辑卷的区域大小。创建 RAID 逻辑卷后，您可以更改卷的区域大小。此参数定义跟踪脏或干净状态的粒度。位映射中的脏位定义 RAID 卷脏关闭后要同步的工作集，例如系统故障。

如果将 **raid_region_size** 设置为更高的值，它会减小位映射的大小以及阻塞。但它会影响重新同步区域期间的**写**操作，因为写入 RAID 会延迟，直到同步区域完成。

流程

1. 创建 RAID 逻辑卷：

```
# lvcreate --type raid1 -m 1 -L 10G test
  Logical volume "lvol0" created.
```

2. 查看 RAID 逻辑卷：

```
# lvs -a -o +devices,region_size

LV              VG      Attr     LSize Pool Origin Data% Meta% Move Log   Cpy%Sync Convert
Devices                          Region
lvol0           test rwi-a-r--- 10.00g                          100.00
lvol0_rimage_0(0),lvol0_rimage_1(0)  2.00m
[lvol0_rimage_0]  test iwi-aor--- 10.00g                                              /dev/sde1(1)
```

```
0
[lvol0_rimage_1]  test iwi-aor--- 10.00g                                    /dev/sdf1(1)
0
[lvol0_rmeta_0]   test ewi-aor---  4.00m                                    /dev/sde1(0)
0
[lvol0_rmeta_1]   test ewi-aor---  4.00m
```

Region 列显示 raid_region_size 参数的值。

3. 可选：查看 **raid_region_size** 参数的值：

```
# cat /etc/lvm/lvm.conf | grep raid_region_size

# Configuration option activation/raid_region_size.
 # raid_region_size = 2048
```

4. 更改 RAID 逻辑卷的区域大小：

```
# lvconvert -R 4096K my_vg/my_lv

Do you really want to change the region_size 512.00 KiB of LV my_vg/my_lv to 4.00 MiB?
[y/n]: y
  Changed region size on RAID LV my_vg/my_lv to 4.00 MiB.
```

5. 重新同步 RAID 逻辑卷：

```
# lvchange --resync my_vg/my_lv

Do you really want to deactivate logical volume my_vg/my_lv to resync it? [y/n]: y
```

验证

1. 查看 RAID 逻辑卷：

```
# lvs -a -o +devices,region_size

LV              VG   Attr       LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices                          Region
lvol0           test rwi-a-r--- 10.00g                                    6.25
lvol0_rimage_0(0),lvol0_rimage_1(0)  4.00m
[lvol0_rimage_0] test iwi-aor--- 10.00g                                             /dev/sde1(1)
0
[lvol0_rimage_1] test iwi-aor--- 10.00g                                             /dev/sdf1(1)
0
[lvol0_rmeta_0]  test ewi-aor---  4.00m                                             /dev/sde1(0)
0
```

Region 列指示 **raid_region_size** 参数的更改值。

2. 查看 **lvm.conf** 文件中的 **raid_region_size** 参数的值：

```
# cat /etc/lvm/lvm.conf | grep raid_region_size

# Configuration option activation/raid_region_size.
```

```
# raid_region_size = 4096
```

其他资源

- **lvconvert(8)** 手册页

第 10 章 逻辑卷快照

使用 LVM 快照功能，您可以在特定时刻创建卷(例如*/dev/sda*)的虚拟镜像，而不会导致服务中断。

10.1. 快照卷的概述

当您制作快照后修改原始卷（源头）时，快照功能会对修改的数据区域创建一个副本来作为更改前的状态，以便可以重建卷的状态。当您创建快照时，需要对原始卷有完全的读写权限。

因为快照只复制创建快照后更改的数据区域，因此快照功能只需要少量的存储。例如，对于很少更新的原始卷，原始容量的 3-5% 就足以进行快照维护。它不提供备份过程的替代品。快照副本是虚拟副本，不是实际的介质备份。

快照的大小控制为存储原始卷的更改而预留的空间量。例如：如果您创建了一个快照，然后完全覆盖了原始卷，则快照应至少与原始卷大小一样方可保存更改。您应该定期监控快照的大小。例如：一个以读为主的短期快照（如 **/usr**）需要的空间小于卷的长期快照，因为它包含很多写入，如 **/home**。

如果快照满了，则快照就会失效，因为它无法跟踪原始卷的变化。但是，您可以将 LVM 配置为每当使用量超过 **snapshot_autoextend_threshold** 值时就自动扩展快照，以免快照失效。快照是完全可调整大小的，您可以执行以下操作：

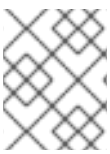
- 如果您有存储容量，您可以增加快照卷的大小，以防止其被删除。
- 如果快照卷大于您的需要，您可以缩小卷的大小来释放空间，以供其他逻辑卷使用。

快照卷提供以下优点：

- 大多数情况下，当需要对逻辑卷执行备份时制作快照，而不用停止持续更新数据的活动系统。
- 您可以在快照文件系统上执行 **fsck** 命令来检查文件系统的完整性，并确定原始文件系统是否需要文件系统修复。
- 由于快照是读/写的，因此您可以通过制作快照并对快照运行测试来对生产数据测试应用程序，而无需接触真实数据。
- 您可以创建 LVM 卷来用于 Red Hat Virtualization 。您可以使用 LVM 快照来创建虚拟客户端镜像的快照。这些快照可方便修改现有客户虚拟机或者使用最小附加存储创建新客户虚拟机。

10.2. 创建原始卷的快照

使用 **lvcreate** 命令创建原始卷（原始）的快照。卷快照是可写的。默认情况下，与精简配置的快照相比，快照卷在正常激活命令过程中是使用原始卷激活的。LVM 不支持创建大于原始卷大小和卷所需的元数据大小的总和。如果您指定了大于这个总和的快照卷，则 LVM 会创建一个原始卷的大小所需的快照卷。



注意

集群中的节点不支持 LVM 快照。您不能在共享卷组中创建快照卷。然而，如果您需要在共享逻辑卷中创建一致的数据备份，您可以单独激活该卷，然后创建快照。

以下流程创建了一个名为 *origin* 的原始逻辑卷和一个名为 *snap* 的原始卷的快照卷。

先决条件

- 您已创建了卷组 *vg001*。如需更多信息，请参阅创建 LVM 卷组。

流程

1. 从卷组 *vg001* 创建一个名为 *origin* 的逻辑卷：

```
# lvcreate -L 1G -n origin vg001
Logical volume "origin" created.
```

2. 创建 */dev/vg001/origin* 的一个名为 *snap* 的快照逻辑卷，大小为 *100 MB* ：

```
# lvcreate --size 100M --name snap --snapshot /dev/vg001/origin
  Logical volume "snap" created.
```

您还可以使用 **-L** 参数，而不是使用 **--size**，**-n**，而不是使用 **--name**，**-s**，而不是使用 **--snapshot** 来创建快照。

如果原始逻辑卷包含一个文件系统，您可以将快照逻辑卷挂载在任意目录上，以访问文件系统的内容，以便在原始文件系统持续更新时运行备份。

3. 显示原始卷以及当前使用的快照卷的百分比：

```
# lvs -a -o +devices
  LV      VG    Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
Devices
  origin  vg001 owi-a-s---   1.00g                                                     /dev/sde1(0)
  snap    vg001 swi-a-s--- 100.00m      origin 0.00                                    /dev/sde1(256)
```

您还可以使用 **lvdisplay */dev/vg001/origin*** 命令显示逻辑卷 */dev/vg001/origin* 以及所有快照逻辑卷及其状态，如 active 或 inactive。

警告

在写入原始 LV 后，快照 LV 中的空间会被消耗。**lvs** 命令在 **Data%** data_percent 字段值中报告当前快照空间使用率。如果快照空间达到 100%，则快照将变为无效且不可用。

在 **Attr** 列的第五个位置或在 **lvs** 中的 **lv_snapshot_invalid** 报告字段中用 **I** 报告一个无效的快照。您可以使用 **lvremove** 命令删除无效的快照。

4. 可选：在其空间变为 100% 满前扩展快照，并使用以下选项之一使其变为无效：

- 使用 **/etc/lvm.conf** 文件中的以下参数将 LVM 配置为自动扩展快照：

snapshot_autoextend_threshold

在使用量超过为此参数设置的值后扩展快照。默认情况下，它被设置为 100，这会禁用自动扩展。此参数的最小值为 50。

snapshot_autoextend_percent

为快照添加额外的空间，其为其当前大小的百分比。默认情况下，它被设置为 20。

在以下示例中，在设置以下参数后，当其用量超过 700M 时，创建的 1G 快照扩展到 1.2G：

例 10.1. 自动扩展快照

```
# vi /etc/lvm.conf
snapshot_autoextend_threshold = 70
snapshot_autoextend_percent = 20
```

注意

这个功能需要卷组中有未分配的空间。快照的自动扩展不会将快照卷的大小增加到超出快照所需的最大计算值。一旦快照增长到足够大来覆盖原始数据后，便不会再监控它是否发生了自动扩展。

- 使用 **lvextend** 命令手动扩展此快照：

```
# lvextend -L+100M /dev/vg001/snap
```

其他资源

- **lvcreate(8)**、**lvextend(8)** 和 **lvs(8)** 手册页
- **/etc/lvm/lvm.conf** 文件

10.3. 将快照合并到其原始卷中

使用 **lvconvert** 命令和 **--merge** 选项，将快照合并到其原始（源头）卷中。如果您丢失了数据或文件，您可以执行系统回滚，否则的话需要将系统恢复到之前的状态。合并快照卷后，得到的逻辑卷具有原始卷的名称、次要号码和 UUID。在合并过程中，对原始卷的读取和写入将会被指向要合并的快照。当合并完成后，会删除合并的快照。

如果原始卷和快照卷都没有打开且处于活跃状态，则合并会立即开始。否则，合并会在原始卷或快照激活后，或两者都关闭后开始。您可以在原始卷激活后，将快照合并到一个不能关闭的原始卷中，如 **root** 文件系统。

流程

1. 合并快照卷。以下命令将快照卷 *vg001/snap* 合并到其 *origin* 中：

```
# lvconvert --merge vg001/snap
Merging of volume vg001/snap started.
  vg001/origin: Merged: 100.00%
```

2. 查看原始卷：

```
# lvs -a -o +devices
  LV      VG    Attr       LSize  Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices
  origin vg001  owi-a-s---  1.00g                                       /dev/sde1(0)
```

其他资源

- **lvconvert(8)** 手册页

10.4. 使用快照 RHEL 系统角色创建 LVM 快照

使用新的 **快照** RHEL 系统角色，您现在可以创建 LVM 快照。它还通过设置 **snapshot_lvm_action** 参数来检查 快照是否有足够空间用于创建快照，且没有与其名称冲突。要挂载创建的快照，将 **snapshot_lvm_action** 设置为 **mount**。

注意

目前，**快照** RHEL 系统角色不支持精简 LVM 快照。

在以下示例中，设置了 **nouuid** 选项，且仅在使用 XFS 文件系统时才需要。使用 XFS 时，不支持同时挂载具有相同 UUID 的多个文件系统。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

```
---
- name: Run the snapshot system role
  hosts: managed-node-01.example.com
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
          percent_space_required: 25
          mountpoint: /data1_snapshot
          options: nouuid
          mountpoint_create: true
        - name: data2 snapshot
          vg: data_vg
          lv: data2
          percent_space_required: 25
          mountpoint: /data2_snapshot
          options: nouuid
          mountpoint_create: true

  tasks:
    - name: Create a snapshot set
      vars:
        snapshot_lvm_action: snapshot
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"
```

```
  - name: Verify the set of snapshots for the LVs
    vars:
      snapshot_lvm_action: check
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"
      snapshot_lvm_verify_only: true

  - name: Mount the snapshot set
    vars:
      snapshot_lvm_action: mount
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"

roles:
  - redhat.rhel_system_roles.snapshot
```

在这里，**snapshot_lvm_set** 参数描述了同一卷组中的特定逻辑卷(LV)。您还可以在设置此参数时指定来自不同 VG 的 LV。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** file
- **/usr/share/doc/rhel-system-roles/snapshot/** directory

10.5. 使用快照 RHEL 系统角色卸载 LVM 快照

您可以通过将 **snapshot_lvm_action** 参数设置为 **umount** 来卸载特定的快照或所有快照。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** ：
   - 卸载特定的 LVM 快照：

```
---
- name: Unmount a snapshot
```

```
hosts: all
tasks:
  - name: Unmount data_vg/data2
    vars:
      snapshot_lvm_snapset_name: snapset1
      snapshot_lvm_action: umount
      snapshot_lvm_vg: data_vg
      snapshot_lvm_lv: data2
      snapshot_lvm_mountpoint: /data2_snapshot

  roles:
    - redhat.rhel_system_roles.snapshot
```

在这里，**snapshot_lvm_lv** 参数描述了特定的逻辑卷(LV)，**snapshot_lvm_vg** 参数描述了特定的卷组(VG)。

- 卸载一组 LVM 快照：

```
---
- name: Unmount a set of snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Unmount a snapshot set
      vars:
        snapshot_lvm_action: umount
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"

  roles:
    - redhat.rhel_system_roles.snapshot
```

此处，**snapshot_lvm_set** 参数描述了同一 VG 中的特定 LV。您还可以在设置此参数时指定来自不同 VG 的 LV。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** file
- **/usr/share/doc/rhel-system-roles/snapshot/** directory

10.6. 使用快照 RHEL 系统角色扩展 LVM 快照

使用新的 **snapshot** RHEL 系统角色，您现在可以通过将 **snapshot_lvm_action** 参数设置为 **扩展 来扩展** LVM 快照。您可以将 **snapshot_lvm_percent_space_required** 参数设置为在扩展快照后分配给快照的所需空间。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/**playbook.yml** ：

   - 通过为 **percent_space_required** 参数指定值来扩展所有 LVM 快照：

```
---
- name: Extend all snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapshot
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Extend the snapshot set
      vars:
        snapshot_lvm_percent_space_required: 40
        snapshot_lvm_all_vgs: true
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"
        snapshot_lvm_action: extend

  roles:
    - redhat.rhel_system_roles.snapshot
```

   在这里，**snapshot_lvm_all_vgs** 参数描述了所有卷组(VG)中的所有逻辑卷(LV)。**snapshot_lvm_set** 参数描述了来自同一 VG 的特定 LV。

   - 通过将每个卷组和逻辑卷对的 **percent_space_required** 设置为不同的值来扩展 LVM 快照：

```
---
```

```
- name: Extend the snapshot
  hosts: all
  tasks:
    - name: Extend data1 LV by 30%
      vars:
        snapshot_lvm_snapset_name: snapset1
        percent_space_required: 30
        snapshot_lvm_action: extend
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data1

    - name: Extend data2 LV by 40%
      vars:
        snapshot_lvm_snapset_name: snapset2
        percent_space_required: 40
        snapshot_lvm_action: extend
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data2

  roles:
    - redhat.rhel_system_roles.snapshot
```

此处，**snapshot_lvm_lv** 参数描述了特定的 LV，**snapshot_lvm_vg** 参数描述了特定的 VG。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

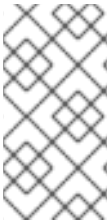
```
$ ansible-playbook ~/playbook.yml
```

其他资源

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** file
- **/usr/share/doc/rhel-system-roles/snapshot/** directory

10.7. 使用快照 RHEL 系统角色恢复 LVM 快照

使用新的 **snapshot** RHEL 系统角色，现在您可以通过将 **snapshot_lvm_action** 参数设置为恢复，将 LVM 快照恢复到其原始卷。



注意

如果逻辑卷和快照卷都没有打开并激活，则恢复操作会立即启动。否则，它会在原始卷或快照被激活且两者都关闭后启动。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 sudo 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/playbook.yml ：

   - 将特定的 LVM 快照恢复到其原始卷：

```
---
- name: Revert a snapshot to its original volume
  hosts: all
  tasks:
    - name: Revert data_vg/data2
      vars:
        snapshot_lvm_snapset_name: snapset1
        snapshot_lvm_action: revert
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data2

  roles:
    - redhat.rhel_system_roles.snapshot
```

     在这里，snapshot_lvm_lv 参数描述了特定的逻辑卷(LV)，snapshot_lvm_vg 参数描述了特定的卷组(VG)。

   - 将一组 LVM 快照恢复到其原始卷中：

```
---
- name: Revert a set of snapshot
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
```

```
      lv: data1
    - name: data2 snapshot
      vg: data_vg
      lv: data2

tasks:
  - name: Revert the snapshot set
    vars:
      snapshot_lvm_action: revert
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"

roles:
  - redhat.rhel_system_roles.snapshot
```

此处，snapshot_lvm_set 参数描述了同一 VG 中的特定 LV。您还可以在设置此参数时指定来自不同 VG 的 LV。

注意

恢复操作 可能需要一些时间才能完成。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

4. 重启主机，或按照以下步骤取消激活并重新激活逻辑卷：

```
$ umount /data1; umount /data2

$ lvchange -an data_vg/data1 data_vg/data2

$ lvchange -ay data_vg/data1 data_vg/data2

$ mount /data1; mount /data2
```

其他资源

- /usr/share/ansible/roles/rhel-system-roles.snapshot/README.md file
- /usr/share/doc/rhel-system-roles/snapshot/ directory

10.8. 使用快照 RHEL 系统角色删除 LVM 快照

使用新的 快照 RHEL 系统角色，现在您可以通过指定快照的前缀或模式来删除所有 LVM 快照，并通过设置 snapshot_lvm_action 参数 来删除。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户具有 sudo 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 ~/playbook.yml ：

   - 删除特定的 LVM 快照：

```
---
- name: Remove a snapshot
  hosts: all
  tasks:
    - name: Remove snapshots in data_vg
      vars:
        snapshot_lvm_snapset_name: snapset1
        snapshot_lvm_action: remove
        snapshot_lvm_vg: data_vg

  roles:
    - redhat.rhel_system_roles.snapshot
```

     在这里，snapshot_lvm_vg 参数描述了卷组中的特定逻辑卷(LV)。

- 删除一组 LVM 快照：

```
---
- name: Remove a set of snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Remove a snapshot set
      vars:
        snapshot_lvm_action: remove
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"

  roles:
    - redhat.rhel_system_roles.snapshot
```

此处，snapshot_lvm_set 参数描述了同一 VG 中的特定 LV。您还可以在设置此参数时指定来自不同 VG 的 LV。

2. 验证 playbook 语法：

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

请注意，这个命令只验证语法，不会防止错误但有效的配置。

3. 运行 playbook:

```
$ ansible-playbook ~/playbook.yml
```

其他资源

- /usr/share/ansible/roles/rhel-system-roles.snapshot/README.md file

- /usr/share/doc/rhel-system-roles/snapshot/ directory

第 11 章 创建和管理精简配置的卷（精简卷）

Red Hat Enterprise Linux 支持精简配置的快照卷和逻辑卷。

逻辑卷和快照卷可以是精简配置的：

- 使用精简配置的逻辑卷，您可以创建大于可用物理存储的逻辑卷。
- 使用精简配置的快照卷，您可以在同一数据卷中存储更多虚拟设备。

11.1. 精简配置概述

很多现代存储堆栈现在提供在密集配置和精简配置之间进行选择的能力：

- 密集配置提供了块存储的传统行为，其中块的分配与其实际用途无关。
- 精简配置允许置备更大的块存储池，其大小可能大于存储数据的物理设备，从而导致过度配置。过度置备可能是因为单个块在实际使用之前没有被分配。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。

通过使用精简配置，您可以超额使用物理存储，且可以管理称为精简池的可用空间池。当应用程序需要时，您可以将这个精简池分配给任意数量的设备。当需要有效分配存储空间时，您可以动态扩展精简池。

例如，如果 10 个用户的每个用户都为他们的应用程序请求一个 100GB 的文件系统，那么您可以为每个用户创建一个 100GB 的文件系统，但其由较少的实际存储支持，仅在需要时使用。

注意

在使用精简配置时，监控存储池，并在可用物理空间耗尽时添加更多容量是非常重要的。

以下是使用精简配置的设备的一些优点：

- 您可以创建大于可用物理存储的逻辑卷。
- 您可以将更多的虚拟设备存储在相同的数据卷中。
- 您可以创建逻辑上可自动增长的文件系统，以支持数据需求，并将未使用的块返回给池，以供池中的任意文件系统使用

以下是使用精简配置的设备的潜在缺陷：

- 精简配置的卷存在耗尽可用物理存储的固有风险。如果过度配置了底层存储，您可能会因为缺少可用物理存储而导致停机。例如，如果您创建了 10T 的精简配置的存储，而只有 1T 的物理存储来支持，则卷将在 1T 耗尽后不可用或不可写。
- 如果卷在精简配置的设备后没有向层发送丢弃，那么对使用情况的统计将不准确。例如，在不使用 -o discard mount 选项的情况下放置文件系统，且不在精简配置的设备之上定期运行 fstrim，则永远不会不分配之前使用的存储。在这种情况下，随着时间的推移，即使没有真正使用它，您最终都会使用全部的置备量。
- 您必须监控逻辑和物理使用情况，以便不会用尽可用的物理空间。
- 在带有快照的文件系统上，写时复制(CoW)操作可能会较慢。
- 数据块可以在多个文件系统之间混合，导致底层存储的随机访问限制，即使它没有向最终用户展示那种方式。

11.2. 创建精简配置的逻辑卷

使用精简配置的逻辑卷，您可以创建大于可用物理存储的逻辑卷。创建精简配置的卷集允许系统分配您所使用的卷，而不是分配所请求的全部存储。

使用 lvcreate 命令的 -T 或 --thin 选项，您可以创建精简池或精简卷。您还可以使用 lvcreate 命令的 -T 选项，使用单个命令创建精简池和精简卷。这个流程描述了如何创建和增大精简配置的逻辑卷。

先决条件

- 您已创建了一个卷组。如需更多信息，请参阅创建 LVM 卷组。

流程

1. 创建精简池：

```
# lvcreate -L 100M -T vg001/mythinpool
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "mythinpool" created.
```

请注意：由于您要创建物理空间池，您必须指定池的大小。lvcreate 命令的 -T 选项不使用参数；它从随命令一起添加的其他选项来确定要创建的设备类型。您还可以使用额外的参数创建精简池，如下面的例子所示：

- 您还可以使用 lvcreate 命令的 --thinpool 参数创建精简池。与 -T 选项不同，--thinpool 参数要求您指定您要创建的精简池逻辑卷的名称。以下示例使用 --thinpool 参数在卷组 *vg001* 中创建名为 *mythinpool* 的精简池，大小为 *100M*：

```
# lvcreate -L 100M --thinpool mythinpool vg001
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "mythinpool" created.
```

- 由于池创建支持条带，所以您可以使用 -i 和 -I 选项来创建条带。以下命令在卷组 *vg001* 中创建一个名为 *thinpool* 的 *100M* 的精简池，其有两个 *64 kB* 条带和一个大小为 *256 kB* 的块。它还会创建一个名为 *vg001/thinvolume* 的 *1T* 的精简卷。

注意

确保卷组中有两个有足够空闲空间的物理卷，否则您无法创建精简池。

```
# lvcreate -i 2 -I 64 -c 256 -L 100M -T vg001/thinpool -V 1T --name thinvolume
```

2. 创建精简卷：

```
# lvcreate -V 1G -T vg001/mythinpool -n thinvolume
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool vg001/mythinpool (100.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of space.
```

```
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.
  Logical volume "thinvolume" created.
```

在这种情况下，您为卷指定了虚拟大小，其大于包含它的池。您还可以使用额外的参数创建精简卷，如下面的例子所示：

- 要创建精简卷和精简池，请使用 lvcreate 命令的 -T 选项，并指定大小和虚拟大小参数：

```
# lvcreate -L 100M -T vg001/mythinpool -V 1G -n thinvolume
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of
data.
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool
vg001/mythinpool (100.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of
space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.
  Logical volume "thinvolume" created.
```

- 要使用剩余的空闲空间来创建精简卷和精简池，请使用 100%FREE 选项：

```
# lvcreate -V 1G -l 100%FREE -T vg001/mythinpool -n thinvolume
Thin pool volume with chunk size 64.00 KiB can address at most <15.88 TiB of
data.
  Logical volume "thinvolume" created.
```

- 要将现有的逻辑卷转换成精简池卷，请使用 lvconvert 命令的 --thinpool 参数。您还必须将 --poolmetadata 参数与--thinpool 参数结合使用，将现有的逻辑卷转换成精简池卷的元数据卷。

以下示例将卷组 *vg001* 中的现有逻辑卷 *lv1* 转换为精简池卷，并将卷组 *vg001* 中的现有逻辑卷 *lv2* 转换为那个精简池卷的元数据卷：

```
# lvconvert --thinpool vg001/lv1 --poolmetadata vg001/lv2
  Converted vg001/lv1 to thin pool.
```

注意

将逻辑卷转换成精简池卷或者精简池元数据卷会破坏逻辑卷的内容，因为 lvconvert 不会保留设备的内容，而是覆盖其内容。

- 默认情况下，lvcreate 命令使用以下公式设置精简池元数据逻辑卷的大致大小：

```
Pool_LV_size / Pool_LV_chunk_size * 64
```

如果您有大量快照，或者您的精简池有小的块，且预期以后精简池的大小会显著增长，则您可能需要使用 lvcreate 命令的 --poolmetadatasize 参数来增加精简池的元数据卷的默认值。精简池元数据逻辑卷所支持的值在 2MiB 到 16GiB 之间。

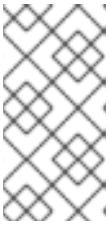
以下示例演示了如何增大精简池的元数据卷的默认值：

```
# lvcreate -V 1G -l 100%FREE -T vg001/mythinpool --poolmetadatasize 16M -n
thinvolume
Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "thinvolume" created.
```

3. 查看创建的精简池和精简卷：

```
# lvs -a -o +devices
  LV              VG    Attr       LSize   Pool       Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  [lvol0_pmspare]    vg001 ewi-------   4.00m
/dev/sda(0)
  mythinpool         vg001 twi-aotz-- 100.00m                 0.00   10.94
mythinpool_tdata(0)
  [mythinpool_tdata] vg001 Twi-ao---- 100.00m
/dev/sda(1)
  [mythinpool_tmeta] vg001 ewi-ao----   4.00m
/dev/sda(26)
  thinvolume         vg001 Vwi-a-tz--   1.00g mythinpool        0.00
```

4. 可选：使用 lvextend 命令扩展精简池的大小。但是您无法缩小精简池的大小。



注意

在创建精简池和精简卷的过程中，如果您使用 -l 100%FREE 参数，这个命令会失败。

以下命令调整了一个已存在的大小为 *100M* 的精简池，将其大小增加 *100M*。

```
# lvextend -L+100M vg001/mythinpool
```

```
  Size of logical volume vg001/mythinpool_tdata changed from 100.00 MiB (25 extents)
to 200.00 MiB (50 extents).
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool
vg001/mythinpool (200.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of
space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.

  Logical volume vg001/mythinpool successfully resized
```

```
# lvs -a -o +devices
  LV              VG    Attr      LSize   Pool       Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  [lvol0_pmspare]    vg001 ewi-------   4.00m
/dev/sda(0)
  mythinpool         vg001 twi-aotz-- 200.00m                 0.00   10.94
mythinpool_tdata(0)
  [mythinpool_tdata] vg001 Twi-ao---- 200.00m
/dev/sda(1)
  [mythinpool_tdata] vg001 Twi-ao---- 200.00m
/dev/sda(27)
  [mythinpool_tmeta] vg001 ewi-ao----   4.00m
/dev/sda(26)
  thinvolume         vg001 Vwi-a-tz--   1.00g mythinpool        0.00
```

5. 可选：要重命名精简池和精简卷，请使用以下命令：

```
# lvrename vg001/mythinpool vg001/mythinpool1
  Renamed "mythinpool" to "mythinpool1" in volume group "vg001"

# lvrename vg001/thinvolume vg001/thinvolume1
  Renamed "thinvolume" to "thinvolume1" in volume group "vg001"
```

重命名后查看精简池和精简卷：

```
# lvs
  LV          VG      Attr     LSize   Pool      Origin Data%  Move Log Copy%  Convert
mythinpool1 vg001   twi-a-tz 100.00m                 0.00
thinvolume1 vg001   Vwi-a-tz   1.00g mythinpool1        0.00
```

6. 可选：要删除精简池，请使用以下命令：

```
# lvremove -f vg001/mythinpool1
  Logical volume "thinvolume1" successfully removed.
  Logical volume "mythinpool1" successfully removed.
```

其他资源

- lvcreate(8)、lvrename(8)、lvs(8) 和 lvconvert(8) 手册页

11.3. 块大小概述

块是专用于快照存储的物理磁盘的最大单位。

使用以下条件来使用块大小：

- 较小的块需要更多元数据并会影响到性能，但可以更好地利用快照。
- 较大的块需要较少的元数据操作，但快照的空间利用率较小。

默认情况下，lvm2 以 64KiB 块大小开头，并为此块大小估算良好的元数据大小。lvm2 可以创建和使用的最小元数据大小是 2 MiB。如果元数据大小需要大于 128 MiB，它开始增大块大小，因此元数据大小会保持紧凑。然而，这可能会导致一些大的块值，这使快照使用效率较低。在这种情况下，一个小些的块大小和大些的元数据大小是一个较好的选择。

要根据您的需要指定块大小，请使用 -c 或 --chunksize 参数来否决 lvm2 估计的块大小。请注意，创建 thinpool 后您无法更改块大小。

如果卷数据的大小在 TiB 范围，使用 ~15.8GiB 作为元数据大小（这是最大支持的大小），并根据您的要求设置块大小。但请注意，如果您需要扩展卷的数据大小且具有小的块，则无法增大元数据大小。

注意

当用户耗尽了 元数据 中的空间，或者由于有限的最大可寻址的精简池大小，使用块大小和元数据的不合适的组合可能会导致出现潜在的问题。

其他资源

- lvmthin (7) 手册页

11.4. 精简配置的快照卷

Red Hat Enterprise Linux 支持精简配置的快照卷。精简逻辑卷的快照也会创建一个精简逻辑卷(LV)。精简快照卷具有与其它精简卷相同的特征。您可以独立激活卷、扩展卷、重新命名卷、删除卷、甚至快照卷。

注意

与所有 LVM 快照卷以及所有精简卷类似，集群中的节点不支持精简快照卷。快照卷必须在一个集群节点中完全激活。

传统快照必须为创建的每一个快照分配新的空间，其中数据作为对源的修改而被保留。但精简配置的快照与源共享相同的空间。精简 LV 的快照非常有效，因为精简 LV 及它的任何快照的公共数据块是共享的。您可以创建精简 LV 的快照或从其他精简快照创建快照。递归快照的通用块也在精简池中共享。

精简快照卷提供以下优点：

- 增加源的快照数量对性能的影响可以忽略不计。
- 精简快照卷可以减少磁盘用量，因为只有新数据被写入，且不会复制到每个快照中。
- 不需要同时激活精简快照卷和源，这是传统快照的要求。
- 当从快照恢复源时，不需要合并精简快照。您可以删除源，而使用快照。传统的快照有单独的卷，其中存储必须要复制回来的更改，即：合并到源以重置它。
- 与传统的快照相比，对允许的快照数量有很大的限制。

虽然使用精简快照卷有很多好处，但在有些情况下，传统的 LVM 快照卷功能可能更适合您的需要。您可以对所有类型的卷使用传统快照。但是，要使用精简快照则要求您使用精简配置。

注意

您不能限制精简快照卷的大小；如有必要，快照将使用精简池中的所有空间。一般说来，在决定使用什么快照格式时，您应该考虑具体的要求。

默认情况下，在通常的激活命令过程中会跳过精简快照卷。

11.5. 创建精简配置的快照卷

使用精简配置的快照卷，可以在同一数据卷上存储更多的虚拟设备。

重要

在创建精简快照卷时，不要指定卷的大小。如果指定了 size 参数，则创建的快照不会是一个精简快照卷，也不会使用精简池来存储数据。例如： lvcreate -s vg/thinvolume -L10M 命令将不会创建精简快照，即使原始卷是一个精简卷。

可为精简配置的原始卷创建精简快照，也可针对不是精简置备的原始卷创建精简快照。以下流程描述了创建精简配置的快照卷的不同方法。

先决条件

- 您已创建了一个精简配置的逻辑卷。如需更多信息，请参阅精简置备概述。

流程

- 创建一个精简配置的快照卷。下面的命令在精简配置的逻辑卷 *vg001/thinvolume* 上创建一个名为 *mysnapshot1* 的精简配置的快照卷：

```
# lvcreate -s --name mysnapshot1 vg001/thinvolume
  Logical volume "mysnapshot1" created
```

```
# lvs
  LV          VG      Attr     LSize   Pool       Origin     Data%  Move Log Copy%  Convert
  mysnapshot1 vg001   Vwi-a-tz   1.00g mythinpool thinvolume   0.00
  mythinpool  vg001   twi-a-tz 100.00m                        0.00
  thinvolume  vg001   Vwi-a-tz   1.00g mythinpool             0.00
```

注意

在使用精简配置时，存储管理员务必要监控存储池，并在其被完全占用时添加更多容量。有关扩展精简卷大小的详情，请参考 创建精简配置的逻辑卷。

- 您还可以为非置备的逻辑卷创建精简配置的快照。因为非精简配置的逻辑卷不包含在精简池中，因此它也被称为外部源。外部原始卷可以被很多精简置备的快照卷使用和共享，即使在不同的精简池中也是如此。在创建精简置备快照时，外部原始源必须不活跃且只读。

  以下示例创建一个名为 *origin_volume* 的、只读的、不活动的逻辑卷的精简快照卷。精简快照卷名为 *mythinsnap*。然后，逻辑卷 *origin_volume* 在卷组 *vg001* 中成为精简快照卷 *mythinsnap* 的精简外部源，其使用现有的精简池 *vg001/pool*。原始卷必须与快照卷位于同一个卷组中。在指定原始逻辑卷时不要指定卷组。

```
# lvcreate -s --thinpool vg001/pool origin_volume --name mythinsnap
```

- 您可以通过执行以下命令，创建第一个快照卷的第二个精简配置的快照卷。

```
# lvcreate -s vg001/mysnapshot1 --name mysnapshot2
Logical volume "mysnapshot2" created.
```

  要创建第三个精简配置的快照卷，请使用以下命令：

```
# lvcreate -s vg001/mysnapshot2 --name mysnapshot3
Logical volume "mysnapshot3" created.
```

验证

- 显示精简快照逻辑卷的所有祖先和后代的列表：

```
$ lvs -o name,lv_ancestors,lv_descendants vg001
  LV          Ancestors                       Descendants
  mysnapshot2 mysnapshot1,thinvolume          mysnapshot3
  mysnapshot1 thinvolume                      mysnapshot2,mysnapshot3
  mysnapshot3 mysnapshot2,mysnapshot1,thinvolume
  mythinpool
  thinvolume                                  mysnapshot1,mysnapshot2,mysnapshot3
```

  在这里，

- *thinvolume* 是卷组 *vg001* 中的一个原始卷。

- *mysnapshot1* 是 *thinvolume* 的一个快照

- *mysnapshot2* 是 *mysnapshot1* 的一个快照

- *mysnapshot3* 是 *mysnapshot2* 的一个快照

注意

lv_ancestors 和 lv_descendants 字段显示现有的依赖关系。但是，如果从链中删除了条目，则它们不会跟踪这些条目，因为这些条目可能会破坏依赖关系链。

其他资源

- lvcreate(8) 手册页

11.6. 历史逻辑卷

现在，您可以通过在 lvm.conf 配置文件中启用 record_lvs_history 元数据选项将系统配置为跟踪已删除的精简快照和精简逻辑卷。这可让您显示完整的精简快照依赖链，其中包括已经从原始依赖链中删除的逻辑卷和已变为 *historical* 的逻辑卷。

您可以通过在 lvm.conf 配置文件中使用 lvs_history_retention_time 元数据选项指定保留时间（以秒为单位），可以将您的系统配置为保留历史卷。

历史逻辑卷保留了已删除的逻辑卷的简化表示，其中包括以下卷报告字段：

- lv_time_removed ：逻辑卷的删除时间

- lv_time ：逻辑卷的创建时间

- lv_name ：逻辑卷的名称

- lv_uuid ：逻辑卷的 UUID

- VG_ NAME ：包含逻辑卷的卷组。

重要

无法重新激活历史逻辑卷。

当您删除卷时，历史逻辑卷名称会有一个连字符作为前缀。例如，如果您删除了逻辑卷 lvol1，历史卷的名称将变为 -lvol1。

如果卷没有实时 descendant.，则逻辑卷管理器 (LVM) 不会保留历史逻辑卷。就是说，如果您在快照链末尾删除逻辑卷，则不会保留逻辑卷作为历史逻辑卷。

要在卷显示中包含历史逻辑卷，请指定 LVM display 命令的 -H|--history 选项。您可以通过指定 lv_full_ancestors 和 lv_full_descendants 报告字段以及 -H 选项来显示包括历史卷的整个精简快照依赖关系链。

11.7. 跟踪并显示已删除的精简快照卷

这个步骤描述了如何显示和管理已删除的历史逻辑卷。

流程

1. 通过在 lvm.conf 文件中设置 record_lvs_history=1 来确保保留了历史逻辑卷。默认不启用这个元数据选项。

2. 可选：以秒为单位设置 lvs_history_retention_time 选项的值。这是记录自动销毁历史逻辑卷的时间间隔。自动默认值为 0，它会禁用此功能。如果没有设置这个选项，也可以手动删除单独的历史卷。

3. 显示精简调配的快照链：

   在本例中：

   - lvol1 是一个原始卷，是链中的第一个卷。

- lvol2 是 lvol1 的快照。
- lvol3 是 lvol2 的快照。
- lvol4 是 lvol3 的快照。
- lvol5 也是 lvol3 的快照。

```
# lvs -H -o name,full_ancestors,full_descendants
  LV    FAncestors       FDescendants
  lvol1                  lvol2,lvol3,lvol4,lvol5
  lvol2 lvol1            lvol3,lvol4,lvol5
  lvol3 lvol2,lvol1      lvol4,lvol5
  lvol4 lvol3,lvol2,lvol1
  lvol5 lvol3,lvol2,lvol1
  pool
```

请注意，虽然使用带有 -H 选项的 lvs 工具，但不会删除精简快照卷，且没有要显示的历史逻辑卷。

4. 从快照链中删除逻辑卷 lvol3 ：

```
# lvremove -f vg/lvol3
  Logical volume "lvol3" successfully removed
```

5. 运行 lvs 工具来查看历史逻辑卷的详情，以及它们的 ancestors 和 descendants：

```
# lvs -H -o name,full_ancestors,full_descendants
  LV     FAncestors        FDescendants
  lvol1                    lvol2,-lvol3,lvol4,lvol5
  lvol2  lvol1             -lvol3,lvol4,lvol5
  -lvol3 lvol2,lvol1       lvol4,lvol5
  lvol4  -lvol3,lvol2,lvol1
  lvol5  -lvol3,lvol2,lvol1
  pool
```

6. 另外，显示历史卷的删除时间戳：

```
# lvs -H -o name,full_ancestors,full_descendants,time_removed
```

```
  LV     FAncestors         FDescendants              RTime
  lvol1                     lvol2,-lvol3,lvol4,lvol5
  lvol2  lvol1               -lvol3,lvol4,lvol5
  -lvol3 lvol2,lvol1         lvol4,lvol5               2016-03-14 14:14:32 +0100
  lvol4  -lvol3,lvol2,lvol1
  lvol5  -lvol3,lvol2,lvol1
  pool
```

7. 您可以通过指定 *vgname/lvname* 格式在 display 命令中单独引用历史逻辑卷：

```
# lvs -H vg/-lvol3
  LV     VG   Attr       LSize
  -lvol3 vg   ----h-----    0
```

请注意，lv_attr 字段中的第五个位被设置为 h，表示该卷是历史的。

8. 如果卷没有实时子卷（后代），LVM 不会保存历史逻辑卷。就是说，如果您在快照链末尾删除逻辑卷，则不会保留逻辑卷作为历史逻辑卷。

```
# lvremove -f vg/lvol5
  Automatically removing historical logical volume vg/-lvol5.
  Logical volume "lvol5" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
  LV     FAncestors         FDescendants
  lvol1                     lvol2,-lvol3,lvol4
  lvol2  lvol1               -lvol3,lvol4
  -lvol3 lvol2,lvol1         lvol4
  lvol4  -lvol3,lvol2,lvol1
  pool
```

9. 删除卷 lvol1 和 lvol2，并查看 lvs 命令在卷被删除后如何显示它们。

```
# lvremove -f vg/lvol1 vg/lvol2
  Logical volume "lvol1" successfully removed
  Logical volume "lvol2" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
  LV     FAncestors          FDescendants
  -lvol1                     -lvol2,-lvol3,lvol4
  -lvol2 -lvol1               -lvol3,lvol4
  -lvol3 -lvol2,-lvol1        lvol4
  lvol4  -lvol3,-lvol2,-lvol1
  pool
```

10. 通过指定包含连字符的历史卷名称来完全删除历史逻辑卷，如下例所示

```
# lvremove -f vg/-lvol3
Historical logical volume "lvol3" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
  LV     FAncestors    FDescendants
  -lvol1               -lvol2,lvol4
  -lvol2 -lvol1        lvol4
  lvol4  -lvol2,-lvol1
  pool
```

第 12 章 启用缓存来提高逻辑卷性能

您可以在 LVM 逻辑卷中添加缓存以提高性能。LVM 然后使用快速设备（如 SSD）将 I/O 操作缓存到逻辑卷中。

下面的过程会从快速设备创建一个特殊的 LV，并将这个特殊 LV 附加到原始 LV，以便提高性能。

12.1. LVM 中的缓存方法

LVM 提供以下缓存类型。每种模式适合逻辑卷中的不同类型的 I/O 模式。

dm-cache

这个方法可通过在更快的卷上缓存数据来加快频繁使用数据的访问速度。这个方法会缓存读写操作。

dm-cache 方法创建类型为 缓存 的逻辑卷。

dm-writecache

这个方法只缓存写操作。使用快速卷进行写入操作，然后将其迁移到后台较慢的磁盘中。快速卷通常是一个 SSD 或持久内存（PMEM）磁盘。

dm-writecache 方法创建类型为 writecache 的逻辑卷。

其他资源

- lvmcache (7) 手册页

12.2. LVM 缓存组件

LVM 支持向 LVM 逻辑卷添加缓存的支持。LVM 缓存使用以下 LVM 逻辑卷类型：

主 LV

较大的、较慢和原始卷。

缓存池 LV

可用于缓存主 LV 数据的复合 LV。它有两个子 LV ：用于保存缓存数据和元数据以管理缓存数据的数据。您可以为数据和元数据配置特定的磁盘。您只能使用具有 dm-cache 的缓存池。

Cachevol LV

可用于缓存主 LV 数据的线性 LV。您无法为数据和元数据配置单独的磁盘。cachevol 只能与 dm-cache 或 dm-writecache 一起使用。

所有这些关联的 LV 必须位于同一卷组中。

您可以将主逻辑卷(LV)与更快速、通常是较小的保存缓存数据的 LV 相结合。快速 LV 是从快速块设备（如 SSD 驱动器）创建的。当您为逻辑卷启用缓存时，LVM 会重新命名并隐藏原始卷，并显示由原始逻辑卷组成的新逻辑卷。新逻辑卷的组成取决于缓存方法以及您是否使用 cachevol 或 cachepool 选项。

cachevol 和 cachepool 选项会公开对缓存组件的放置不同级别的控制：

- 使用 cachevol 选项，快速设备会同时存储数据块的缓存副本以及用于管理缓存的元数据。
- 使用 cachepool 选项，单独的设备可以存储数据块的缓存副本以及用于管理缓存的元数据。

  dm-writecache 方法与 cachepool 不兼容。

在所有配置中，LVM 会公开一个生成的设备，它会将所有缓存组件组合在一起。得到的设备的名称与原来的较慢的逻辑卷的名称相同。

其他资源

- lvmcache (7) 手册页
- 创建和管理精简配置的卷（精简卷）

12.3. 为逻辑卷启用 DM-CACHE 缓存

这个过程使用 dm-cache 方法启用逻辑卷中常用数据缓存。

先决条件

- 您希望使用 dm-cache 加快的逻辑卷存在于系统中。
- 包含较慢逻辑卷的卷组还包含在快速块设备中未使用的物理卷。

流程

1. 在快速设备中创建 cachevol 卷：

```
# lvcreate --size cachevol-size --name <fastvol> <vg> </dev/fast-pv>
```

替换以下值：

cachevol-size

cachevol 卷的大小，如 5G

fastvol

cachevol 卷的名称

vg

卷组名称

/dev/fast-pv

快速块设备的路径，如 /dev/sdf

例 12.1. 创建一个 cachevol 卷

```
# lvcreate --size 5G --name fastvol vg /dev/sdf
Logical volume "fastvol" created.
```

2. 将 cachevol 卷附加到主逻辑卷以开始缓存：

```
# lvconvert --type cache --cachevol <fastvol> <vg/main-lv>
```

替换以下值：

fastvol

cachevol 卷的名称

vg

卷组名称

main-lv

较慢的逻辑卷名称

例 12.2. 将 cachevol 卷附加到主 LV

```
# lvconvert --type cache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]: y
Logical volume vg/main-lv is now cached.
```

验证步骤

- 验证新创建的逻辑卷是否启用了 dm-cache ：

```
# lvs --all --options +devices <vg>

LV              Pool          Type   Devices
main-lv         [fastvol_cvol] cache  main-lv_corig(0)
[fastvol_cvol]                linear /dev/fast-pv
[main-lv_corig]               linear /dev/slow-pv
```

其他资源

- lvmcache (7) 手册页

12.4. 使用 CACHEPOOL 为逻辑卷启用 DM-CACHE 缓存

这个过程可让您单独创建缓存数据和缓存元数据逻辑卷，然后将卷合并到缓存池中。

先决条件

先决条件

- 您希望使用 dm-cache 加快的逻辑卷存在于系统中。
- 包含较慢逻辑卷的卷组还包含在快速块设备中未使用的物理卷。

流程

1. 在快速设备中创建 cachepool 卷：

```
# lvcreate --type cache-pool --size <cachepool-size> --name <fastpool> <vg /dev/fast>
```

替换以下值：

cachepool-size

cachepool 的大小，如 5G

fastpool

cachepool 卷的名称

vg

卷组名称

/dev/fast

到快速块设备的路径，如 /dev/sdf1

注意

您可以使用 **--poolmetadata** 选项指定创建 **cache-pool** 时池元数据的位置。

例 12.3. 创建一个 **cachepool** 卷

```
# lvcreate --type cache-pool --size 5G --name fastpool vg /dev/sde
Logical volume "fastpool" created.
```

2. 将 cachepool 附加到主逻辑卷中开始缓存：

```
# lvconvert --type cache --cachepool <fastpool> <vg/main>
```

替换以下值：

fastpool

cachepool 卷的名称

vg

卷组名称

main

较慢的逻辑卷名称

例 12.4. 将 cachepool 附加到主 LV

```
# lvconvert --type cache --cachepool fastpool vg/main
Do you want wipe existing metadata of cache pool vg/fastpool? [y/n]: y
Logical volume vg/main is now cached.
```

验证步骤

- 检查新创建的 cache-pool 类型的 devicevolume：

```
# lvs --all --options +devices <vg>

LV                 Pool              Type       Devices
[fastpool_cpool]                     cache-pool  fastpool_pool_cdata(0)
[fastpool_cpool_cdata]               linear      /dev/sdf1(4)
[fastpool_cpool_cmeta]               linear      /dev/sdf1(2)
[lvol0_pmspare]                      linear      /dev/sdf1(0)
main               [fastpoool_cpool] cache       main_corig(0)
[main_corig]                         linear      /dev/sdf1(O)
```

其他资源

- lvcreate(8) 手册页

- **lvmcache (7) 手册页**
- **lvconvert(8) 手册页**

12.5. 为逻辑卷启用 DM-WRITECACHE 缓存

这个过程允许使用 dm-writecache 方法将 I/O 操作缓存到逻辑卷中。

先决条件

- 您希望使用 dm-writecache 加快的逻辑卷存在于系统中。
- 包含较慢逻辑卷的卷组还包含在快速块设备中未使用的物理卷。
- 如果较慢的逻辑卷处于活跃状态，请停用它。

流程

1. 如果一个较慢的逻辑卷是活跃的，取消激活它：

```
# lvchange --activate n <vg>/<main-lv>
```

替换以下值：

vg

卷组名称

main-lv

较慢的逻辑卷名称

2. 在快速设备中创建一个已停用的 cachevol 卷：

```
# lvcreate --activate n --size <cachevol-size> --name <fastvol> <vg> </dev/fast-pv>
```

替换以下值：

cachevol-size

cachevol 卷的大小，如 5G

fastvol

cachevol 卷的名称

vg

卷组名称

/dev/fast-pv

快速块设备的路径，如 /dev/sdf

例 12.5. 创建一个停用的 cachevol 卷

```
# lvcreate --activate n --size 5G --name fastvol vg /dev/sdf
WARNING: Logical volume vg/fastvol not zeroed.
Logical volume "fastvol" created.
```

3. 将 cachevol 卷附加到主逻辑卷以开始缓存：

```
# lvconvert --type writecache --cachevol <fastvol> <vg/main-lv>
```

替换以下值：

fastvol

cachevol 卷的名称

vg

卷组名称

main-lv

较慢的逻辑卷名称

例 12.6. 将 cachevol 卷附加到主 LV

```
# lvconvert --type writecache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]?: y
Using writecache block size 4096 for unknown file system block size, logical block
size 512, physical block size 512.
WARNING: unable to detect a file system block size on vg/main-lv
WARNING: using a writecache block size larger than the file system block size may
corrupt the file system.
Use writecache block size 4096? [y/n]: y
Logical volume vg/main-lv now has writecache.
```

4. **激活生成的逻辑卷：**

```
# lvchange --activate y <vg/main-lv>
```

替换以下值：

vg

卷组名称

main-lv

较慢的逻辑卷名称

验证步骤

- **检查新创建的设备：**

```
# lvs --all --options +devices vg

LV              VG Attr      LSize   Pool          Origin          Data%  Meta%  Move Log
Cpy%Sync Convert Devices
 main-lv        vg Cwi-a-C--- 500.00m [fastvol_cvol] [main-lv_wcorig] 0.00
main-lv_wcorig(0)
 [fastvol_cvol]   vg Cwi-aoC--- 252.00m
/dev/sdc1(0)
 [main-lv_wcorig] vg owi-aoC--- 500.00m
/dev/sdb1(0)
```

其他资源

- **lvmcache (7) 手册页**

12.6. 为逻辑卷禁用缓存

这个过程禁用 **dm-cache 或 dm-writecache** 缓存，该缓存在逻辑卷中当前启用。

先决条件

- **在逻辑卷中启用了缓存。**

流程

1. **取消激活逻辑卷：**

```
# lvchange --activate n <vg>/<main-lv>
```

使用卷组名称替换 *vg*，使用启用了缓存的逻辑卷的名称替换 *main-lv*。

2. **分离 cachevol 或 cachepool 卷：**

 # lvconvert --splitcache *<vg>*/*<main-lv>*

替换以下值：

使用卷组名称替换 *vg*，使用启用了缓存的逻辑卷的名称替换 *main-lv*。

例 12.7. 分离 cachevol 或 cachepool 卷

```
# lvconvert --splitcache vg/main-lv
Detaching writecache already clean.
Logical volume vg/main-lv writecache has been detached.
```

验证步骤

- 检查逻辑卷不再连接在一起：

```
# lvs --all --options +devices <vg>

LV      Attr       Type   Devices
fastvol -wi------- linear /dev/fast-pv
main-lv -wi------- linear /dev/slow-pv
```

其他资源

- lvmcache (7) 手册页

第 13 章 逻辑卷激活

默认情况下，当您创建逻辑卷时，它处于活动状态。处于活跃状态的逻辑卷可以通过块设备使用。激活的逻辑卷可以被访问，并可能会发生变化。

有些情况下，您需要使单个逻辑卷不活跃，因此对内核未知。您可以使用 lvchange 命令的 -a 选项激活或停用单独的逻辑卷。

以下是停用单个逻辑卷的格式：

```
# lvchange -an vg/lv
```

以下是激活单个逻辑卷的格式：

```
# lvchange -ay vg/lv
```

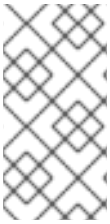
您可以使用 vgchange 命令的 -a 选项激活或停用卷组中的所有逻辑卷。这等同于在卷组的每个独立逻辑卷中运行 lvchange -a 命令。

以下是停用卷组中所有逻辑卷的格式：

```
# vgchange -an vg
```

以下是激活卷组中所有逻辑卷的格式：

```
# vgchange -ay vg
```



注意

在手动激活过程中，systemd 会使用 /etc/fstab 文件中相应的挂载点自动挂载 LVM 卷，除非 systemd-mount 单元被屏蔽。

13.1. 控制逻辑卷和卷组的自动激活

自动激活逻辑卷指的是，在系统启动时基于事件自动激活逻辑卷。

您可以在 VG 或 LV 上设置自动激活属性。当禁用了自动激活属性时，VG 或 LV 将不会通过自动激活命令被激活，如 vgchange、lvchange，或使用 -aay 选项的 pvscan。如果在 VG 上禁用了自动激活，则不会在该 VG 中自动激活 LV，自动激活没有效果。如果在 VG 上启用了自动激活，则可以为单个 LV 禁用自动激活。

流程

- 您可以使用以下方法之一更新自动激活设置：
  - 使用命令行控制 VG 的自动激活：

```
# vgchange --setautoactivation <y|n>
```

  - 使用命令行控制 LV 的自动激活：

```
# lvchange --setautoactivation <y|n>
```

  - 您可以使用 /etc/lvm/lvm.conf 配置文件中的 activation/auto_activation_volume_list 配置选项来控制特定 LV 和 VG 的自动激活：

    auto_activation_volume_list = ["*<VG_name>*", "*<VG_name>*/*<LV_name>*", "*<@tag1>*", "*<...>*"]

    如果将 auto_activation_volume_list 设置为 [] （空列表），则自动激活被完全禁用。

其他资源

- /etc/lvm/lvm.conf 配置文件
- lvmautoactivation (7) 手册页

13.2. 控制逻辑卷激活

您可以使用以下方法控制逻辑卷的激活：

- 通过 /etc/lvm/conf 文件中的 activation/volume_list 设置。这可让您指定激活哪些逻辑卷。有关使用这个选项的详情，请查看 /etc/lvm/lvm.conf 配置文件。

- 逻辑卷的激活跳过标签。当为逻辑卷设定这个标签时，会在正常的激活命令中跳过该卷。

或者，您可以将 --setactivationskip y|n 选项与 lvcreate 或 lvchange 命令一起使用，以启用或禁用激活跳过标志。

流程

- 您可以使用以下方法在逻辑卷上设置激活跳过标签：

  - 要确定是否为逻辑卷设置了激活跳过标签，请运行 lvs 命令，该命令显示 k 属性，如下例所示：

```
# lvs vg/thin1s1
  LV         VG  Attr       LSize Pool  Origin
  thin1s1    vg  Vwi---tz-k 1.00t pool0 thin1
```

    除了标准的 -ay 或 --activate y 选项外，您还可以使用 -K 或 --ignoreactivationskip 选项来激活具有 k 属性的逻辑卷。

    默认情况下，精简快照卷在创建时将其标记为激活跳过。您可以使用 /etc/lvm/lvm.conf 文件中的 auto_set_activation_skip 设置控制新精简快照卷的默认激活跳过设置。

  - 下面的命令激活设置了激活跳过标签的精简快照逻辑卷：

```
# lvchange -ay -K VG/SnapLV
```

  - 以下命令创建没有激活跳过标签的精简快照：

```
# lvcreate -n SnapLV -kn -s vg/ThinLV --thinpool vg/ThinPoolLV
```

  - 以下命令从快照逻辑卷中删除激活跳过标签：

```
# lvchange -kn VG/SnapLV
```

验证步骤

- **验证是否创建了没有激活跳过标签的精简快照：**

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize    Pool       Origin Data%  Meta%  Move Log
Cpy%Sync Convert Devices             Type
  SnapLV          vg          Vwi-a-tz--  100.00m ThinPoolLV ThinLV 0.00
thin
  ThinLV          vg          Vwi-a-tz--  100.00m ThinPoolLV        0.00
thin
  ThinPoolLV      vg          twi-aotz--  100.00m                   0.00   10.94
ThinPoolLV_tdata(0) thin-pool
  [ThinPoolLV_tdata] vg        Twi-ao----  100.00m
/dev/sdc1(1)       linear
  [ThinPoolLV_tmeta] vg        ewi-ao----    4.00m
/dev/sdd1(0)       linear
  [lvol0_pmspare]    vg        ewi-------    4.00m
/dev/sdc1(0)       linear
```

13.3. 激活共享逻辑卷

您可以使用 lvchange 和 vgchange 命令的 -a 选项控制共享逻辑卷的逻辑卷激活，如下所示：

命令	激活
lvchange -ay|-aey	以相互排斥的模式激活共享逻辑卷，只允许一个主机激活逻辑卷。如果激活失败，如逻辑卷在另外一个主机上激活一样，会报告一个错误。
lvchange -asy	以共享模式激活共享逻辑卷，允许多个主机同时激活逻辑卷。如果激活失败，如逻辑卷只在另一个主机中激活时一样，会出错。如果逻辑类型禁止共享访问，比如快照，命令将报告错误并失败。无法从多个主机同时使用的逻辑卷类型包括 thin、cache、raid 和 snapshot。
lvchange -an	取消激活逻辑卷。

13.4. 在缺少设备的情况下激活逻辑卷

您可以控制缺少设备的 LV 是否可以使用带有 --activationmode partial|degraded|complete 选项的 lvchange 命令激活。这些值如下所述：

激活模式	含义
complete	只允许激活没有缺失物理卷的逻辑卷。这是限制性最强的模式。
degraded	允许激活含有缺失物理卷的 RAID 逻辑卷。
partial	允许激活任何含有缺失物理卷的逻辑卷。这个选项只应用于恢复或修复。

activationmode 的默认值由 /etc/lvm/lvm.conf 文件中的 activationmode 设置决定。如果未给出命令行选项，则会使用它。

其他资源

- **lvmraid (7) 手册**页

第 14 章 限制 LVM 设备可见性和用法

您可以通过控制 LVM 可扫描的设备来限制逻辑卷管理器(LVM)可用的设备。

使用 LVM 命令控制 LVM 设备扫描。LVM 命令与名为 system.devices 文件的文件交互，该文件列出了可见和可用的设备。在 Red Hat Enterprise Linux 9 中默认启用这个功能。

如果您禁用设备文件功能，则 LVM 设备过滤器会自动启用。

要调整 LVM 设备扫描的配置，请编辑 /etc/lvm/lvm.conf 文件中的 LVM 设备过滤器设置。lvm.conf 文件中的过滤器由一系列简单的正则表达式组成。系统会将这些表达式应用于 /dev 目录中的每个设备名称，以确定是否接受或拒绝每个检测到的块设备。

14.1. LVM 设备文件

Logical Volume Manager (LVM) system.devices 文件控制 LVM 设备可见性和可用性。您可以在 /etc/lvm/devices/ 目录中找到设备文件。使用 LVM 命令管理设备文件。不要直接编辑 system.devices 文件。

默认情况下，在 Red Hat Enterprise Linux 9 中启用了 system.devices 文件功能。当激活时，它会替换 LVM 设备过滤器。要启用 LVM 设备过滤器，请禁用 system.devices 文件。如需更多信息，请参阅禁用 system.devices 文件。

14.1.1. 其他资源

- lvmdevices (8) 和 lvm.conf (5) 手册页

14.1.2. 在 system.devices 文件中添加设备

要将设备与逻辑卷管理器(LVM)一起使用，system.devices 文件必须包含设备 ID 列表，否则 LVM 会忽略它们。操作系统 (OS) 安装程序在安装过程中将设备添加到 system.devices 文件中。新安装的系统会自动将 root 设备自动包含在设备文件中。在操作系统安装期间附加到系统的任何物理卷 (PV) 也包含在设备文件中。您还可以在设备文件中具体添加设备。LVM 会检测并使用在设备文件中存储的设备列表。

流程

使用以下方法之一在 system.devices 文件中添加设备：

- 通过在设备文件中包括名称来添加设备：

```
$ lvmdevices --adddev <device_name>
```

- 在设备文件中添加卷组 (VG) 中的所有设备：

```
$ vgimportdevices <vg_name>
```

- 在设备文件中添加所有可见的 VG 中的所有设备：

```
$ vgimportdevices --all
```

要隐式将新设备包含到 system.devices 文件中，请使用以下命令之一：

- 使用 pvcreate 命令初始化新设备：

```
$ pvcreate <device_name>
```

  - 这个操作会自动将新物理卷 (PV) 添加到 system.devices 文件中。

- 初始化新设备并在设备文件中自动添加新设备参数：

```
$ vgcreate <vg_name> <device_names>
```

  - 将 ***<vg_name>*** 替换为您要在其中添加设备的 VG 的名称。
  - 将 ***<device_names>*** 替换为您要添加的设备列表。

- 使用 vgextend 命令初始化新设备：

```
$ vgextend <vg_name> <device_names>
```

-
  - 将 *<vg_name>* 替换为您要在其中添加设备的 VG 的名称。
  - 将 *<device_names>* 替换为您要添加的设备名称。
  - 这会在设备文件中自动添加新设备参数。

验证

只有在您需要将新设备显式添加到 system.devices 文件中时，才使用以下验证步骤。

- 显示 system.devices 文件，检查设备列表：

```
$ cat /etc/lvm/devices/system.devices
```

- 更新 system.devices 文件以匹配最新的设备信息：

```
$ lvmdevices --update
```

其他资源

- lvmdevices(8), pvcreate(8), vgcreate(8) 和 vgextend(8) man pages

14.1.3. 从 system.devices 文件中删除设备

删除设备以防止逻辑卷管理器 (LVM) 检测或使用该设备。

流程

- 根据您有关该设备的信息，使用以下方法之一删除设备：
  - 按名称删除设备：

```
$ lvmdevices --deldev <device_name>
```

- 通过该设备的物理卷 ID (PVID) 删除设备：

```
$ lvmdevices --delpvid <PV_UUID>
```

验证

仅在需要明确删除 system.devices 文件中的设备时，才使用以下验证步骤。

- 显示 system.devices 文件进行验证，是否不再存在删除的设备：

```
$ cat /etc/lvm/devices/system.devices
```

- 更新 system.devices 文件以匹配最新的设备信息：

```
$ lvmdevices --update
```

其他资源

- lvmdevices (8) man page

14.1.4. 创建自定义设备文件

逻辑卷管理器 (LVM) 命令使用系统的默认 system.devices 文件。您还可以通过在 LVM 命令中指定新文件名来创建和使用自定义设备文件。当只有某些应用程序需要使用某些设备时，自定义设备文件很有用。

流程

1. 在 /etc/lvm/devices/ 目录中创建自定义设备文件。
2. 在 LVM 命令中包括新设备文件名：

```
$ lvmdevices --devicesfile <devices_file_name>
```

3. 可选：显示新设备文件以验证是否存在新设备名称：

```
$ cat /etc/lvm/devices/<devices_file_name>
```

其他资源

- **lvmdevices (8) man page**

14.1.5. 访问系统中的所有设备

您可以启用逻辑卷管理器(LVM)访问和使用系统中的所有设备，这将覆盖 system.devices 文件中列出的设备所造成的限制。

流程

- **指定空设备文件：**

```
$ lvmdevices --devicesfile ""
```

其他资源

- **lvmdevices (8) man page**

14.1.6. 禁用 system.devices 文件

您可以禁用 system.devices 文件功能。这个操作会自动启用逻辑卷管理器 (LVM) 设备过滤器。

流程

1. **打开 lvm.conf 文件。**
2. **在 devices 部分中设置以下值：**

```
use_devicesfile=0
```

重要

如果您删除 system.devices 文件，则此操作会有效禁用它。即使您在 lvm.conf 配置文件中启用 system.devices 文件，通过在 devices 部分中设置 use_devicesfile=1，也会应用它。禁用设备文件会自动启用 lvm.conf 设备过滤器。

其他资源

- lvmdevices (8) 和 lvm.conf (5) 手册页

14.2. LVM 过滤的持久性标识符

传统的 Linux 设备名称（如 /dev/sda ）可能会在系统修改和重启过程中有所变化。全球识别符(WWID)、通用唯一标识符(UUID)）等持久性命名属性(PNA)和路径名称都基于存储设备的唯一特征，并可适应硬件配置中的变化。这使得它们在系统重启后更稳定且可预测。

在 LVM 过滤中持久性设备标识符的实现提高了 LVM 配置的稳定性和可靠性。它还降低了与设备名称的动态性质相关的系统引导失败的风险。

其他资源

- 持久性命名属性
- 当本地磁盘名称不是持久时，如何配置 lvm 过滤器？

14.3. LVM 设备过滤器

Logical Volume Manager (LVM)设备过滤器是设备名称模式列表。您可以使用它来指定一组强制标准，系统可以使用它来评估设备，并将其视为可以有效地与 LVM 一起使用。LVM 设备过滤器可让您控制 LVM 可以使用哪个设备。这有助于防止意外数据丢失或未授权访问存储设备。

14.3.1. LVM 设备过滤器模式特征

LVM 设备过滤器的模式采用正则表达式的形式。正则表达式用一个字符分隔，前面是 a 表示接受，或 r 表示拒绝。匹配设备的列表中的第一个正则表达式决定了 LVM 接受还是拒绝（忽略）一个特定设备。然后，LVM 会在与设备路径匹配的列表中查找初始正则表达式。LVM 使用这个正则表达式来确定是否应该使用 a 结果批准该设备，或使用r 结果拒绝该设备。

如果单个设备有多个路径名称，LVM 会根据列出的顺序访问这些路径名称。在任何 r 模式之前，如果至少有一个路径名称与 a 模式匹配，则 LVM 会批准该设备。但是，如果 a 模式发现之前，所有路径名称都与 r 模式一致，则设备将被拒绝。

与模式不匹配的路径名称不会影响设备的批准状态。如果没有路径名称与设备的模式对应，则 LVM 仍然会批准该设备。

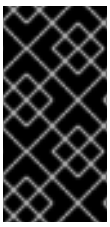
对于系统上的每个设备，udev 规则会生成多个符号链接。目录包含符号链接，如 /dev/disk/by-id/、/dev/disk/by-uuid/、/dev/disk/by-path/，以确保系统上的每个设备都可以通过多个路径名称访问。

要拒绝过滤器中的设备，与该特定设备关联的所有路径名称必须与相应的拒绝 r 表达式匹配。但是，识别要拒绝的所有可能路径名称可能会是一个挑战。因此，最好创建专门接受某些路径，而拒绝所有其他路径的过滤器，使用一系列特定的 a 表达式，后跟一个拒绝所有其它任何东西的 r|.*| 表达式。

在过滤器中定义特定设备时，为该设备使用符号链接名称，而不是内核名称。设备的内核名称可能会改变，如 /dev/sda，而某些符号链接名称不会改变，如 /dev/disk/by-id/wwn the。

默认设备过滤器接受所有连接到系统的设备。理想的用户配置的设备过滤器接受一个或多个模式，并拒绝所有其他模式。例如，以 r|114| 结尾的模式列表。

您可以在 lvm.conf 文件的 devices/filter 和 devices/global_filter 配置字段中找到 LVM 设备过滤器配置。devices/filter 和 devices/global_filter 配置字段是等同的。



重要

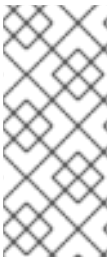
在 Red Hat Enterprise Linux 9 中，默认启用 /etc/lvm/devices/system.devices 文件。当禁用 system.devices 文件时，系统会自动启用 LVM 设备过滤器。

其他资源

- lvm.conf(5) 手册页

14.3.2. LVM 设备过滤器配置示例

以下示例显示了过滤配置，以控制 LVM 扫描并稍后使用的设备。要在 lvm.conf 文件中配置设备过滤器，请参阅



注意

在处理复制或克隆的 PV 时，您可能会遇到重复的物理卷(PV)警告。您可以设置过滤器来解决这个问题。请参阅 防止重复 PV 警告的 LVM 设备过滤器示例 中的过滤器配置示例。

- 要扫描所有设备，请输入：

```
filter = [ "a|.*|" ]
```

- 要删除 cdrom 设备，以避免在驱动器不包含介质时的延迟，请输入：

```
filter = [ "r|^/dev/cdrom$|" ]
```

- 要添加所有循环设备并删除所有其他设备，请输入：

```
filter = [ "a|loop|", "r|.*|" ]
```

- 要添加所有循环和 SCSI 设备，并删除所有其他块设备，请输入：

```
filter = [ "a|loop|", "a|/dev/sd.*|", "r|.*|" ]
```

- 要只在第一个 SCSI 驱动器上添加分区 8，并删除所有其他块设备，请输入：

```
filter = [ "a|^/dev/sda8$|", "r|.*|" ]
```

- 要从 WWID 标识的特定设备和所有多路径设备添加所有分区，请输入：

```
filter = [ "a|/dev/disk/by-id/<disk-id>.|", "a|/dev/mapper/mpath.|", "r|.*|" ]
```

命令还会删除任何其他块设备。

其他资源

- lvm.conf(5) 手册页

14.3.3. 应用 LVM 设备过滤器配置

您可以通过在 lvm.conf 配置文件中设置过滤器来控制 LVM 扫描哪个设备。

先决条件

- 您已禁用了 system.devices 文件功能。
- 您已准备了要使用的设备过滤器模式。

流程

1. 使用以下命令来测试设备过滤器模式，而不是实际修改 /etc/lvm/lvm.conf 文件。以下包括一个示例过滤器配置。

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|*.|" ] }'
```

2. 在 /etc/lvm/lvm.conf 文件的配置部分 devices 中添加设备过滤器模式：

```
filter = [ "a|/dev/emcpower.*|", "r|*.|" ]
```

3. 在重启时仅扫描所需的设备：

```
# dracut --force --verbose
```

这个命令重建 initramfs 文件系统，以便 LVM 在重启时只扫描必要的设备。

第 15 章 控制 LVM 分配

默认情况下，卷组使用 normal 分配策略。这会根据通用规则（如不在同一物理卷上放置并行条带）分配物理扩展。您可以使用 vgcreate 命令的 --alloc 参数指定不同的分配策略(contiguous、anywhere 或 cling)。通常，只在特殊情况下需要指定非通常或非标准扩展分配时，才需要不同于 normal 的分配策略。

15.1. 从指定的设备分配扩展

您可以在 lvcreate 和 lvconvert 命令的命令行末尾使用 device 参数来限制特定设备的分配。您可以为每个设备指定实际的扩展范围，以提供更多控制。命令只使用指定的物理卷(PV)作为参数，来为新逻辑卷(LV)分配扩展。它使用每个 PV 的可用扩展，直到它们耗尽，然后使用列出的下一个 PV 中的扩展。如果所有列出的 PV 上没有足够的空间用于请求的 LV 大小，则命令失败。请注意，命令只从指定的 PV 进行分配。RAID LV 对单独的 raid 镜像或单独的条带使用顺序 PV 。如果对于整个 raid 镜像来说 PV 不够大，则生成的设备的使用是不能完全预测的。

流程

1. 创建卷组(VG)：

```
# vgcreate <vg_name> <PV> ...
```

其中：

- ***<vg_name>*** 是 VG 的名称。
- ***<PV>*** 是 PV。

2. 您可以分配 PV 来创建不同的卷类型，如 linear 或 raid ：

   a. 分配扩展以创建线性卷：

```
# lvcreate -n <lv_name> -L <lv_size> <vg_name> [ <PV> ... ]
```

其中：

- *<lv_name>* 是 LV 的名称。
- *<lv_size>* 是 LV 的大小。默认单位是 MB。
- *<vg_name>* 是 VG 的名称。
- [*<PV ...>*] 是 PV。

  您可以在命令行上指定其中一个 PV、所有 PV 或 none ：

  - 如果您指定一个 PV，则会从它分配 LV 的扩展。

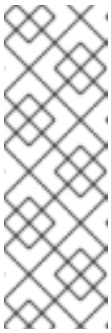
    

    注意

    如果对整个 LV 来说 PV 没有足够的可用扩展，则 lvcreate 失败。

  - 如果您指定了两个 PV，则 LV 的扩展将从其中一个 PV 或两者的组合中分配。

  - 如果没有指定任何 PV，则扩展将从 VG 中的一个 PV 或者 VG 中所有 PV 的组合中分配。

    注意

    在这些情况下，LVM 可能没有使用所有命名的 PV 或可用的 PV。如果第一个 PV 有足够的空闲扩展用于整个 LV，则可能不使用其他 PV。但是，如果第一个 PV 没有设置空闲扩展的分配大小，则 LV 可能会从第一个 PV 中分配一部分，并从第二个 PV 中分配一部分。

例 15.1. 从一个 PV 分配扩展

在这个示例中，lv1 扩展将从 sda 中分配。

```
# lvcreate -n lv1 -L1G vg /dev/sda
```

例 15.2. 从两个 PV 中分配扩展

在这个示例中，lv2 扩展将从 sda 或 sdb 或两者的组合中分配。

```
# lvcreate -n lv2 L1G vg /dev/sda /dev/sdb
```

例 15.3. 在不指定 PV 的情况下分配扩展

在本例中，lv3 扩展将从 VG 中的一个 PV 或 VG 中所有 PV 的组合中分配。

```
# lvcreate -n lv3 -L1G vg
```

或者

b.

分配扩展以创建 raid 卷：

```
# lvcreate --type <segment_type> -m <mirror_images> -n <lv_name> -L <lv_size>
<vg_name> [ <PV> ... ]
```

其中：

- ***<segment_type>* 是指定的片段类型（如 raid5、mirror、snapshot）。**
- ***<mirror_images>* 创建一个 raid1 或具有指定镜像数的镜像 LV。例如，-m 1 会产生具有两个镜像的 raid1 LV。**
- ***<lv_name>* 是 LV 的名称。**
- ***<lv_size>* 是 LV 的大小。默认单位是 MB。**

- *<vg_name>* 是 VG 的名称。

- *<[PV ...]>* 是 PV。

  第一个 raid 镜像将从第一个 PV 中分配，第二个 raid 镜像将从第二个 PV 中分配，以此类推。

例 15.4. 从两个 PV 中分配 raid 镜像

在这个示例中，第一个镜像 lv4 将从 sda 中分配，第二个镜像将从 sdb 中分配。

```
# lvcreate --type raid1 -m 1 -n lv4 -L1G vg /dev/sda /dev/sdb
```

例 15.5. 从三个 PV 中分配 raid 镜像

在本例中，第一个 raid 镜像 lv5 将从 sda 中分配，第二个镜像将从 sdb 中分配，第三个镜像将从 sdc 中分配。

```
# lvcreate --type raid1 -m 2 -n lv5 -L1G vg /dev/sda /dev/sdb /dev/sdc
```

其他资源

- lvcreate(8) 手册页

- lvconvert(8) 手册页

- lvmraid (7) 手册页

15.2. LVM 分配策略

当 LVM 操作必须为一个或多个逻辑卷(LV)分配物理扩展时，分配过程如下：

-

生成卷组中的未分配物理扩展的完整集以供考虑。如果您在命令行末尾提供了任意物理扩展的范围，则只考虑指定物理卷(PV)上这些范围内未分配的物理扩展。

- 依次尝试每个分配策略，从最严格的策略(contiguous)开始，以使用 --alloc 选项指定的分配策略结束，或者设置为特定 LV 或卷组(VG)的默认分配策略。对于每个策略，从需要填充的空 LV 空间的最低编号的逻辑扩展开始工作，根据分配策略施加的限制，分配尽可能多的空间。如果需要更多空间，LVM 会进入下一个策略。

分配策略的限制如下：

- contiguous 策略要求任何逻辑扩展的物理位置紧挨着在前一个逻辑扩展的物理位置，但 LV 的第一个逻辑扩展除外。

  当 LV 为条带或镜像时，contiguous 分配限制独立应用于每个需要空间的条带或 raid 镜像。

- cling 分配策略要求将用于任何逻辑扩展的 PV 添加到现有 LV 中，该 LV 之前已被 LV 中至少一个逻辑扩展使用。

- normal 分配策略不会选择共享同一 PV 的物理扩展，因为逻辑卷已在并行 LV 中的相同偏移处分配给了一个并行 LV（即，不同的条带或 raid 镜像）。

- 如果有足够的空闲扩展满足分配请求，但 normal 分配策略不使用它们，则 anywhere 分配策略将使用它们，即使将两个条带放在同一 PV 上会降低性能。

您可以使用 vgchange 命令更改分配策略。

注意

未来的更新可能会根据定义的分配策略在布局行为中引入代码更改。例如：如果您在命令行中提供两个空物理卷，它们有相同数量的可用物理扩展可用于分配，LVM 当前会以它们列出的顺序处理它们，但不保证在将来的版本中这个行为不会有变化。如果您需要特定 LV 的具体布局，请通过 lvcreate 和 lvconvert 步骤序列构建它，以便应用到每个步骤的分配策略不会让 LVM 自行决定布局。

15.3. 防止在物理卷中分配

您可以使用 pvchange 命令防止在一个或多个物理卷的空闲空间上分配物理扩展。如果有磁盘错误或者要删除物理卷，这可能是必要的。

流程

- 使用以下命令禁止在 *device_name* 上分配物理扩展：

```
# pvchange -x n /dev/sdk1
```

  您还可以使用 pvchange 命令的 -xy 参数允许对之前禁止分配的设备进行分配。

其他资源

- pvchange (8) 手册页

第 16 章 使用标签对 LVM 对象进行分组

您可以为逻辑卷管理(LVM)对象分配标签来对它们进行分组。使用这个功能，您可以按组对 LVM 行为（如激活）进行自动控制，比如激活。您还可以将 LVM 对象上的标签用作命令。

16.1. LVM 对象标签

逻辑卷管理(LVM)标签是一个单词，用于对相同类型的 LVM2 对象进行分组。您可以将标签附加到对象，如物理卷、卷组和逻辑卷。

为避免歧义，请在每个标签前面加上 @ 前缀。通过将其替换为拥有该标签的所有对象，并且该标签是命令行上其位置所期望的类型来扩展每个标签。

LVM 标签是最多 1024 个字符的字符串。LVM 标签不能以连字符开头。

有效的标签仅由有限的字符范围组成。允许的字符是 A-Z a-z 0-9 _ + . - / = ! : # & amp; .

只能标记卷组中的对象。如果从卷组中移除了标签，则物理卷会丢失标签；这是因为标签作为卷组元数据的一部分存储，并在物理卷被删除时删除。

您可以将一些命令应用到具有相同标签的所有卷组(VG)、逻辑卷(LV)或物理卷(PV)。给定命令的手册页显示语法，如 VG|Tag、LV|Tag 或 PV|Tag，当您可以替换 VG、LV 或 PV 名称的标签名称时。

16.2. 向 LVM 对象添加标签

您可以使用 --addtag 选项和各种卷管理命令向 LVM 对象添加标签来对它们进行分组。

先决条件

- 已安装 lvm2 软件包。

流程

- 要向现有的 PV 添加标签，请使用：

```
# pvchange --addtag <@tag> <PV>
```

- 要向现有的 VG 添加标签，请使用：

```
# vgchange --addtag <@tag> <VG>
```

- 要在创建过程中向 VG 添加标签，请使用：

```
# vgcreate --addtag <@tag> <VG>
```

- 要向现有的 LV 添加标签，请使用：

```
# lvchange --addtag <@tag> <LV>
```

- 要在创建过程中向 LV 添加标签，请使用：

```
# lvcreate --addtag <@tag> ...
```

16.3. 从 LVM 对象中删除标签

如果您不再希望保留 LVM 对象分组，您可以使用 **--deltag** 选项和各种卷管理命令从对象中删除标签。

先决条件

- lvm2 软件包已安装。
- 您已在物理卷(PV)、卷组(VG)或逻辑卷(LV)上创建了标签。

流程

- 要从现有的 PV 中删除标签，请使用：

```
# pvchange --deltag @tag PV
```

- 要从现有的 VG 中删除标签，请使用：

```
# vgchange --deltag @tag VG
```

- 要从现有的 LV 中删除标签，请使用：

```
# lvchange --deltag @tag LV
```

16.4. 显示 LVM 对象上的标签

您可以使用以下命令显示 LVM 对象上的标签。

先决条件

- lvm2 软件包已安装。
- 您已在物理卷(PV)、卷组(VG)或逻辑卷(LV)上创建了标签。

流程

- 要显示现有 PV 上的所有标签，请使用：

```
# pvs -o tags <PV>
```

- 要显示现有 VG 上的所有标签，请使用：

```
# vgs -o tags <VG>
```

- 要显示现有 LV 上的所有标签，请使用：

```
# lvs -o tags <LV>
```

16.5. 使用标签控制逻辑卷激活

这个步骤描述了如何在该主机上激活特定逻辑卷的配置文件中指定。

流程

例如，以下条目充当激活请求的过滤器（如 **vgchange -ay**），仅激活 **vg1/lvol0** 以及具有该主机上元数据中的 **数据**库 标签的任何逻辑卷或卷组：

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

特殊匹配 **@***，仅当任何元数据标签匹配该机器上的任何主机标签时才使匹配。

另外，请考虑集群中每台机器在配置文件中有以下条目的情况：

```
tags { hosttags = 1 }
```

如果您想仅在主机 **db2** 上激活 **vg1/lvol2**，请执行以下操作：

1. 从集群中的任何主机运行 **lvchange --addtag @db2 vg1/lvol2**。
2. 运行 **lvchange -ay vg1/lvol2**。

这个解决方案涉及将主机名存储在卷组元数据中。

第 17 章 LVM 故障排除

您可以使用逻辑卷管理器(LVM)工具排除 LVM 卷和组群中的各种问题。

17.1. 在 LVM 中收集诊断数据

如果 LVM 命令没有按预期工作，您可以使用以下方法收集诊断信息。

流程

- 使用以下方法收集不同类型的诊断数据：
  - 向任何 LVM 命令添加 **-v** 参数，以提高命令输出的详细程度。添加更多的 **v** 会进一步增加输出的详细程度。最多允许 4 个这样的 **v**，例如 **-vvvv**。
  - 在 **/etc/lvm/lvm.conf** 配置文件的 **log** 部分中，增加 **level** 选项的值。这会导致 LVM 在系统日志中提供更多详情。
  - 如果问题与逻辑卷激活有关，请启用 LVM 在激活过程中记录信息：
    i. 在 **/etc/lvm/lvm.conf** 配置文件的 **log** 部分中设置 **activation = 1** 选项。
    ii. 使用 **-vvvv** 选项执行 LVM 命令。
    iii. 检查命令输出。
    iv. 将 **activation** 选项重置为 **0**。

    如果您没有将选项重置为 **0**，则系统在内存不足时可能会变得无响应。
  - 为诊断显示信息转储：

```
# lvmdump
```

- 显示附加系统信息：

```
# lvs -v
```

```
# pvs --all
```

```
# dmsetup info --columns
```

- 检查 /etc/lvm/backup/ 目录中的最后一个 LVM 元数据备份，并在 /etc/lvm/archive/ 目录中检查存档版本。

- 检查当前的配置信息：

```
# lvmconfig
```

- 检查 /run/lvm/hints 缓存文件以获取哪些设备上具有物理卷的记录。

其他资源

- lvmdump(8) 手册页

17.2. 显示有关失败的 LVM 设备的信息

有关失败逻辑卷管理器(LVM)卷的故障排除信息可帮助您确定失败的原因。您可以检查以下最常见的 LVM 卷失败的示例。

例 17.1. 失败的卷组

在本例中，组成卷组 *myvg* 的设备之一失败。卷组可用性取决于故障类型。例如，如果还涉及到 RAID 卷，卷组仍可用。您还可以查看有关失败的设备的信息。

```
# vgs --options +devices
 /dev/vdb1: open failed: No such device or address
 /dev/vdb1: open failed: No such device or address
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
```

```
/dev/sdb1).
  WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
devices.

VG    #PV #LV #SN Attr   VSize  VFree  Devices
myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](0)
myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](5120),/dev/vdb1(0)
```

例 17.2. 逻辑卷失败

在这个示例中，其中一个设备失败了。这可能是卷组中逻辑卷失败的原因。命令输出显示失败的逻辑卷。

```
# lvs --all --options +devices

  /dev/vdb1: open failed: No such device or address
  /dev/vdb1: open failed: No such device or address
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
/dev/sdb1).
  WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
devices.

  LV    VG  Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
  mylv myvg -wi-a---p- 20.00g                                                [unknown](0)
[unknown](5120),/dev/sdc1(0)
```

例 17.3. 失败的 RAID 逻辑卷的镜像

下面的例子显示 RAID 逻辑卷的镜像失败时，pvs 和 lvs 工具的命令输出。逻辑卷仍然可用。

```
# pvs

  Error reading device /dev/sdc1 at 0 length 4.

  Error reading device /dev/sdc1 at 4096 length 4.

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and
assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and
assumed devices.

  PV         VG         Fmt  Attr PSize    PFree
  /dev/sda2  rhel_bp-01 lvm2 a--  <464.76g    4.00m
  /dev/sdb1  myvg       lvm2 a--  <836.69g  736.68g
```

```
  /dev/sdd1    myvg       lvm2 a--  <836.69g <836.69g
  /dev/sde1    myvg       lvm2 a--  <836.69g <836.69g
  [unknown]    myvg       lvm2 a-m  <836.69g  736.68g
```

```
# lvs -a --options name,vgname,attr,size,devices myvg

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and assumed devices.

  LV                VG   Attr       LSize   Devices
  my_raid1          myvg rwi-a-r-p- 100.00g my_raid1_rimage_0(0),my_raid1_rimage_1(0)
  [my_raid1_rimage_0] myvg iwi-aor--- 100.00g /dev/sdb1(1)
  [my_raid1_rimage_1] myvg Iwi-aor-p- 100.00g [unknown](1)
  [my_raid1_rmeta_0]  myvg ewi-aor---   4.00m /dev/sdb1(0)
  [my_raid1_rmeta_1]  myvg ewi-aor-p-   4.00m [unknown](0)
```

17.3. 从卷组中删除丢失的 LVM 物理卷

如果物理卷失败，您可以激活卷组中剩余的物理卷，并从卷组中删除所有使用该物理卷的逻辑卷。

流程

1. **激活卷组中剩余的物理卷：**

```
# vgchange --activate y --partial myvg
```

2. **检查要删除哪些逻辑卷：**

```
# vgreduce --removemissing --test myvg
```

3. **从卷组中删除所有使用丢失的物理卷的逻辑卷：**

```
# vgreduce --removemissing --force myvg
```

4. **可选：如果您意外删除要保留的逻辑卷，您可以撤销 vgreduce 操作：**

```
# vgcfgrestore myvg
```

-

警告

如果您删除了精简池，LVM 无法撤销操作。

17.4. 查找丢失的 LVM 物理卷的元数据

如果意外覆盖或者破坏了卷组物理卷元数据区域，您会得到出错信息表示元数据区域不正确，或者系统无法使用特定的 UUID 找到物理卷。

这个过程找到丢失或者损坏的物理卷的最新归档元数据。

流程

1. 查找包含物理卷的卷组元数据文件。归档的元数据文件位于 **/etc/lvm/archive/*volume-group-name_backup-number*.vg** 路径中：

```
# cat /etc/lvm/archive/myvg_00000-1248998876.vg
```

使用备份号替换 ***00000-1248998876***。选择该卷组最高数字最后已知的有效元数据文件。

2. 找到物理卷的 UUID。使用以下任一方法。

- 列出逻辑卷：

```
# lvs --all --options +devices

 Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
```

- 检查归档的元数据文件。在卷组配置的 **physical_volumes** 部分中，查找标记为 **id =** 的 **UUID**。

- **使用 --partial 选项取消激活卷组：**

```
# vgchange --activate n --partial myvg

  PARTIAL MODE. Incomplete logical volumes will be processed.
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to /dev/vdb1).
  0 logical volume(s) in volume group "myvg" now active
```

17.5. 在 LVM 物理卷中恢复元数据

这个过程恢复被损坏或者替换为新设备的物理卷的元数据。您可以通过重写物理卷的元数据区域从物理卷中恢复数据。

警告

不要在正常的 LVM 逻辑卷中尝试这个步骤。如果您指定了不正确的 UUID，将会丢失您的数据。

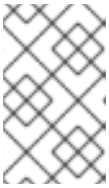
先决条件

- 您已找出丢失的物理卷的元数据。详情请查看查找缺少的 LVM 物理卷的元数据。

流程

1. **恢复物理卷中的元数据：**

```
# pvcreate --uuid physical-volume-uuid \
        --restorefile /etc/lvm/archive/volume-group-name_backup-number.vg \
        block-device
```



注意

该命令只覆盖 LVM 元数据区域，不会影响现有的数据区域。

例 17.4. 在 */dev/vdb1*上恢复物理卷

以下示例使用以下属性将 */dev/vdb1* 设备标记为物理卷：

- FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk 的 UUID
- VG_00050.vg 中包含的元数据信息，它是卷组最新的好归档元数据。

```
# pvcreate --uuid "FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk" \
        --restorefile /etc/lvm/archive/VG_00050.vg \
        /dev/vdb1

  ...
  Physical volume "/dev/vdb1" successfully created
```

2. 恢复卷组的元数据：

```
# vgcfgrestore myvg

  Restored volume group myvg
```

3. 显示卷组中的逻辑卷：

```
# lvs --all --options +devices myvg
```

逻辑卷目前不活跃。例如：

```
  LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  mylv myvg   -wi--- 300.00G                             /dev/vdb1 (0),/dev/vdb1(0)
  mylv myvg   -wi--- 300.00G                             /dev/vdb1 (34728),/dev/vdb1(0)
```

4. 如果逻辑卷的片段类型是 RAID，则重新同步逻辑卷：

```
# lvchange --resync myvg/mylv
```

5. 激活逻辑卷：

```
# lvchange --activate y myvg/mylv
```

6. 如果磁盘中的 LVM 元数据至少使用了覆盖其数据的空间，这个过程可以恢复物理卷。如果覆盖元数据的数据超过了元数据区域，则该卷中的数据可能会受到影响。您可能能够使用 fsck 命令恢复这些数据。

验证步骤

- 显示活跃逻辑卷：

```
# lvs --all --options +devices

 LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
 mylv myvg   -wi--- 300.00G                                /dev/vdb1 (0),/dev/vdb1(0)
 mylv myvg   -wi--- 300.00G                                /dev/vdb1 (34728),/dev/vdb1(0)
```

17.6. LVM 输出中的轮询错误

LVM 命令报告卷组中的空间使用情况，将报告的编号舍入到 2 个十进制位置，以提供人类可读的输出。这包括 vgdisplay 和 vgs 实用程序。

因此，报告的剩余空间值可能大于卷组中物理扩展提供的内容。如果您试图根据报告可用空间的大小创建逻辑卷，则可能会遇到以下错误：

```
Insufficient free extents
```

要临时解决这个问题，您必须检查卷组中可用物理扩展的数量，即可用空间的具体值。然后您可以使用扩展数目成功创建逻辑卷。

17.7. 防止创建 LVM 卷时出现循环错误

在创建 LVM 逻辑卷时，您可以指定逻辑卷的逻辑扩展数目以避免循环错误。

流程

1. 在卷组中找到可用物理扩展数目：

```
# vgdisplay myvg
```

例 17.5. 卷组中可用扩展

例如：以下卷组有 **8780** 可用物理扩展：

```
--- Volume group ---
 VG Name               myvg
 System ID
 Format                lvm2
 Metadata Areas        4
 Metadata Sequence No  6
 VG Access             read/write
[...]
Free  PE / Size       8780 / 34.30 GB
```

2. 创建逻辑卷。以扩展而不是字节为单位输入卷大小。

例 17.6. 通过指定扩展数目来创建逻辑卷

```
# lvcreate --extents 8780 --name mylv myvg
```

例 17.7. 创建逻辑卷以占据所有剩余空间

另外，您可以扩展逻辑卷使其使用卷组中剩余的可用空间的比例。例如：

```
# lvcreate --extents 100%FREE --name mylv myvg
```

验证步骤

- 检查卷组现在使用的扩展数目：

```
# vgs --options +vg_free_count,vg_extent_count

  VG     #PV #LV #SN  Attr   VSize   VFree  Free  #Ext
  myvg   2   1   0   wz--n- 34.30G    0    0     8780
```

17.8. LVM 元数据及其在磁盘上的位置

提供不同偏移和大小的 LVM 标头和元数据区域。

默认的 LVM 磁盘标头：

- 可在 label_header 和 pv_header 结构中找到。
- 在磁盘的第二个 512 字节扇区中。请注意，如果在创建物理卷(PV)时没有指定默认位置，则标头也可以在第一个或第三个扇区中。

标准的 LVM 元数据区域：

- 从磁盘开始的头 4096 个字节。
- 从磁盘开始的最后 1 MiB。
- 从包含 mda_header 结构的 512 字节扇区开始。

元数据文本区域从 mda_header 扇区之后开始，一直到元数据区域的末尾。LVM VG 元数据文本以循环方式写入元数据文本区域中。mda_header 指向文本区域中最新 VG 元数据的位置。

您可以使用 # pvck --dump headers /dev/sda 命令打印磁盘中的 LVM 标头。此命令打印 label_header、pv_header、mda_header 以及元数据文本的位置（如果发现的话）。错误字段使用 CHECK 前缀打印。

LVM 元数据区偏移将匹配创建 PV 的机器的页大小，因此元数据区域也可以从磁盘开始的 8K、16K 或 64K 开始。

在创建 PV 时可以指定较大或较小的元数据区域，在这种情况下，元数据区域可能不在 1 MiB 处结束。pv_header 指定元数据区域的大小。

在创建 PV 时，可选择在磁盘末尾处启用第二个元数据区域。pv_header 包含元数据区域的位置。

17.9. 从磁盘中提取 VG 元数据

根据您的情况，选择以下流程之一，来从磁盘中提取 VG 元数据。有关如何保存提取的元数据的详情，请参考 将提取的元数据保存在文件中。

注意

要进行修复，您可以使用 /etc/lvm/backup/ 中的备份文件，而无需从磁盘中提取元数据。

流程

- 打印有效的 mda_header 中提到的当前元数据文本：

```
# pvck --dump metadata <disk>
```

例 17.8. 有效的 mda_header 中的元数据文本

```
# pvck --dump metadata /dev/sdb
  metadata text at 172032 crc Oxc627522f # vgname test segno 59
  ---
  <raw metadata from disk>
  ---
```

- 根据找到的有效的 mda_header，打印元数据区域中找到的所有元数据副本的位置：

```
# pvck --dump metadata_all <disk>
```

例 17.9. 元数据区域中元数据副本的位置

```
# pvck --dump metadata_all /dev/sdb
  metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
```

- 搜索元数据区域中所有的元数据副本，而不使用 mda_header，例如，如果标头丢失或损坏：

```
# pvck --dump metadata_search <disk>
```

例 17.10. 不使用 mda_header 的元数据区域中的元数据副本

```
# pvck --dump metadata_search /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
```

- **在 dump 命令中包含 -v 选项，以显示每个元数据副本的描述：**

```
# pvck --dump metadata -v <disk>
```

例 17.11. 显示元数据的每个副本的描述

```
# pvck --dump metadata -v /dev/sdb
  metadata text at 199680 crc 0x628cf243 # vgname my_vg seqno 40
  ---
my_vg {
id = "dmEbPi-gsgx-VbvS-Uaia-HczM-iu32-Rb7iOf"
seqno = 40
format = "lvm2"
status = ["RESIZEABLE", "READ", "WRITE"]
flags = []
extent_size = 8192
max_lv = 0
max_pv = 0
metadata_copies = 0

physical_volumes {

pv0 {
id = "8gn0is-Hj8p-njgs-NM19-wuL9-mcB3-kUDiOQ"
device = "/dev/sda"

device_id_type = "sys_wwid"
device_id = "naa.6001405e635dbaab125476d88030a196"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}

pv1 {
id = "E9qChJ-5ElL-HVEp-rc7d-U5Fg-fHxL-2QLyID"
device = "/dev/sdb"

device_id_type = "sys_wwid"
```

```
device_id = "naa.6001405f3f9396fddcd4012a50029a90"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}
```

此文件可用于修复。默认情况下，第一个元数据区域用于转储元数据。如果磁盘在磁盘末尾处有第二个元数据区域，则您可以使用 **--settings "mda_num=2"** 选项来将第二个元数据区域用于转储元数据。

17.10. 将提取的元数据保存到文件中

如果您需要使用转储的元数据进行修复，则需要使用 **-f** 选项和 **--setings** 选项将提取的元数据保存到文件中。

流程

- 如果将 **-f <filename>** 添加到 **--dump metadata** 中，则原始元数据被写入到指定的文件中。您可以使用此文件进行修复。

- 如果将 **-f <filename>** 添加到 **--dump metadata_all** 或 **--dump metadata_search** 中，则所有位置的原始元数据都被写入到指定的文件中。

- 要保存 **--dump metadata_all|metadata_search add --settings "metadata_offset= <offset>"** 中的一个元数据文本的实例，其中 **<offset>** 来自列表输出 **"metadata at <offset>"**。

例 17.12. 命令的输出

```
# pvck --dump metadata_search --settings metadata_offset=5632 -f meta.txt /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-
Xl4i-zb6G-BYat-u53Fxv
# head -2 meta.txt
test {
id = "FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv"
```

17.11. 使用 PVCREATE 和 VGCFGRESTORE 命令修复带有损坏的 LVM 标头和元数据的磁盘

您可以恢复损坏的或者使用新设备替换的物理卷上的元数据和标头。您可以通过重写物理卷的元数据区域从物理卷中恢复数据。

警告

这些指令应当谨慎使用，只有在您熟悉每个命令的含义、卷的当前布局、您需要实现的布局以及备份元数据文件的内容时才应使用。这些命令可能会损坏数据，因此建议您联系红帽全球支持服务来帮助进行故障排除。

先决条件

- 您已找出丢失的物理卷的元数据。详情请查看查找缺少的 LVM 物理卷的元数据。

流程

1. 收集 pvcreate 和 vgcfgrestore 命令需要的以下信息。您可以通过运行 # pvs -o+uuid 命令收集有关磁盘和 UUID 的信息。
   - metadata-file 是 VG 的最新元数据备份文件的路径，例如：/etc/lvm/backup/*<vg-name>*
   - VG-name 是有损坏或缺失 PV 的 VG 的名称。
   - 此设备上损坏的 PV 的 UUID 是从 # pvs -i+uuid 命令的输出中获得的值。
   - disk 是 PV 所在磁盘的名称，例如 /dev/sdb。请注意，这是正确的磁盘，或寻求帮助，否则以下这些步骤可能导致数据丢失。
2. 在磁盘上重新创建 LVM 标头：

```
# pvcreate --restorefile <metadata-file> --uuid <UUID> <disk>
```

（可选）验证标头是否有效：

```
# pvck --dump headers <disk>
```

3. 恢复磁盘上的 VG 元数据：

```
# vgcfgrestore --file <metadata-file> <vg-name>
```

（可选）验证元数据是否已恢复：

```
# pvck --dump metadata <disk>
```

如果没有 VG 的元数据备份文件，您可以使用 将提取的元数据保存到文件 中的流程来获得。

验证

- 要验证新物理卷是否完整，且卷组是否正常工作，请检查以下命令的输出：

```
# vgs
```

其他资源

- pvck (8)手册页
- 从物理卷中提取 LVM 元数据备份
- 如何在线修复物理卷上的元数据？
- 如果组成卷组的一个物理卷失败，如何在 Red Hat Enterprise Linux 上恢复卷组？

17.12. 使用 PVCK 命令修复带有损坏的 LVM 标头和元数据的磁盘

这是 使用 pvcreate 和 vgcfgrestore 命令修复带有损坏的 LVM 标头和元数据的磁盘 的替代方法。有些情况下，pvcreate 和 vgcfgrestore 命令可能无法正常工作。这个方法更针对损坏的磁盘。

此方法使用 pvck --dump 提取的元数据输入文件，或者 /etc/lvm/backup 中的备份文件。在可能的情

况下，使用 pvck --dump 从同一 VG 中的其他 PV 中保存的元数据，或者从 PV 上的第二个元数据区域中保存的元数据。如需更多信息，请参阅 将提取的元数据保存到文件中。

流程

- 修复磁盘上的标头和元数据：

```
# pvck --repair -f <metadata-file> <disk>
```

其中

- *<metadata-file>* 是包含 VG 的最新元数据的文件。这可以是 /etc/lvm/backup/*vg-name*，也可以是包含 pvck --dump metadata_search 命令输出中的原始元数据文本的文件。
- *<disk>* 是 PV 所在的磁盘的名称，例如 /dev/sdb。要防止数据丢失，请验证是否为正确的磁盘。如果您不确定磁盘是否正确，请联系红帽支持团队。

注意

如果元数据文件是一个备份文件，则 pvck --repair 应在 VG 中保存元数据的每个 PV 上运行。如果元数据文件是从另一个 PV 中提取的原始元数据，则仅需要在损坏的 PV 上运行 pvck --repair。

验证

- 要检查新物理卷是否完整，且卷组是否工作正常，请检查以下命令的输出：

```
# vgs <vgname>
```

```
# pvs <pvname>
```

```
# lvs <lvname>
```

其他资源

- pvck (8)手册页

- 从物理卷中提取 LVM 元数据备份。
- 如何在线修复物理卷上的元数据？
- 如果组成卷组的一个物理卷失败，如何在 Red Hat Enterprise Linux 上恢复卷组？

17.13. LVM RAID 故障排除

您可以对 LVM RAID 设备中的多个问题进行故障排除，修正数据错误、恢复设备或者替换失败的设备。

17.13.1. 检查 RAID 逻辑卷中数据的一致性

LVM 提供对 RAID 逻辑卷的清理支持。RAID 清理是读取阵列中的所有数据和奇偶校验块的过程，并检查它们是否是分配的。lvchange --syncaction repair 命令对阵列启动后台同步操作。以下属性提供有关数据一致性的详情：

- raid_sync_action 字段显示当前 RAID 逻辑卷正在执行的同步操作。可以是以下值之一：

  idle

  完成所有 sync 操作（什么都不做）。

  resync

  在不干净的机器关闭后初始化或重新同步阵列。

  recover

  替换阵列中的设备。

  check

  查找阵列的不一致。

  repair

  查找并修复不一致.

- raid_mismatch_count 字段显示 check 操作过程中发现的差异数。

- Cpy%Sync 字段显示 sync 操作的进度。

- lv_attr 字段提供额外的指示。这个字段中的第 9 位显示逻辑卷的健康状况，它支持以下指示：

  m 或 mismatches

  表示 RAID 逻辑卷中存在差异。您可以在清理操作检测到 RAID 部分中的不一致后看到此字符。

  r 或 refresh

  表示 RAID 阵列中失败的设备，即使 LVM 可以读取设备标签，并认为设备可以正常工作。刷新逻辑卷通知内核该设备现在可用；如果您怀疑设备失败，则替换该设备。

流程

1. 可选：限制清理过程使用的 I/O 带宽。当您执行 RAID 清理操作时，sync 操作所需的后台 I/O 可能会将其他 I/O 分离到 LVM 设备，如对卷组元数据的更新。这可能导致其它 LVM 操作速度下降。

   您可以使用节流功能控制清理操作的速度。您可以将 **--maxrecoveryrate Rate[bBsSkKmMgG]** 或 **--minrecoveryrate Rate[bBsSkKmMgG]** 与 **lvchange --syncaction** 命令一起使用来设置恢复率。如需更多信息，请参阅 最小和最大 I/O 速率选项。

   指定*比率*，格式为“数量/每秒/阵列中的每个设备”。如果没有后缀，选项会假定为 kiB/每秒/每个设备。

2. 显示阵列中未修复的差异的数量，没有修复它们：

   ```
   # lvchange --syncaction check my_vg/my_lv
   ```

   此命令对阵列启动后台同步操作。

3. 可选：查看 var/log/syslog 文件以了解内核消息。

4. 修正阵列中的差异：

```
# lvchange --syncaction repair my_vg/my_lv
```

这个命令修复或者替换 RAID 逻辑卷中失败的设备。您可以在执行此命令后查看 var/log/syslog 文件以了解内核消息。

验证

1. 显示有关清理操作的信息：

```
# lvs -o +raid_sync_action,raid_mismatch_count my_vg/my_lv
LV    VG    Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
SyncAction Mismatches
my_lv my_vg rwi-a-r--- 500.00m                                100.00          idle        0
```

其他资源

- lvchange (8) 和 lvmraid (7) 手册页
- 最小和最大 I/O 速率选项

17.13.2. 在逻辑卷中替换失败的 RAID 设备

RAID 与传统的 LVM 镜像不同。如果是 LVM 镜像，请删除失败的设备。否则，当 RAID 阵列继续使用失败的设备运行时，镜像逻辑卷将挂起。对于 RAID1 以外的 RAID 级别，删除设备意味着转换到较低的 RAID 级别，例如从 RAID6 转换到 RAID5，或者从 RAID4 或 RAID5 转换到 RAID0。

您可以使用 lvconvert 命令的 --repair 参数替换 RAID 逻辑卷中作为物理卷的故障设备，而不是删除失败的设备并分配一个替换设备。

先决条件

- 卷组包含一个物理卷，它有足够的可用容量替换失败的设备。

如果卷组中没有足够空闲扩展的物理卷，请使用 vgextend 工具添加一个新的、足够大的物理卷。

流程

1. 查看 **RAID** 逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
  LV              Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdc1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdc1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

2. 在 */dev/sdc* 设备失败后查看 **RAID** 逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking
used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking
used and assumed devices.
  LV              Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         [unknown](1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          [unknown](0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

3. 替换失败的设备：

```
# lvconvert --repair my_vg/my_lv
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking
used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking
used and assumed devices.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
Faulty devices in my_vg/my_lv successfully replaced.
```

4. 可选：手动指定替换失败设备的物理卷：

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 使用替换检查逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg

 /dev/sdc: open failed: No such device or address
 /dev/sdc1: open failed: No such device or address
 Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
 LV              Cpy%Sync Devices
 my_lv           43.79    my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0]         /dev/sde1(1)
 [my_lv_rimage_1]         /dev/sdb1(1)
 [my_lv_rimage_2]         /dev/sdd1(1)
 [my_lv_rmeta_0]          /dev/sde1(0)
 [my_lv_rmeta_1]          /dev/sdb1(0)
 [my_lv_rmeta_2]          /dev/sdd1(0)
```

在您从卷组中删除失败的设备前，LVM 工具仍然指示 LVM 无法找到失败的设备。

6. 从卷组中删除失败的设备：

```
# vgreduce --removemissing my_vg
```

验证

1. 删除失败的设备后查看可用的物理卷：

```
# pvscan
PV /dev/sde1 VG rhel_virt-506 lvm2 [<7.00 GiB / 0 free]
PV /dev/sdb1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
```

2. 在替换失败的设备后检查逻辑卷：

```
# lvs --all --options name,copy_percent,devices my_vg
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0]         /dev/sde1(1)
 [my_lv_rimage_1]         /dev/sdb1(1)
 [my_lv_rimage_2]         /dev/sdd1(1)
 [my_lv_rmeta_0]          /dev/sde1(0)
 [my_lv_rmeta_1]          /dev/sdb1(0)
 [my_lv_rmeta_2]          /dev/sdd1(0)
```

其他资源

- lvconvert (8) 和 vgreduce (8) 手册页

17.14. 对多路径 LVM 设备进行重复的物理卷警告进行故障排除

当将 LVM 与多路径存储搭配使用时，列出卷组或者逻辑卷的 LVM 命令可能会显示如下信息：

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/dm-5 not /dev/sdd
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowerb not /dev/sde
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sddlmab not /dev/sdf
```

您可以排除这些警告来了解 LVM 显示它们的原因，或者隐藏警告信息。

17.14.1. 重复 PV 警告的根本原因

当设备映射器多路径(DM 多路径)、EMC PowerPath 或 Hitachi Dynamic Link Manager (HDLM)等多路径软件管理系统上的存储设备时，每个路径都会作为不同的 SCSI 设备注册。

然后多路径软件会创建一个映射到这些独立路径的新设备。因为每个 LUN 在 /dev 目录中有多个指向同一底层数据的设备节点，所以所有设备节点都包含相同的 LVM 元数据。

表 17.1. 不同多路径软件的设备映射示例

多路径软件	到 LUN 的 SCSI 路径	多路径设备映射到路径
DM Multipath	**/dev/sdb** 和 **/dev/sdc**	**/dev/mapper/mpath1** 或 **/dev/mapper/mpatha**
EMC PowerPath		**/dev/emcpowera**
HDLM		**/dev/sddlmab**

由于多个设备节点，LVM 工具会多次查找相同的元数据，并将其作为重复报告。

17.14.2. 重复 PV 警告的情况

LVM 在以下任一情况下显示重复的 PV 警告：

指向同一设备的单路径

输出中显示的两个设备都是指向同一设备的单一路径。

以下示例显示一个重复的 PV 警告，在该示例中重复的设备是同一设备的单一路径。

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sdd not /dev/sdf
```

如果您使用 multipath -ll 命令列出当前的 DM 多路径拓扑，您可以在相同的多路径映射中找到 /dev/sdd 和 /dev/sdf。

这些重复的信息只是警告，并不意味着 LVM 操作失败。相反，它们会提醒您 LVM 只使用其中一个设备作为物理卷，并忽略其它设备。

如果消息表示 LVM 选择了不正确的设备，或者警告是否对用户造成破坏，您可以应用过滤器。过滤器将 LVM 配置为只搜索物理卷所需的设备，并将任何基本路径留给多路径设备。因此，不再会出现警告。

多路径映射

输出中显示的两个设备都是多路径映射。

以下示例显示两个设备都有重复的 PV 警告，它们是多路径映射。重复的物理卷位于两个不同的设备中，而不是位于同一设备的两个不同路径中。

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/mapper/mpatha not
/dev/mapper/mpathc

Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowera not
/dev/emcpowerh
```

对于同一设备上的单一路径的设备来说，这种情形比重复的警告更为严重。这些警告通常意味着机器正在访问它不应该访问的设备：例如： LUN 克隆或镜像(mirror)。

除非您明确知道您应该从机器中删除哪些设备，否则这个情况可能无法恢复。红帽建议联系红帽技术支持来解决这个问题。

17.14.3. 防止重复 PV 警告的 LVM 设备过滤器示例

以下示例显示 LVM 设备过滤器，它避免了由单一逻辑单元(LUN)由多个存储路径导致的重复物理卷警告。

您可以为逻辑卷管理器(LVM)配置过滤器来检查所有设备的元数据。元数据包括上面包含根卷组的本地硬盘，以及任何多路径设备。通过拒绝多路径设备的底层路径（如 /dev/sdb、/dev/sdd），您可以避免这些重复的 PV 警告，因为 LVM 在多路径设备上一次只查找一个唯一的元数据区域。

- 要接受第一个硬盘上的第二个分区以及任何设备映射器(DM)多路径设备，并拒绝任何其他分区，请输入：

```
filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

- 要接受所有 HP SmartArray 控制器和任何 EMC PowerPath 设备，请输入：

```
filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|" ]
```

- 要接受第一个 IDE 驱动器上的任何分区以及任何多路径设备，请输入：

```
filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

17.14.4. 其他资源

其他资源

- 限制 LVM 设备可见性和用法
- LVM 设备过滤器