Red Hat Enterprise Linux 9

管理存储设备

配置和管理本地和远程存储设备

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 管理存储设备

配置和管理本地和远程存储设备

法律通告

摘要

Red Hat Enterprise Linux (RHEL)提供了几个本地和远程存储选项。使用可用的存储选项，您可以执行以下任务： 根据您的要求创建磁盘分区。使用磁盘加密来保护块设备上的数据。 创建一个独立磁盘冗余阵列(RAID)，来在多个驱动器上存储数据，并避免数据丢失。 使用 iSCSI 和 Fabrics 上的 NVMe 通过网络访问存储。 设置 Stratis 来管理物理存储设备的池。

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** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

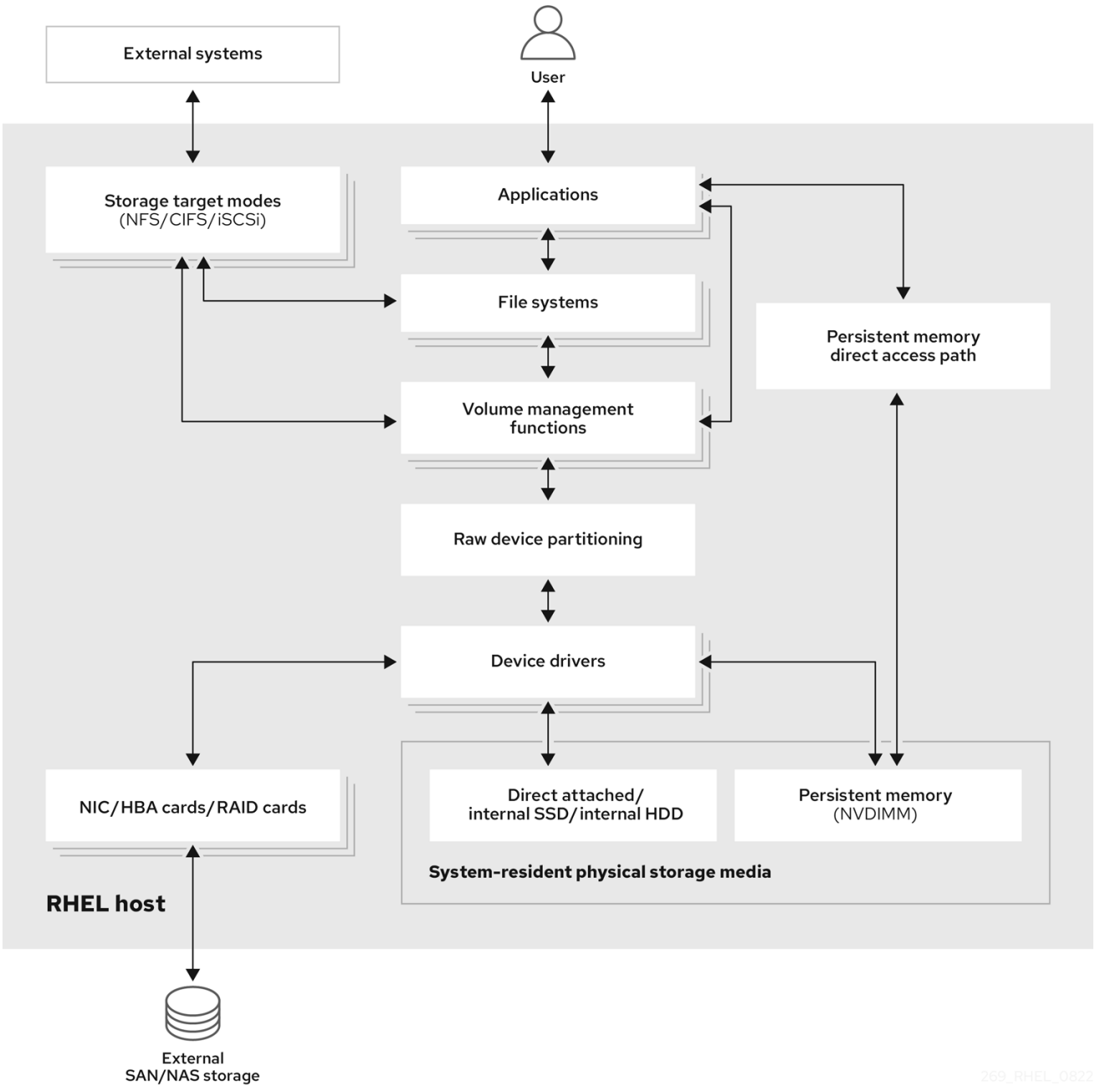
第 1 章 可用存储选项概述

Red Hat Enterprise Linux 9 中提供了几个本地、远程和基于集群的存储选项。

本地存储意味着存储设备安装在系统上，或者直接附加到系统。

使用远程存储时，设备可以通过 LAN、互联网或使用光纤通道网络进行访问。以下高级别 Red Hat Enterprise Linux 存储图描述了不同的存储选项。

图 1.1. Red Hat Enterprise Linux 存储图



1.1. 本地存储概述

Red Hat Enterprise Linux 9 提供多个本地存储选项。

基本磁盘管理

使用 **parted** 和 **fdisk**，您可以创建、修改、删除和查看磁盘分区。以下是分区布局标准：

主引导记录（MBR）

它用于基于 BIOS 的计算机。您可以创建主分区、扩展分区及逻辑分区。

GUID 分区表（GPT）

它使用全局唯一标识符(GUID)并提供唯一的磁盘和分区 GUID。

要加密分区，您可以使用 Linux Unified Key Setup-on-disk-format（LUKS）。要加密分区，选择安装过程中的选项，系统会显示输入密码短语的提示。这个密码短语用于解锁加密密钥。

存储消耗选项

Non-Volatile Dual In-line Memory Modules（NVDIMM）管理

它是内存和存储的组合。您可以在连接到您的系统的 NVDIMM 设备中启用和管理各种存储类型。

块存储管理

数据以块形式存储，每个块都有唯一的标识符。

文件存储

数据存储在本地系统中的文件级别。这些数据可使用 XFS（默认）或者 ext4 本地访问，并使用 NFS 和 SMB 通过网络访问。

逻辑卷

逻辑卷管理器（LVM）

它从物理设备中创建逻辑设备。逻辑卷(LV)是物理卷(PV)和卷组(VG)的组合。配置 LVM 包括：

- 从硬盘创建 PV。
- 从 PV 创建 VG。
- 从 VG 创建 LV，分配挂载点到 LV。

Virtual Data Optimizer（VDO）

它被用来通过重复数据删除（deduplication）、压缩和精简置备来减少数据。在 LV 下使用 VDO 可以帮助：

- 扩展 VDO 卷
- 通过多个设备生成 VDO 卷

本地文件系统

XFS

默认 RHEL 文件系统。

ext4

传统的文件系统。

Stratis

Stratis 是一个支持高级存储功能的混合用户和内核本地存储管理系统。

1.2. 远程存储概述

以下是 Red Hat Enterprise Linux 9 中可用的远程存储选项：

存储连接选项

iSCSI

RHEL 9 使用 targetcli 工具来添加、删除、查看和监控 iSCSI 存储间的连接。

光纤频道（FC）

RHEL 9 提供以下原生 Fibre Channel 驱动程序：

- **lpfc**
- **qla2xxx**
- **Zfcp**

Non-volatile Memory Express（NVMe）

允许主机软件实用程序与固态驱动器进行通信的接口。使用以下类型的光纤传输来通过光纤配置 NVMe：

- 使用 Remote Direct Memory Access(RDMA)的 NVMe over fabrics。
- 使用光纤通道(FC)的 NVMe over fabrics

设备映射器多路径（DM 多路径）

允许您将服务器节点和存储阵列间的多个 I/O 路径配置为单一设备。这些 I/O 路径是可包含独立电缆、交换机和控制器的物理 SAN 连接。

网络文件系统

- NFS
- SMB

1.3. GFS2 文件系统概述

Red Hat Global File System 2 (GFS2) 是一个 64 位对称集群文件系统，它提供了一个共享名称空间，并管理共享一个常见块设备的多个节点间的一致性。GFS2 文件系统旨在提供尽量与本地文件系统类似的功能，同时在节点间强制实施完整集群一致性。为达到此目的，节点在文件系统资源中使用集群范围的锁定方案。这个锁定方案使用 TCP/IP 等通讯协议来交换锁定信息。

在某些情况下，Linux 文件系统 API 不允许具有集群特性的 GFS2 完全透明。例如，在 GFS2 中使用 POSIX 锁定的程序应该避免使用 **GETLK** 功能，因为在集群的环境中，该进程 ID 可能用于集群中的不同节点。然而,多数情况下 GFS2 文件系统的功能和本地文件系统的功能是一样的。

Red Hat Enterprise Linux Resilient Storage Add-On 提供 GFS2，它依赖于 Red Hat Enterprise Linux High Availability 附加组件来提供 GFS2 所需的集群管理。

gfs2.ko 内核模块实现 GFS2 文件系统，并加载在 GFS2 集群节点上。

要获得最佳 GFS2 性能，请务必考虑基础设计中给出的性能注意事项。和本地文件系统一样，GFS2 依赖于页面缓存以便通过本地缓存来提高经常使用数据的性能。为了在集群中的节点间保持一致性，缓存控制由 *glock* 状态机器提供。

其他资源

- 配置 GFS2 文件系统

第 2 章 持久命名属性

您识别和管理存储设备的方式确保了系统的稳定性和可预测性。为此，Red Hat Enterprise Linux 9 使用了两个主要命名方案：传统设备名称和持久命名属性。

传统设备名称

传统设备名称由 Linux 内核根据系统中设备的物理位置来确定。例如，第一个 SATA 驱动器通常被标记为 **/dev/sda**，第二个被标记为 **/dev/sdb**，以此类推。虽然这些名称非常简单，当设备被添加或删除或者硬件配置被修改时，它们可能会发生改变。这可能会对脚本和配置文件造成挑战。另外，传统名称缺少有关设备用途或特征的描述性信息。

持久命名属性

持久命名属性基于存储设备的唯一特征，使其在系统重启后更加稳定且可预测。与传统的命名相比，实施 PNA 涉及到更详细的初始配置。PNA 的一个主要好处是其对硬件配置变化的弹性，使其非常适合维护一致的命名约定。使用 PNA 时，您可以在脚本、配置文件和管理工具中引用存储设备，而无需担心意外的名称变化。另外，PNA 通常包括重要的元数据，如设备类型或制造商信息，增强了有效设备识别和管理的描述。

2.1. 用于识别文件系统和块设备的持久属性

在 Red Hat Enterprise Linux 9 存储中，持久命名属性(PNA)是系统重启、硬件变化或其他事件后为存储设备提供一致且可靠的命名机制。这些属性用于一致地识别存储设备，即使存储设备被添加、被删除或被重新配置。

PNA 用于识别文件系统和块设备，但它们的用途不同：

用于识别文件系统的持久属性

- 通用唯一标识符(UUID)
  UUID 主要用于唯一识别存储设备上的文件系统。每个文件系统都有自己的 UUID，即使文件系统被卸载、被重新挂载或者设备被分离和重新附加，此标识符也会保持不变。

- 标签
  标签是用户为文件系统分配的名称。虽然它们可以用来识别和引用文件系统，但它们并不像 UUID 那样标准化。标签通常用作 UUID 的替代方法，用来在配置文件中指定文件系统。

  当您为文件系统分配标签时，它会成为文件系统元数据的一部分。标签会随文件系统一起存在，即使在不同的挂载点或不同的系统上挂载文件系统。

用于识别块设备的持久属性

- 通用唯一标识符(UUID)
  UUID 可用于识别存储块设备。当存储设备被格式化或者在其上创建文件系统时，通常会为设备本身分配一个 UUID。此 UUID 被嵌入在文件系统元数据或分区表中，用作持久设备命名的引用。它允许您唯一标识块设备，即使您更改了文件系统或重新格式化了它。

- 全球标识符(WWID)
  WWID 是与存储块设备关联的全局唯一标识符。它们通常用于光纤通道存储区域网络(SAN)来识别将服务器连接到 SAN 存储设备的主机总线适配器(HBA)或网络接口。WWID 确保服务器和 SAN 存储设备之间的一致通信，并帮助管理存储设备的冗余路径。

- 序列号
  序列号是制造商分配给每个存储块设备的唯一标识符。它可用于区分存储设备，并可与其他属性（如 UUID 或 WWID）结合使用，用于设备管理。

2.2. UDEV 设备命名规则

用户空间设备管理器(**udev**)子系统允许您为设备分配持久名称定义规则。这些规则存储在 **/etc/udev/rules.d/** 目录中带有 **.rules** 扩展名的文件中。这些规则的目的是确保存储设备的识别一致且可预测，即使系统重启和配置发生了改变也是如此。

udev 规则使用键-值对以人类可读的格式编写。当设备被检测或初始化时，**udev** 会根据它们的定义顺序来顺序评估这些规则。第一个匹配规则被应用到设备，确定其名称以及其如何在系统中被识别。

对于存储设备，**udev** 规则在 **/dev/disk/** 目录中创建符号链接。这些符号链接为存储设备提供用户友好的别名，从而使引用和管理这些设备更为方便。

您可以创建自定义 **udev** 规则，以指定如何根据序列号、WWN (World Wide Name)标识符或其他特定于设备的特征等各种属性来命名设备。通过定义特定的命名规则，您可以精确控制设备如何在系统中被识别。

udev 规则有两个主要位置：

- **/lib/udev/rules.d/** 目录包含 **udev** 软件包附带的默认规则。
- **/etc/udev/rules.d** 目录用于自定义 **udev** 规则。

虽然 **udev** 规则非常灵活，但了解 **udev** 限制很重要：

- 可访问性计时：在 **udev** 查询时可能无法访问一些存储设备。
- 基于事件的处理：内核可以随时发送 **udev** 事件，如果设备无法访问，可能会触发规则处理和链接删除。
- 处理延迟：事件生成和处理之间可能会有延迟，特别是有多个设备时，从而导致内核检测和链路可用性之间的滞后。
- 设备可访问性：**udev** 规则调用的外部程序（如 **blkid** ）可能会短暂打开该设备，使其他任务暂时无法访问它。
- 链接更新：由 **/dev/disk/** 中的 **udev** 管理的设备名称可能会在主版本间有所变化，需要链接更新。

其它资源

- **udev** 手册页

第 3 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。

3.1. 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

3.2. 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，以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

3.3. 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，以立即并永久挂载 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
```

其它资源

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

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

3.4. 使用 STORAGE RHEL 系统角色管理逻辑卷

示例 Ansible playbook 应用 **storage** 角色在卷组中创建 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
```

其它资源

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

3.5. 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

3.6. 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** 角色来创建并挂载 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

3.7. 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** 角色来创建并挂载 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

3.8. 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

3.9. 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

3.10. 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

3.11. 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设置 Ansible playbook，以配置带有 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

3.12. 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设置 Ansible playbook，以配置带有 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

3.13. 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

3.14. 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

3.15. 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 4 章 磁盘分区

要将磁盘分成一个或多个逻辑区域，请使用磁盘分区实用程序。这可以对每个分区进行独立的管理。

4.1. 分区概述

硬盘在分区表中保存每个磁盘分区的位置和大小信息。使用分区表中的信息，操作系统会将每个分区视为一个逻辑磁盘。磁盘分区的一些优点包括：

- 减少对物理卷的管理
- 保证有足够的备份
- 提供有效的磁盘管理

其他资源

- 在 LUN 中使用分区（直接或通过 LVM）有哪些优点和缺点？

4.2. 修改磁盘分区前的注意事项

在创建、删除或调整任何磁盘分区的大小之前，请考虑以下方面：

在设备中，分区表的类型决定单个分区的最大数目和大小。

最大分区数：

- 在使用**主引导记录(MBR)** 分区表格式化的设备上，您可以有：
  - 最多四个主分区。
  - 最多三个主分区，再加上一个扩展分区
    - 扩展分区中可以有多个逻辑分区
- 在使用 **GUID 分区表(GPT)** 格式化的设备中，您可以：
  - 使用 **parted** 实用程序，最多 128 个分区。
    - 虽然 GPT 规格可以通过增加分区表的保留大小来带有更多分区，当 parted 会限制 128 分区所需的区域。

最大分区大小：

- 在使用**主引导记录(MBR)** 分区表格式化的设备上:
  - 在使用 512b 扇区驱动器时，最大大小为 2 TiB。
  - 在使用 4k 扇区驱动器时，最大大小为 16 TiB。
- 在使用 **GUID 分区表(GPT)** 格式化的设备中：
  - 使用 512b 扇区驱动器时，最大大小为 8 ZiB。
  - 使用 4k 扇区驱动器时，最大大小为 64 ZiB。

通过使用 **parted** 工具，您可以使用多个不同的后缀指定分区大小：

- **MiB、GiB 或 TiB**
  - 大小为 2 的指数代表。
  - 分区的起点与根据大小指定的扇区一致。
  - 结束点与指定大小减 1 扇区一致。
- **MB、GB 或 TB ：**
  - 以 10 的指数表示容量。
  - 起点和结束点在指定大小单元的一半内一致。例如，如果后缀是 MB 时为 ±500KB。

注意

本节不涵盖 DASD 分区表，它特定于 IBM Z 构架。

其他资源

- 在 IBM Z 中配置 Linux 实例
- 您需要了解的 DASD 的信息

4.3. 分区表类型比较

要在设备中启用分区，使用不同类型的分区表格式化块设备。下表比较您可以在块设备中创建的不同类型的分区表的属性。

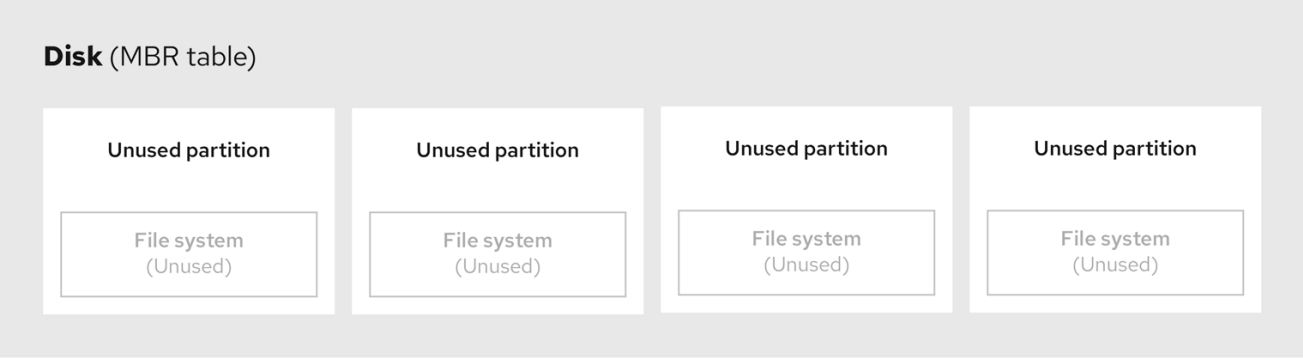
表 4.1. 分区表类型

分区表	最多分区数	最大分区大小
主引导记录（MBR）	4 个主分区，或 3 个主分区和 1 个扩展分区（带有 12 个逻辑分区）	2TiB
GUID 分区表（GPT）	128	8ZiB

4.4. MBR 磁盘分区

分区表保存在磁盘的最开始，任何文件系统或用户数据都会保存在它的后面。为了更明确的显示，下图中的不同分区表分开显示。

图 4.1. 有 MBR 分区表的磁盘



如上图所示，分区表被分为四个未使用主分区的四个部分。主分区是在硬盘中仅包含一个逻辑驱动器（或部分）的分区。每个逻辑驱动器都有定义单个分区所需的信息，这意味着分区表可以定义不超过四个主分区。

每个分区表条目都包含分区的重要特性：

- 磁盘上分区启动和结束的点
- 分区的状态，因为只有一个分区可以被标记为 **活跃**分区
- 分区的类型

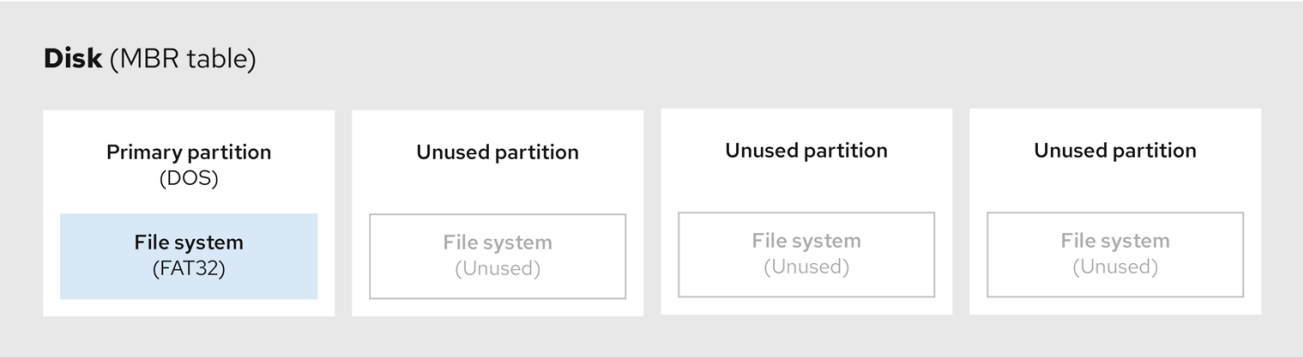
起点和结束点定义了磁盘上分区的大小和位置。有些操作系统引导装载程序使用 **active** 标记。这意味着，在标记为"活跃"的分区中的操作系统被引导。

类型由一个数字代表，用于识别分区预定的使用情况。有些操作系统使用分区类型来：

- 表示特定的文件系统类型
- 将分区标记为与特定操作系统关联的
- 指明分区包含可引导操作系统

下图显示了含有单一分区的驱动器示例：在这个示例中，第一个分区被标记为 **DOS** 分区类型：

图 4.2. 只有一个分区的磁盘



其他资源

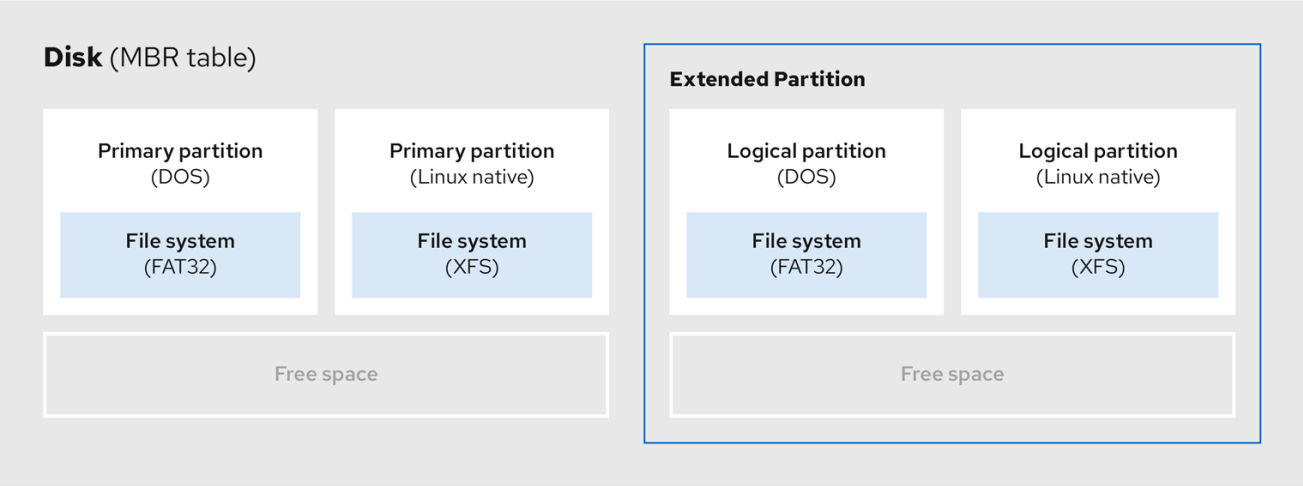
- MBR 分区类型

4.5. 扩展 MBR 分区

要创建额外分区（如果需要），请将类型设置为 **extended**。

扩展分区类似于磁盘驱动器。它有自己的分区表，指向一个或多个逻辑分区，完全包含在扩展分区中。下图显示了一个磁盘驱动器，它有两个主分区和一个包含两个逻辑分区（以及一些未分区的可用空间）的扩展分区：

图 4.3. 带两个主分区和扩展的 MBR 分区的磁盘



最多只能有 4 个主分区和扩展分区，但逻辑分区的数量没有固定限制。作为 Linux 中的限制访问分区，单一磁盘驱动器允许最多 15 个逻辑分区。

4.6. MBR 分区类型

下表显示了一些最常用的 MBR 分区类型和用于代表它们的十六进制数字。

表 4.2. MBR 分区类型

MBR 分区类型	值	MBR 分区类型	值
空	00	Novell Netware 386	65
DOS 12-bit FAT	01	PIC/IX	75
XENIX root	02	Old MINIX	80
XENIX usr	03	Linux/MINUX	81
DOS 16-bit ⇐32M	04	Linux swap	82
Extended	05	Linux native	83

DOS 16-bit >=32	06	Linux extended	85
OS/2 HPFS	07	Amoeba	93
AIX	08	Amoeba BBT	94
AIX bootable	09	BSD/386	a5
OS/2 Boot Manager	0a	OpenBSD	a6
Win95 FAT32	0b	NEXTSTEP	a7
Win95 FAT32(LBA)	0c	BSDI fs	b7
Win95 FAT16(LBA)	0e	BSDI swap	b8
Win95 Extended (LBA)	0f	Syrinx	c7
Venix 80286	40	CP/M	db
Novell	51	DOS access	e1
PRep Boot	41	DOS R/O	e3
GNU HURD	63	DOS secondary	f2
Novell Netware 286	64	BBT	ff

4.7. GUID 分区表

GUID 分区表(GPT)是基于全局唯一标识符(GUID)的分区方案。

GPT 处理 Mater Boot Record (MBR)分区表的限制。MBR 分区表无法处理大于 2 TiB 的存储，相当于大约 2.2 TB。相反，GPT 支持容量较大的硬盘。使用 512b 扇区驱动器时，最大可寻址磁盘大小为 8 ZiB，在使用 4096b 扇区驱动器时，为 64 ZiB。另外，默认情况下，GPT 支持创建最多 128 个主分区。通过向分区表分配更多空间来扩展主分区的最大数量。

注意

GPT 有基于 GUID 的分区类型。某些分区需要特定的 GUID。例如，可扩展固件接口(EFI)引导装载程序的系统分区需要 GUID **C12A7328-F81F-11D2-BA4B-00A0C93EC93B**。

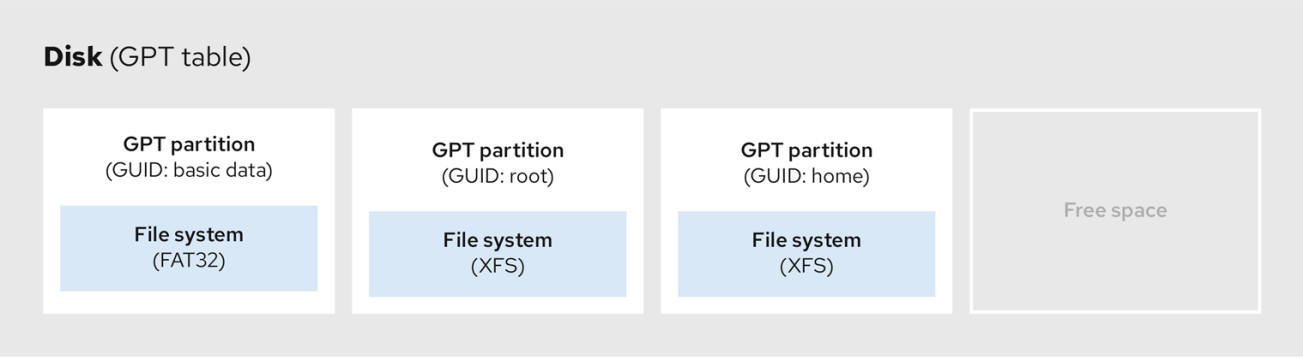
GPT 磁盘使用逻辑块寻址(LBA)和分区布局，如下所示：

- 为了向后兼容 MBR 磁盘，系统为 MBR 数据保留 GPT 的第一个扇区(LBA 0)，并应用名称"protective MBR"。
- 主 GPT

- 标头从设备的第二个逻辑块(LBA 1)开始。标头中包含磁盘 GUID、主分区表的位置、从属 GPT 标头的位置、自身的 CRC32 checksum 以及主分区表。它还指定表上的分区条目数目。
- 默认情况下，主 GPT 包含 128 个分区条目。每个分区的条目大小为 128 字节、分区类型 GUID 和唯一的分区 GUID。

- 二级 GPT
  - 对于恢复，当主分区表格损坏时，备份表非常有用。
  - 磁盘的最后一个逻辑扇区包含二级 GPT 标头并恢复 GPT 信息（如果主标头损坏）。
  - 它包含：
    - 磁盘 GUID
    - 二级分区表和主 GPT 标头的位置
    - 自身的 CRC32 checksums
    - 二级分区表
    - 可能的分区条目数

图 4.4. 带有 GUID 分区表的磁盘



重要

对于在 GPT 磁盘上成功安装引导装载程序，必须存在 BIOS 引导分区。只有当磁盘已包含 BIOS 引导分区时，才可以重复使用。这包括 **Anaconda** 安装程序初始化的磁盘。

4.8. 分区类型

管理分区类型的方法有多种：

- **fdisk** 工具通过指定十六进制代码来支持所有的分区类型。
- **systemd-gpt-auto-generator** （单位生成器程序程序）使用分区类型自动识别和挂载设备。
- **parted** 实用程序使用 *flags* 对分区类型进行映射。**parted** 工具只处理某些分区类型，如 LVM、swap 或 RAID。
  parted 实用程序支持设置以下标记：

- **boot**
- **root**
- **swap**
- **hidden**
- **raid**
- **lvm**
- **lba**
- **legacy_boot**
- **irst**
- **esp**
- **palo**

在带有 **parted** 3.5 的 Red Hat Enterprise Linux 9 中，您可以使用额外的标记 **chromeos_kernel** 和 **bls_boot**。

parted 工具在创建分区时接受文件系统类型参数。请参阅 使用 parted 创建分区

对于所需条件的列表。使用值来：

- 在 MBR 中设置分区标记。
- 在 GPT 中设定分区 UUID 类型。例如，**swap**、**fat** 或 **hfs** 文件系统类型设置不同的 GUID。默认值为 Linux Data GUID。

参数不会修改分区中的文件系统。它只会区分受支持的标志和 GUID。

支持以下文件系统类型：

- **xfs**
- **ext2**
- **ext3**
- **ext4**
- **fat16**
- **fat32**
- **hfs**
- **hfs+**
- **linux-swap**
- **ntfs**

- **reiserfs**

4.9. 分区命名方案

Red Hat Enterprise Linux 使用基于文件的命名方案，其文件名采用 **/dev/*xxyN*** 的形式。

设备和分区名称由以下结构组成：

/dev/

包含所有设备文件的目录的名称。硬盘包含分区，因此代表所有可能分区的文件位于 **/dev** 中。

xx

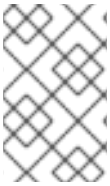
分区名的前两个字母标明包含该分区的设备类型。

y

这个字母标明包含分区的特定设备。例如： **/dev/sda** 用于第一个硬盘，第二个硬盘为 **/dev/sdb**。您可以在超过 26 个驱动器的系统中使用更多字母，例如 **/dev/sdaa1**。

N

最后的字母代表分区的数字。前四个（主或扩展）分区编号为 **1** 到 **4**。逻辑分区从 **5** 开始。例如，**/dev/sda3** 是第一个硬盘上的第三个主分区或扩展分区，**/dev/sdb6** 是第二个硬盘上的第二个逻辑分区。驱动器分区编号只适用于 MBR 分区表。请注意，***N*** 并不总是意味着分区。



注意

即使 Red Hat Enterprise Linux 可以识别和引用 *所有*类型的磁盘分区，它可能无法读取文件系统，因此无法访问分区类型中保存的数据。然而，在很多情况下，成功访问专用于另一个操作系统的分区中的数据是可能的。

4.10. 挂载点和磁盘分区

在 Red Hat Enterprise Linux 中，每个分区都被用来成为支持一组文件和目录所必需的存储的一部分。挂载分区会导致该分区的存储可用，从指定目录开始，称为 *挂载点*。

例如，如果分区 **/dev/sda5** 挂载在 **/usr/** 上，这意味着 **/usr/** 下的所有文件和目录都在 **/dev/sda5** 上。文件 **/usr/share/doc/FAQ/txt/Linux-FAQ** 位于 **/dev/sda5** 中，而文件 **/etc/gdm/custom.conf** 不在。

继续这个示例，也可以将 **/usr/** 下的一个或多个目录作为其他分区的挂载点。例如：**/usr/local/man/whatis** 位于 **/dev/sda7** 上，而不是位于 **/dev/sda5** 上，如果 **/usr/local** 包含挂载的 **/dev/sda7** 分区。

第 5 章 分区入门

使用磁盘分区将磁盘分成一个或多个逻辑区域，这些区域可以单独在每个分区上工作。硬盘在分区表中保存每个磁盘分区的位置和大小信息。使用表，每个分区然后显示为操作系统的逻辑磁盘。然后您可以在这些独立磁盘中进行读取和写入。

有关在块设备中使用分区的优点和缺点的概述，请参阅在 LUN 上使用分区（可以在两者间直接或通过 LVM）使用哪些优点和缺陷？

5.1. 使用 PARTED 在磁盘中创建分区表

使用 **parted** 实用程序更轻松地使用分区表格式化块设备。

警告

使用分区表格式化块设备会删除该设备中所有存储的数据。

流程

1. 启动交互式 **parted** shell：

```
# parted block-device
```

2. 确定该设备中是否已有一个分区表：

```
# (parted) print
```

如果设备已经包含分区，则后续步骤中将删除它们。

3. 创建新分区表：

```
# (parted) mklabel table-type
```

- 使用预期的分区表类型替换 *table-type* ：
  - 用于的 MBR 的 **msdos**
  - 用于 GPT 的 **gpt**

例 5.1. 创建 GUID 分区表(GPT)表

要在磁盘上创建 GPT 表，请使用：

```
# (parted) mklabel gpt
```

在输入以下命令后，这些更改将开始应用。

4. 查看分区表以确认其已创建：

```
# (parted) print
```

5. 退出 **parted** shell:

```
# (parted) quit
```

其它资源

- **parted(8)** 手册页。

5.2. 查看使用 PARTED 的分区表

显示块设备的分区表，以查看分区布局和单个分区的详情。您可以使用 **parted** 实用程序查看块设备上的分区表。

流程

1. 启动 **parted** 工具。例如：以下输出列出了设备 **/dev/sda** ：

```
# parted /dev/sda
```

2. 查看分区表：

```
# (parted) print

Model: ATA SAMSUNG MZNLN256 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  269MB   268MB   primary   xfs          boot
 2      269MB   34.6GB  34.4GB  primary
 3      34.6GB  45.4GB  10.7GB  primary
 4      45.4GB  256GB   211GB   extended
 5      45.4GB  256GB   211GB   logical
```

3. 可选：切换到您要检查的设备：

```
# (parted) select block-device
```

有关打印命令输出的详细描述，请查看以下信息：

模型：ATA SAMSUNG MZNLN256(scsi)

磁盘类型、制造商、型号号和接口。

磁盘 /dev/sda: 256GB

块设备的文件路径和存储容量。

分区表：msdos

磁盘标签类型。

Number

分区号。例如，次号 1 的分区对应于 **/dev/sda1**。

Start 和 End

在分区启动和结束的设备中的位置。

Type

有效类型为 metadata、free、primary、extended 或 logical。

File system

文件系统类型。如果设备的 **File system** 字段未显示值，这意味着其文件系统类型为未知。**parted** 工具无法识别加密设备上的文件系统。

标记

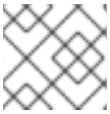
列出为分区设置的标记。可用的标志有 **boot**、**root**、**swap**、**hidden**、**raid**、**lvm** 或 **lba**。

其它资源

- **parted(8)** 手册页。

5.3. 使用 PARTED 创建分区

作为系统管理员，您可以使用 **parted** 实用程序在磁盘上创建新分区。



注意

所需分区是 **swap**、**/boot/** 和 **/(root)**。

先决条件

- 磁盘上的分区表。
- 如果要创建的分区大于 2TiB，使用 **GUID 分区表(GPT)** 格式化磁盘。

流程

1. 启动 **parted** 工具：

```
# parted block-device
```

2. 查看当前的分区表来确定是否有足够空闲空间：

```
# (parted) print
```

   - 如果分区没有足够的可用空间，则调整分区大小。
   - 从分区表中决定：
     - 新分区的开始和结束点。
     - 在 MBR 上，应该是什么分区类型。

3. 创建新分区：

```
# (parted) mkpart part-type name fs-type start end
```

- 将 *part-type* 替换为 **primary**, **logical**, 或 **extended**。这只适用于 MBR 分区表。
- 使用任意分区名称替换 *name*。对于 GPT 分区表，这是必需的。
- 将 *fs-type* 替换为 **xfs**, **ext2**, **ext3**, **ext4**, **fat16**, **fat32**, **hfs**, **hfs+**, **linux-swap**, **ntfs**, 或 **reiserfs**。*fs-type* 参数是可选的。请注意，**parted** 实用程序不会在分区中创建文件系统。
- 使用从磁盘开头计算分区开始和结束点的大小替换 *start* 和 *end*。您可以使用大小后缀，如 **512MiB**、**20GiB** 或 **1.5TiB**。默认的大小是 MB。

> **例 5.2. 创建小的主分区**
>
> 要从 1024MiB 创建主分区，直到 MBR 表中的 2048MiB，请使用：
>
> ```
> # (parted) mkpart primary 1024MiB 2048MiB
> ```

在输入以下命令后，这些更改开始应用。

4. 查看分区表以确认创建的分区位于分区表中，并具有正确的分区类型、文件系统类型和大小：

```
# (parted) print
```

5. 退出 **parted** shell:

```
# (parted) quit
```

6. 注册新设备节点：

```
# udevadm settle
```

7. 验证内核是否识别了新的分区：

```
# cat /proc/partitions
```

其它资源

- **parted(8)** 手册页。
- 使用 parted 在磁盘中创建分区表.
- 使用 parted 重新定义分区大小

5.4. 使用 FDISK 设置分区类型

您可以使用 **fdisk** 实用程序设置分区类型或标志。

先决条件

- 磁盘上的一个分区。

流程

1. 启动交互式 **fdisk** shell：

```
# fdisk block-device
```

2. 查看当前的分区表以确定副分区号码：

```
Command (m for help): print
```

您可以在 **Type** 列中看到当前的分区类型，在 **Id** 列中看到相应的类型 ID 。

3. 输入分区类型命令并使用它的副号码选择分区：

```
Command (m for help): type
Partition number (1,2,3 default 3): 2
```

4. 可选：查看十六进制代码列表：

```
Hex code (type L to list all codes): L
```

5. 设置分区类型：

```
Hex code (type L to list all codes): 8e
```

6. 写入更改并退出 **fdisk** shell：

```
Command (m for help): write
The partition table has been altered.
Syncing disks.
```

7. 验证您的更改：

```
# fdisk --list block-device
```

5.5. 使用 PARTED 重新定义分区大小

使用 **parted** 工具，扩展分区以使用未使用的磁盘空间，或者缩小分区以将其容量用于不同目的。

先决条件

- 在缩小分区前备份数据。
- 如果要创建的分区大于 2TiB，使用 **GUID 分区表(GPT)** 格式化磁盘。
- 如果您想缩小分区，首先缩小文件系统，使其不大于重新定义大小的分区。

注意

XFS 不支持缩小。

流程

1. 启动 **parted** 工具：

```
# parted block-device
```

2. 查看当前的分区表：

```
# (parted) print
```

从分区表中决定：

- 分区的副号码。
- 调整大小后现有分区的位置和新结束点。

3. 重新定义分区大小：

```
# (parted) resizepart 1 2GiB
```

- 使用您要重新定义分区的副号码替换 *1*。
- 将 *2* 替换为确定重新定义重新定义分区大小的新结束点的大小，从磁盘开始计算。您可以使用大小后缀，如 **512MiB**、**20GiB** 或 **1.5TiB**。默认的大小是 MB。

4. 查看分区表以确认调整了大小的分区位于分区表中，且大小正确：

```
# (parted) print
```

5. 退出 **parted** shell:

```
# (parted) quit
```

6. 验证内核是否注册了新分区：

```
# cat /proc/partitions
```

7. 可选：如果您扩展分区，还要扩展它的文件系统。

其他资源

- **parted(8)** 手册页。
- 使用 parted 在磁盘中创建分区表
- 重新定义 ext4 文件系统大小
- 增加 XFS 文件系统的大小

5.6. 使用 PARTED 删除分区

使用 **parted** 实用程序，您可以删除磁盘分区以释放磁盘空间。

警告

删除分区将删除保存在分区中的所有数据。

流程

1. 启动交互式 **parted** shell：

```
# parted block-device
```

   - 使用您要删除分区的设备的路径替换 *block-device* ：例如 /**dev**/**sda**。

2. 查看当前的分区表以确定要删除的分区的次号：

```
(parted) print
```

3. 删除分区：

```
(parted) rm minor-number
```

   - 使用您要删除的分区的副号码替换 *minor-number*。

   输入此命令后，这些更改会立即应用。

4. 验证您是否已从分区表中删除了分区：

```
(parted) print
```

5. 退出 **parted** shell:

```
(parted) quit
```

6. 验证内核是否注册分区是否已删除：

```
# cat /proc/partitions
```

7. 如果分区存在，从 /**etc**/**fstab** 文件中删除分区。找到声明删除的分区的行，并将其从文件中删除。

8. 重新生成挂载单元，以便您的系统注册新的 /**etc**/**fstab** 配置：

```
# systemctl daemon-reload
```

9. 如果您删除了交换分区或删除 LVM 部分，请从内核命令行中删除对分区的所有引用：

   a. 列出活跃内核选项并查看是否有选项引用删除的分区：

```
# grubby --info=ALL
```

b. 删除引用已删除分区的内核选项：

```
# grubby --update-kernel=ALL --remove-args="option"
```

10. 要在早期引导系统中注册更改，请重建 **initramfs** 文件系统：

```
# dracut --force --verbose
```

其它资源

- **parted(8)** 手册页

第 6 章 重新分区磁盘策略

重新分区磁盘的方法有多种。包括：

- 有可用的未分区的空闲空间。
- 一个未使用的分区可用。
- 在一个活跃使用的分区中的空闲空间是可用。

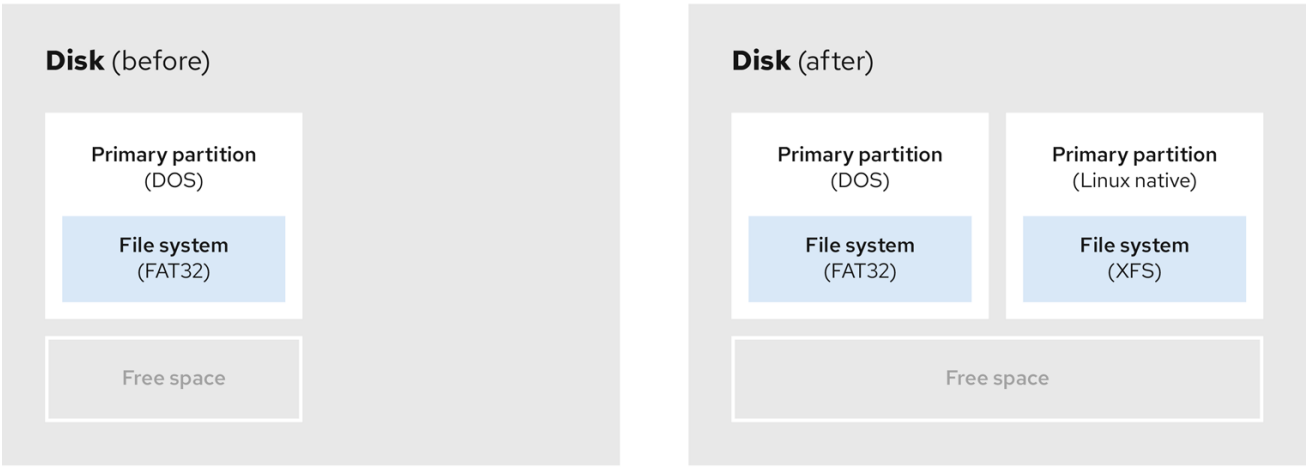
注意

为清晰起见，以下示例没有反映在实际安装 Red Hat Enterprise Linux 时的确切分区布局。

6.1. 使用未分区的空闲空间

已定义且没有跨越整个硬盘的分区，保留不属于任何定义的分区的未分配空间。下图显示了这种情况。

图 6.1. 有未分区的可用空间的磁盘



第一个图代表一个带有一个主分区的磁盘以及带有未分配空间的未定义分区。第二个图代表有两个已分配空间的分区的磁盘。

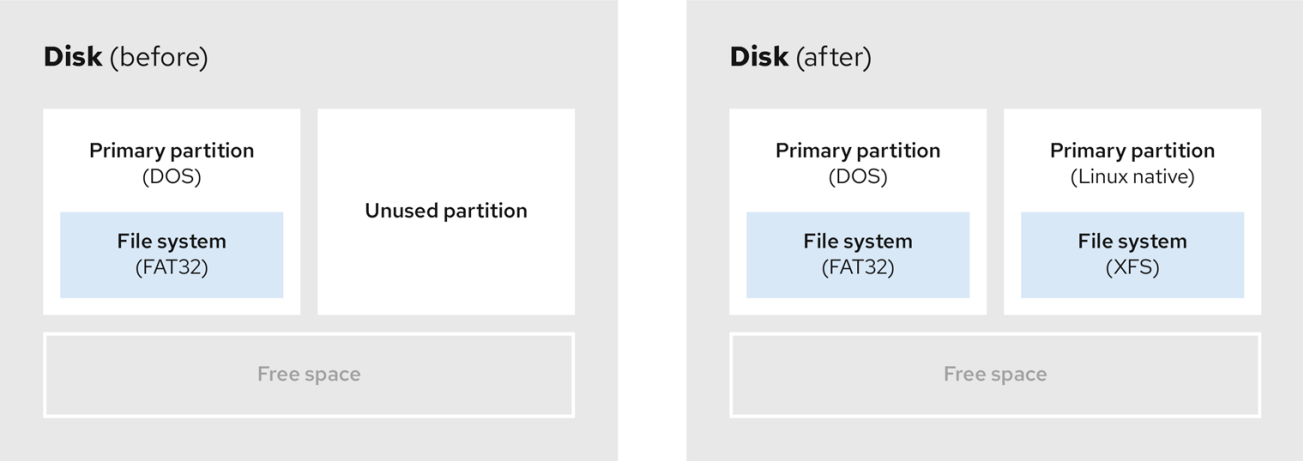
未使用的硬盘也属于这一类别。唯一的区别是，*所有* 空间并不是任何定义的分区的一部分。

在新磁盘上，您可以从未使用的空间创建必要的分区。大部分预安装的操作系统都被配置为占据磁盘驱动器上所有可用空间。

6.2. 使用未使用分区中的空间

在以下示例中，第一个图代表有未使用分区的磁盘。第二个图代表为 Linux 分配未使用的分区。

图 6.2. 有未使用分区的磁盘



要使用分配给未使用分区的空间，请删除分区，然后创建适当的 Linux 分区。或者，在安装过程中，删除未使用的分区并手动创建新分区。

6.3. 使用活跃分区中的空闲空间

因为已经使用的一个活跃分区包含所需的可用空间，所以此过程可能很难管理。在大多数情况下，预安装软件的计算机的硬盘包含一个大型分区，存放操作系统和数据。

警告

如果要在活跃分区中使用操作系统(OS)，您必须重新安装操作系统。请注意，一些计算机可能会包含预安装的软件，不没有提供用于重新安装操作系统的安装介质。在销毁原始分区和操作系统安装前，请检查是否适用于您的操作系统。

要选择使用可用空间，您可以使用破坏性或非破坏性重新分区的方法。

6.3.1. 破坏性重新分区

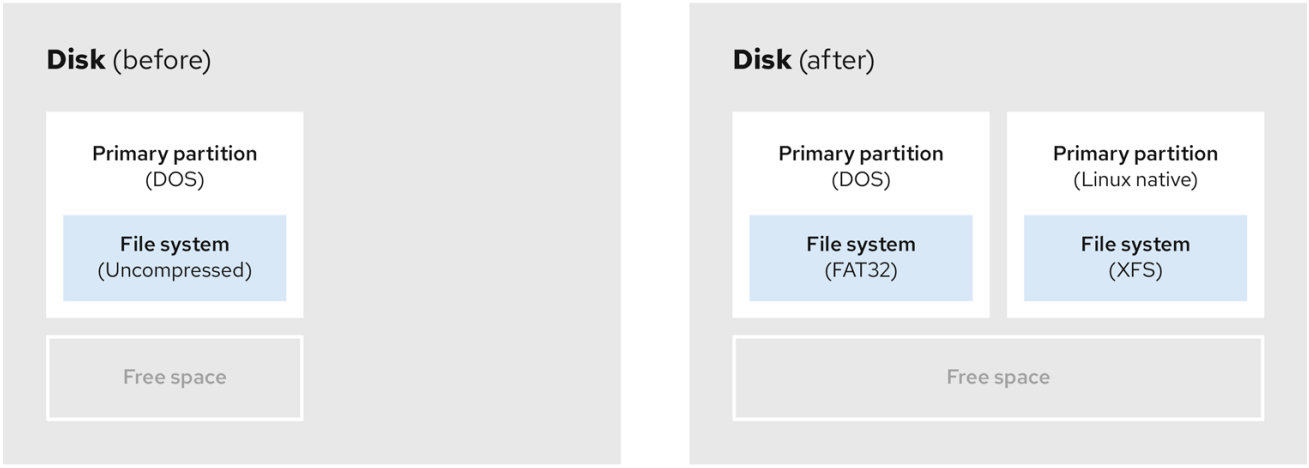
破坏性重新分区破坏硬盘中的分区并创建几个较小的分区。从原始分区备份所有需要的数据，因为此方法会删除完整内容。

为现有操作系统创建一个较小的分区后，您可以：

- 重新安装软件。
- 恢复您的数据。
- 开始您的 Red Hat Enterprise Linux 安装。

下图显示了使用破坏性重新分区方法的简化形式。

图 6.3. 在磁盘上进行破坏性重新分区动作



警告

这个方法会删除之前存储在原始分区中的所有数据。

6.3.2. 非破坏性重新分区

非破坏性重新分区分区大小，没有任何数据丢失。这个方法是可靠的，但在大型驱动器上需要更长的处理时间。

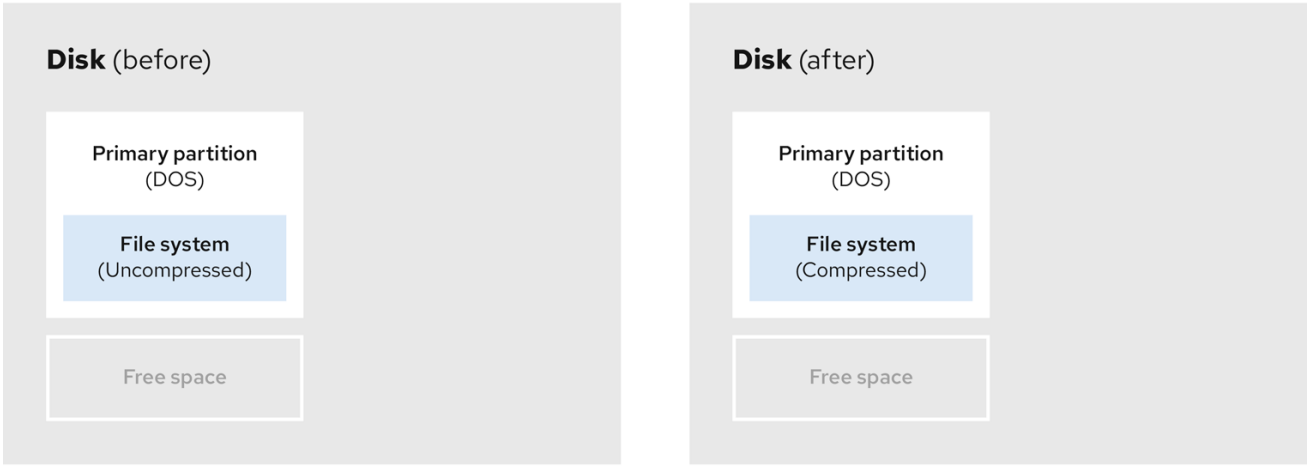
以下是帮助启动非破坏性重新分区的方法列表。

- 压缩现有数据

无法更改部分数据的存储位置。这可以防止分区大小到所需大小，最终导致破坏性重新分区过程。压缩现有分区中的数据可帮助您调整分区的大小。它还有助于最大程度提高可用空间。

下图显示了此过程的简化形式。

图 6.4. 磁盘中的数据压缩



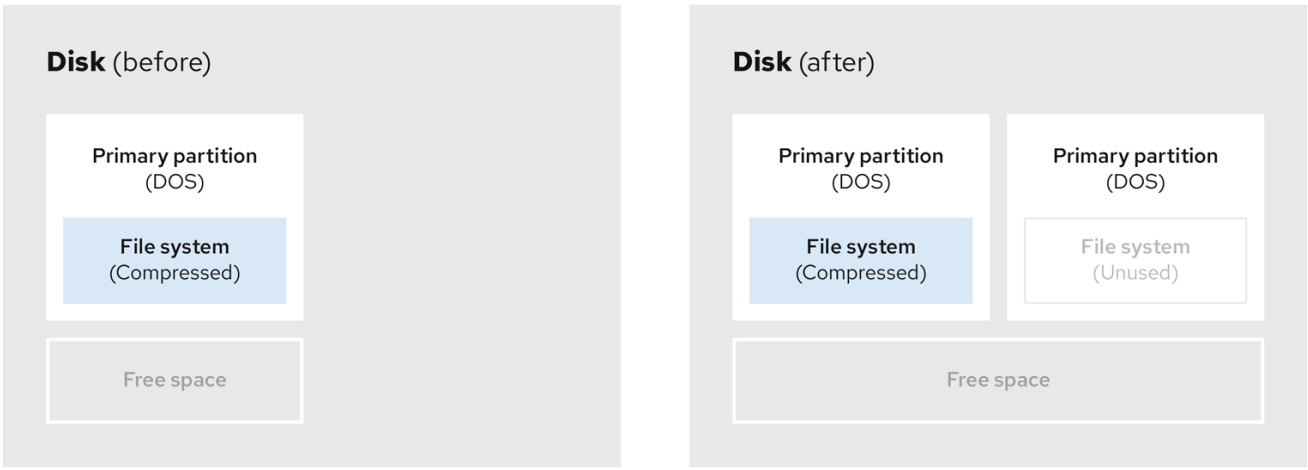
为了避免任何可能的数据丢失，请在继续压缩过程前创建备份。

- 重新划分现存分区的大小

通过重新定义已存在的分区的大小，您可以释放更多空间。根据您的软件重新定义大小，结果可能会有所不同。在大多数情况下，您可以创建同一类型的新未格式化的分区，与原始分区不同。

调整大小后采取的步骤可以取决于您所使用的软件。在以下示例中，最佳实践是删除新的 DOS(Disk Operating System)分区，而是创建一个 Linux 分区。在启动重新定义大小过程前，验证最适合您的磁盘。

图 6.5. 在磁盘上重新定义分区大小

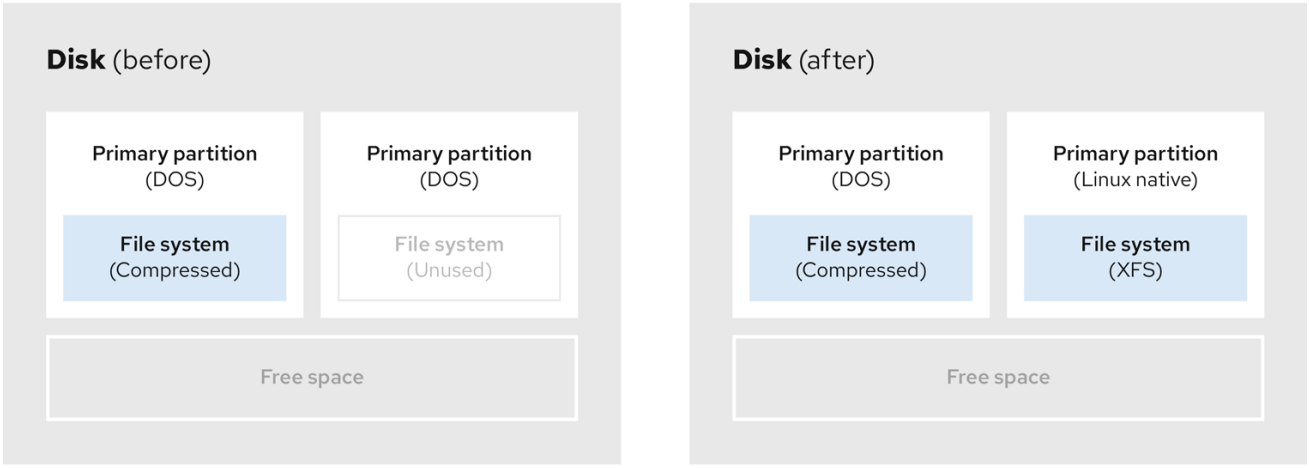


- 可选：创建新分区

一些可以实现重新调整大小的软件会支持基于 Linux 的系统。在这种情况下，在调整大小后不需要删除新创建的分区。之后创建新分区取决于您使用的软件。

下图显示了创建新分区前和之后的磁盘状态。

图 6.6. 带有最终分区配置的磁盘



第 7 章 配置 ISCSI 目标

Red Hat Enterprise Linux 使用 **targetcli** shell 作为命令行界面执行以下操作：

- 添加、删除、查看和监控 iSCSI 存储间的连接来利用 iSCSI 硬件。
- 将由文件、卷、本地 SCSI 设备或者 RAM 磁盘支持的本地存储资源导出到远程系统。

targetcli 工具带有一个基于树形的布局，包括内置的 tab 完成 、自动完成支持和在线文档。

7.1. 安装 TARGETCLI

安装 **targetcli** 工具来添加、监控和删除 iSCSI 存储间的连接。

步骤

1. 安装 **targetcli** 工具：

```
# dnf install targetcli
```

2. 启动目标服务：

```
# systemctl start target
```

3. 将目标配置为在引导时启动：

```
# systemctl enable target
```

4. 在防火墙中打开端口 **3260**，并重新载入防火墙配置：

```
# firewall-cmd --permanent --add-port=3260/tcp
Success

# firewall-cmd --reload
Success
```

验证

- 查看 **targetcli** 布局：

```
# targetcli
/> ls
o- /.....[...]
  o- backstores.....[...]
  | o- block.....[Storage Objects: 0]
  | o- fileio.....[Storage Objects: 0]
  | o- pscsi.....[Storage Objects: 0]
  | o- ramdisk.....[Storage Objects: 0]
  o- iscsi.....[Targets: 0]
  o- loopback.....[Targets: 0]
```

其他资源

- **targetcli(8)** 手册页

7.2. 创建 ISCSI 目标

创建 iSCSI 目标可让客户端的 iSCSI 发起程序访问服务器中的存储设备。目标和发起方都有唯一的标识名称。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。

步骤

1. 进入 iSCSI 目录：

```
/> iscsi/
```

注意

cd 命令用于更改目录以及列出要移动到的路径。

2. 使用以下选项之一创建 iSCSI 对象：

   a. 使用默认目标名称创建 iSCSI 对象：

```
/iscsi> create

Created target
iqn.2003-01.org.linux-iscsi.hostname.x8664:sn.78b473f296ff
Created TPG1
```

   b. 使用特定名称创建 iSCSI 对象：

```
/iscsi> create iqn.2006-04.com.example:444

Created target iqn.2006-04.com.example:444
Created TPG1
Here iqn.2006-04.com.example:444 is target_iqn_name
```

   将 *iqn.2006-04.com.example:444* 替换为特定目标名称。

3. 验证新创建的目标：

```
/iscsi> ls

o- iscsi.....[1 Target]
  o- iqn.2006-04.com.example:444.....[1 TPG]
    o- tpg1.....[enabled, auth]
      o- acls.....[0 ACL]
       o- luns.....[0 LUN]
      o- portals.....[0 Portal]
```

其他资源

- **targetcli(8)** 手册页

7.3. ISCSI BACKSTORE

iSCSI 后端存储支持支持在本地计算机上存储导出的 LUN 数据的不同方法。创建存储对象定义了后端存储使用的资源。

管理员可以选择 Linux-IO(LIO)支持的任何后端存储设备：

fileio 后端存储

如果您将本地文件系统中的常规文件用作磁盘镜像，请创建一个 **fileio** 存储对象。有关创建 **fileio** 后端存储的信息，请参阅创建 fileio 存储对象。

Block backstore

如果您使用任何本地块设备和逻辑设备，请创建一个 **块存储** 对象。有关创建块后备存储的信息，请参阅 创建块存储对象。

pscsi 后端存储

如果您的存储对象支持直接通过 SCSI 命令，请创建一个 **pscsi** 存储对象。有关创建 **pscsi** 后端存储的信息，请参阅创建 pscsi 存储对象。

ramdisk 后端存储

如果要创建临时 RAM 支持的设备，请创建一个 **ramdisk** 存储对象。有关创建 **ramdisk** 后端存储，请参阅创建内存复制 RAM 磁盘存储对象。

其他资源

- **targetcli(8)** 手册页

7.4. 创建 FILEIO 存储对象

fileio 存储对象可以支持 **write_back** 或 **write_thru** 操作。**write_back** 操作启用本地文件系统缓存。这提高了性能，但会增加数据丢失的风险。

建议使用 **write_back=false** 禁用 **write_back** 操作来使用 **write_thru** 操作。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。

步骤

1. 从 **backstores/** 目录中进入 **fileio/**:

```
/> backstores/fileio
```

2. 创建 **fileio** 存储对象：

```
/backstores/fileio> create file1 /tmp/disk1.img 200M write_back=false

Created fileio file1 with size 209715200
```

验证

验证

- 验证创建的 **fileio** 存储对象：

```
/backstores/fileio> ls
```

其他资源

- **targetcli(8)** 手册页

7.5. 创建块存储对象

块驱动程序允许使用 **/sys/block/** 目录中出现的任何块设备来与 Linux-IO(LIO)一起使用。这包括物理设备，如 HDD、SSD、CD 和 DVD，以及逻辑设备，如软件或硬件 RAID 卷、LVM 卷。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。

步骤

1. 从 **backstores/** 目录中进入 **block/**:

```
/> backstores/block/
```

2. 创建 块 后备存储：

```
/backstores/block> create name=block_backend dev=/dev/sdb

Generating a wwn serial.
Created block storage object block_backend using /dev/vdb.
```

验证

- 验证创建的块存储对象：

```
/backstores/block> ls
```

注意

您还可以在逻辑卷中创建 块 后备存储。

其他资源

- **targetcli(8)** 手册页

7.6. 创建 PSCSI 存储对象

您可以将任何支持直接传递 SCSI 命令的存储对象配置为没有 SCSI 模拟的后端存储，并带有一个底层 SCSI 设备，它在 **/proc/scsi/scsi** 中显示带有 **lsscsi**（如一个 SAS 硬盘）。这个子系统支持 SCSI-3 及更高系统。

警告

pscsi 应该仅由高级用户使用。高级 SCSI 命令（如 Asymmetric Logical Unit Assignment (ALUAs) 或 Persistent Reservations（例如被 VMware ESX, 和 vSphere 使用））通常不会在设备固件中实施，并可能导致出现故障或崩溃。如果有疑问，请在生产环境中改为使用 **block** 后端存储。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。

步骤

1. 从 **backstores/** 目录中进入到 **pscsi/** ：

```
/> backstores/pscsi/
```

2. 为物理 SCSI 设备创建 **pscsi** 后端存储，本示例中是使用 **/dev/sr0** 的 TYPE_ROM 设备：

```
/backstores/pscsi> create name=pscsi_backend dev=/dev/sr0

Generating a wwn serial.
Created pscsi storage object pscsi_backend using /dev/sr0
```

验证

- 验证创建的 **pscsi** 存储对象：

```
/backstores/pscsi> ls
```

其他资源

- **targetcli(8)** 手册页

7.7. 创建内存副本 RAM 磁盘存储对象

Memory Copy RAM 磁盘(**ramdisk**)为 RAM 磁盘提供完整的 SCSI 模拟，并使用启动器的内存副本来分隔内存映射。这为多会话提供了功能，在用于生产目的的快速和可变量存储中特别有用。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。

步骤

1. 从 **backstores/** 目录进入到 **ramdisk/** ：

```
/> backstores/ramdisk/
```

2. 创建 1GB RAM 磁盘后备存储：

```
/backstores/ramdisk> create name=rd_backend size=1GB

Generating a wwn serial.
Created rd_mcp ramdisk rd_backend with size 1GB.
```

验证

- 验证创建的 **ramdisk** 存储对象：

```
/backstores/ramdisk> ls
```

其他资源

- **targetcli(8)** 手册页

7.8. 创建 ISCSI 门户

创建 iSCSI 门户，为目标添加一个 IP 地址和端口来启用目标。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。
- 与目标门户组(TPG)关联的 iSCSI 目标。如需更多信息，请参阅创建 iSCSI 目标。

步骤

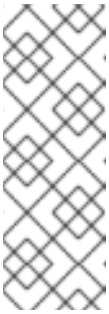
1. 进入 TPG 目录：

```
/iscsi> iqn.2006-04.example:444/tpg1/
```

2. 使用以下选项之一创建 iSCSI 门户：

   a. 创建默认门户使用默认 iSCSI 端口 **3260**，并允许目标侦听该端口上的所有 IP 地址：

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create

Using default IP port 3260
Binding to INADDR_Any (0.0.0.0)
Created network portal 0.0.0.0:3260
```



注意

当创建 iSCSI 目标时，也会创建一个默认门户网站。这个门户设置为使用默认端口号侦听所有 IP 地址：**0.0.0.0:3260**。

要删除默认门户网站，请使用以下命令：

```
/iscsi/iqn-name/tpg1/portals delete ip_address=0.0.0.0 ip_port=3260
```

b. 使用特定 IP 地址创建门户：

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create 192.168.122.137

Using default IP port 3260
Created network portal 192.168.122.137:3260
```

验证

- 验证新创建的门户网站：

```
/iscsi/iqn.20...mple:444/tpg1> ls

o- tpg..... [enambled, auth]
  o- acls .....[0 ACL]
  o- luns .....[0 LUN]
  o- portals .....[1 Portal]
    o- 192.168.122.137:3260.....[OK]
```

其他资源

- **targetcli(8)** 手册页

7.9. 创建 ISCSI LUN

逻辑单元号(LUN)是一个由 iSCSI 后端存储支持的物理设备。每个 LUN 都有唯一的数字。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。
- 与目标门户组(TPG)关联的 iSCSI 目标。如需更多信息，请参阅创建 iSCSI 目标。
- 已创建存储对象。如需更多信息，请参阅 iSCSI Backstore。

步骤

1. 创建已创建的存储对象的 LUN：

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/ramdisk/rd_backend
Created LUN 0.

/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/block/block_backend
Created LUN 1.

/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/fileio/file1
Created LUN 2.
```

2. 验证创建的 LUN：

```
/iscsi/iqn.20...mple:444/tpg1> ls

o- tpg..... [enambled, auth]
```

```
o- acls ..........[0 ACL]
o- luns ..........[3 LUNs]
| o- lun0.....[ramdisk/ramdisk1]
| o- lun1.....[block/block1 (/dev/vdb1)]
| o- lun2.....[fileio/file1 (/foo.img)]
o- portals ..........[1 Portal]
  o- 192.168.122.137:3260.....[OK]
```

默认 LUN 名称以 **0** 开头。

重要

默认情况下，使用读写权限创建 LUN。如果在创建 ACL 后添加了新的 LUN，LUN 会自动映射到所有可用 ACL，并可能导致安全风险。要创建具有只读权限的 LUN，请参阅创建只读 iSCSI LUN。

3. 配置 ACL。如需更多信息，请参阅创建 iSCSI ACL。

其他资源

- **targetcli(8)** 手册页

7.10. 创建只读 ISCSI LUN

默认情况下，使用读写权限创建 LUN。这个步骤描述了如何创建只读 LUN。

先决条件

- 已安装并运行 **targetcli**。如需更多信息，请参阅安装 targetcli。
- 与目标门户组(TPG)关联的 iSCSI 目标。如需更多信息，请参阅创建 iSCSI 目标。
- 已创建存储对象。如需更多信息，请参阅 iSCSI Backstore。

步骤

1. 设置只读权限：

```
/> set global auto_add_mapped_luns=false

Parameter auto_add_mapped_luns is now 'false'.
```

这样可防止 LUN 自动映射到现有 ACL，从而允许手动映射 LUN。

2. 进入 *initiator_iqn_name* 目录：

```
/> iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name/
```

3. 创建 LUN：

```
/iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name> create
mapped_lun=next_sequential_LUN_number tpg_lun_or_backstore=backstore
write_protect=1
```

例如：

```
/iscsi/target_iqn_name/tpg1/acls/2006-04.com.example:888> create mapped_lun=1
tpg_lun_or_backstore=/backstores/block/block2 write_protect=1

Created LUN 1.
Created Mapped LUN 1.
```

4. 验证所创建的 LUN：

```
/iscsi/target_iqn_name/tpg1/acls/2006-04.com.example:888> ls
 o- 2006-04.com.example:888 .. [Mapped LUNs: 2]
 | o- mapped_lun0 ..... [lun0 block/disk1 (rw)]
 | o- mapped_lun1 ..... [lun1 block/disk2 (ro)]
```

mapping_lun1 行现在在结尾带有(**ro**)（不像 mapping_lun0 为 (**rw**)），这表示它是只读的。

5. 配置 ACL。如需更多信息，请参阅创建 iSCSI ACL。

其他资源

- **targetcli(8)** 手册页

7.11. 创建 ISCSI ACL

targetcli 服务使用访问控制列表(ACL)来定义访问规则，并授予每个启动器访问逻辑单元号(LUN)的权限。

目标和发起方都有唯一的标识名称。您必须知道配置 ACL 的发起方的唯一名称。由 **iscsi-initiator-utils** 软件包提供的 **/etc/iscsi/initiatorname.iscsi** 文件包含 iSCSI 启动器名称。

先决条件

- **targetcli** 服务 已安装 并运行。
- 与目标门户组(TPG)关联的 iSCSI 目标。

流程

1. 可选：要禁用 LUN 到 ACL 的自动映射，请参阅 创建只读 iSCSI LUN。
2. 进入到 acls 目录：

```
/> iscsi/target_iqn_name/tpg_name/acls/
```

3. 使用以下选项之一创建 ACL：
   - 使用启动器上 **/etc/iscsi/initiatorname.iscsi** 文件中的 *initiator_iqn_name* ：

```
iscsi/target_iqn_name/tpg_name/acls> create initiator_iqn_name

Created Node ACL for initiator_iqn_name
```

```
Created mapped LUN 2.
Created mapped LUN 1.
Created mapped LUN 0.
```

- 使用 *custom_name*，并更新启动器以匹配它：

```
iscsi/target_iqn_name/tpg_name/acls> create custom_name

Created Node ACL for custom_name
Created mapped LUN 2.
Created mapped LUN 1.
Created mapped LUN 0.
```

有关更新启动器名称的详情，请参考 创建 iSCSI 启动器。

验证

- 验证创建的 ACL：

```
iscsi/target_iqn_name/tpg_name/acls> ls

o- acls .....[1 ACL]
  o- target_iqn_name ....[3 Mapped LUNs, auth]
    o- mapped_lun0 .....[lun0 ramdisk/ramdisk1 (rw)]
    o- mapped_lun1 .....[lun1 block/block1 (rw)]
    o- mapped_lun2 .....[lun2 fileio/file1 (rw)]
```

其他资源

- **targetcli(8)** 手册页

7.12. 为目标设置 CHALLENGE-HANDSHAKE 验证协议

通过使用 **Challenge-Handshake Authentication Protocol(CHAP)**，用户可以使用密码保护目标。发起方必须了解这个密码才能连接到目标。

先决条件

- 创建 iSCSI ACL。如需更多信息，请参阅创建 iSCSI ACL。

步骤

1. 设置属性身份验证：

```
/iscsi/iqn.20...mple:444/tpg1> set attribute authentication=1

Parameter authentication is now '1'.
```

2. 设置 **userid** 和**密码** ：

```
/tpg1> set auth userid=redhat
Parameter userid is now 'redhat'.
```

```
/iscsi/iqn.20...689dcbb3/tpg1> set auth password=redhat_passwd
Parameter password is now 'redhat_passwd'.
```

其他资源

- **targetcli(8)** 手册页

7.13. 使用 TARGETCLI 工具删除 ISCSI 对象

这个步骤描述了如何使用 **targetcli** 工具删除 iSCSI 对象。

步骤

1. 从目标登出：

```
# iscsiadm -m node -T iqn.2006-04.example:444 -u
```

有关如何登录到目标的更多信息，请参阅 创建 iSCSI 启动器。

2. 删除整个目标，包括所有 ACL、LUN 和门户：

```
/> iscsi/ delete iqn.2006-04.com.example:444
```

将 *iqn.2006-04.com.example:444* 替换为 target_iqn_name。

- 删除 iSCSI 后端存储：

```
/> backstores/backstore-type/ delete block_backend
```

  - 使用 **fileio**、**block**、**pscsi** 或 **ramdisk** 替换 *backstore-type*。
  - 使用您要删除的 *backstore-name* 替换 *block_backend*。

- 要删除 iSCSI 目标的部分，如 ACL：

```
/> /iscsi/iqn-name/tpg/acls/ delete iqn.2006-04.com.example:444
```

验证

- 查看更改：

```
/> iscsi/ ls
```

其他资源

- **targetcli(8)** 手册页

第 8 章 配置 ISCSI INITIATOR

iSCSI 启动程序形成连接 iSCSI 目标的会话。默认情况下，iSCSI 服务是"懒启动"，服务只在运行 **iscsiadm** 命令后才启动。如果 root 没有位于 iSCSI 设备，或者没有标记为 **node.startup = automatic** 的节点，则 iSCSI 服务将不会启动，直到执行 **iscsiadm** 命令后，需要 **iscsid** 或 **iscsi** 内核模块启动。

以 root 用户身份执行 **systemctl start iscsid.service** 命令，以强制 **iscsid** 守护进程运行和 iSCSI 内核模块来加载。

8.1. 创建 ISCSI 启动程序

创建 iSCSI 启动器以连接到 iSCSI 目标，以访问服务器上的存储设备。

先决条件

- 您有一个 iSCSI 目标的主机名和 IP 地址：
  - 如果您要连接到外部软件创建的存储目标，请从存储管理员查找目标主机名和 IP 地址。
  - 如果您要创建 iSCSI 目标，请参阅创建 iSCSI 目标。

流程

1. 在客户端机器上安装 **iscsi-initiator-utils**:

```
# dnf install iscsi-initiator-utils
```

2. 检查 initiator 名称：

```
# cat /etc/iscsi/initiatorname.iscsi

InitiatorName=iqn.2006-04.com.example:888
```

3. 如果在 创建 iSCI ACL 时 ACL 被指定了一个自定义名称，请更新启动器名称以匹配 ACL：
   a. 打开 **/etc/iscsi/initiatorname.iscsi** 文件并修改启动器名称：

```
# vi /etc/iscsi/initiatorname.iscsi

InitiatorName=custom-name
```

   b. 重启 **iscsid** 服务：

```
# systemctl restart iscsid
```

4. 发现目标并使用显示的目标 IQN 登录到目标：

```
# iscsiadm -m discovery -t st -p 10.64.24.179
   10.64.24.179:3260,1 iqn.2006-04.example:444

# iscsiadm -m node -T iqn.2006-04.example:444 -l
   Logging in to [iface: default, target: iqn.2006-04.example:444, portal: 10.64.24.179,3260]
```

```
(multiple)
    Login to [iface: default, target: iqn.2006-04.example:444, portal: 10.64.24.179,3260]
successful.
```

将 *10.64.24.179* 替换为 target-ip-address。

如果将相应的启动器名称添加到 ACL 中，您可以将这个步骤用于连接到同一目标的任意启动器名称，如创建 iSCSI ACL 所述。

5. 找到 iSCSI 磁盘名称并在这个 iSCSI 磁盘中创建文件系统：

```
# grep "Attached SCSI" /var/log/messages

# mkfs.ext4 /dev/disk_name
```

使用 **/var/log/messages** 文件中显示的 iSCSI 磁盘名称替换 *disk_name*。

6. 挂载文件系统：

```
# mkdir /mount/point

# mount /dev/disk_name /mount/point
```

使用分区的挂载点替换 */mount/point*。

7. 编辑 **/etc/fstab** 文件，以便在系统引导时自动挂载文件系统：

```
# vi /etc/fstab

/dev/disk_name /mount/point ext4 _netdev 0 0
```

使用 iSCSI 磁盘名称替换 *disk_name*，使用分区的挂载点替换 */mount/point*。

其他资源

- **targetcli(8)** 和 **iscsiadm(8)** man page

8.2. 为发起方设置 CHALLENGE-HANDSHAKE AUTHENTICATION PROTOCOL

通过使用 **Challenge-Handshake Authentication Protocol(CHAP)**，用户可以使用密码保护目标。发起方必须了解这个密码才能连接到目标。

先决条件

- 创建 iSCSI initiator。如需更多信息，请参阅创建 iSCSI 启动器。
- 为目标设置 **CHAP**。如需更多信息，请参阅 为目标设置 Challenge-Handshake Authentication Protocol。

步骤

1. 在 **iscsid.conf** 文件中启用 CHAP 验证：

```
# vi /etc/iscsi/iscsid.conf

node.session.auth.authmethod = CHAP
```

默认情况下，**node.session.auth.authmethod** 设置为 **None**

2. 在 **iscsid.conf** 文件中添加目标**用户名**和**密码** ：

```
node.session.auth.username = redhat
node.session.auth.password = redhat_passwd
```

3. 启动 **iscsid** 守护进程：

```
# systemctl start iscsid.service
```

其他资源

- **iscsiadm(8)** man page

8.3. 使用 ISCSIADM 程序监控 ISCSI 会话

这个步骤描述了如何使用 **iscsiadm** 程序监控 iscsi 会话。

默认情况下，iSCSI 服务是**懒**启动，服务只在运行 **iscsiadm** 命令后才启动。如果 root 没有位于 iSCSI 设备，或者没有标记为 **node.startup = automatic** 的节点，则 iSCSI 服务将不会启动，直到执行 **iscsiadm** 命令后，需要 **iscsid** 或 **iscsi** 内核模块启动。

以 root 用户身份执行 **systemctl start iscsid.service** 命令，以强制 **iscsid** 守护进程运行和 iSCSI 内核模块来加载。

步骤

1. 在客户端机器上安装 **iscsi-initiator-utils**:

```
# dnf install iscsi-initiator-utils
```

2. 查找正在运行的会话的信息：

```
# iscsiadm -m session -P 3
```

这个命令显示会话或设备状态、会话 ID(sid)、一些协商的参数以及可通过会话访问的 SCSI 设备。

- 如果只需要简短的输出，例如：只显示 **sid-to-node** 映射，请运行：

```
# iscsiadm -m session -P 0
      or
# iscsiadm -m session

tcp [2] 10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
tcp [3] 10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

这些命令以以下格式显示了正在运行的会话列表： **driver [sid] target_ip:port,target_portal_group_tag proper_target_name**。

其他资源

- **/usr/share/doc/iscsi-initiator-utils-version/README** 文件
- **iscsiadm(8)** man page

8.4. DM 多路径覆盖设备超时

restore_tmo sysfs 选项控制一个特定 iSCSI 设备的超时时间。以下选项全局覆盖 **recovery_tmo** 值：

- **replacement_timeout** 配置选项会全局覆盖所有 iSCSI 设备的 **recovery_tmo** 值。
- 对于由 DM 多路径管理的所有 iSCSI 设备，DM 多路径中的 **fast_io_fail_tmo** 选项会全局覆盖 **recovery_tmo** 值。
  DM 多路径中的 **fast_io_fail_tmo** 选项会覆盖光纤通道设备的 **fast_io_fail_tmo** 选项。

DM 多路径 **fast_io_fail_tmo** 选项优先于 **replacement_timeout**。红帽不推荐使用 **replacement_timeout** 覆盖由 DM 多路径管理的设备中的 **recovery_tmo**，因为在 **multipathd** 服务重新载入时 DM 多路径总是重置 **recovery_tmo**。

第 9 章 使用光纤通道设备

Red Hat Enterprise Linux 9 提供以下原生光纤频道驱动程序：

- **lpfc**
- **qla2xxx**
- **zfcp**

9.1. 重新调整光纤通道逻辑单元的大小

作为系统管理员，您可以重新定义光纤通道逻辑单元大小。

步骤

1. 确定哪些设备是 **多路径** 逻辑单元的路径：

```
multipath -ll
```

2. 在使用多路径的系统中重新扫描光纤通道逻辑单元：

```
$ echo 1 > /sys/block/sdX/device/rescan
```

其他资源

- **multipath(8)** 手册页

9.2. 使用光纤通道确定设备链路丢失行为

如果驱动程序实现了传输 **dev_loss_tmo** 回调，当检测到传输问题时，通过链接访问设备的尝试将被阻止。

步骤

- 确定远程端口的状态：

```
$ cat /sys/class/fc_remote_port/rport-host:bus:remote-port/port_state
```

这个命令返回以下输出结果之一：

  - 当远程端口以及通过它访问的设备被阻止时为 **Blocked**。
  - 如果远程端口正常运行，则为 **Online**
    如果在 **dev_loss_tmo** 秒内没有解决这个问题，则 **rport** 和设备将被取消阻塞。所有在该设备上运行的 I/O 以及发送到该设备的新 I/O 将失败。

当链路丢失超过 **dev_loss_tmo** 时，会删除 **scsi_device** 和 **sd_N_** 设备。通常，光纤通道类会将设备保留原样，即 **/dev/*sdx*** 将保留 **/dev/*sdx***。这是因为目标绑定由 Fibre Channel 驱动程序保存，当目标端口返回时，SCSI 地址会被重新创建。但是，这无法保证，只有在无法进行 LUN 的存储框中配置时，才会恢复 ***sdx*** 设备。

其他资源

- **multipath.conf(5)** 手册页
- 推荐在 scsi、multipath 和应用程序层进行调优，同时配置 Oracle RAC 集群 知识库文章

9.3. FIBRE CHANNEL 配置文件

以下是 **/sys/class/** 目录中为 Fibre Channel 提供用户空间 API 的配置文件列表。

项目使用以下变量：

H
主机号

B
总线号

T
目标

L
逻辑单元(LUN)

R
远程端口号

重要

如果您的系统使用多路径软件，请在更改本节中描述的任何值前咨询您的硬件厂商。

传输配置 /sys/class/fc_transport/target*H*:*B*:*T*/

port_id
24 位端口 ID/地址

node_name
64 位节点名称

port_name
64 位端口名称

远程端口配置 /sys/class/fc_remote_ports/rport-*H*:*B*-*R*/

- **port_id**
- **node_name**
- **port_name**
- **dev_loss_tmo**
  控制 scsi 设备从系统中删除的时间。在 **dev_loss_tmo** 触发后，scsi 设备被删除。在 **multipath.conf** 文件中，您可以将 **dev_loss_tmo** 设置为 **infinity**。

  在 Red Hat Enterprise Linux 9 中，如果您没有设置 **fast_io_fail_tmo** 选项，**dev_loss_tmo** 的上限将为 **600** 秒。默认情况下，在 Red Hat Enterprise Linux 9 中， 如果 **multipathd** 服务正在运行，**fast_io_fail_tmo** 会被设置为 **5** 秒，如果没有运行，会被设置为 **off**。

- **fast_io_fail_tmo**
  指定在将链接标记为"bad"之前要等待的秒数。链接被标记为坏的后，现有正在运行的 I/O 或相应路径上的任何新 I/O 都将失败。

  如果 I/O 处于阻塞队列中，则在 **dev_loss_tmo** 到期前和队列未阻塞前，它不会失败。

  如果 **fast_io_fail_tmo** 被设为不是 off 的任何值时，则会取消封 **dev_loss_tmo** 的上限。如果 **fast_io_fail_tmo** 设为 off，则在设备从系统中删除之前不会出现 I/O 失败。如果 **fast_io_fail_tmo** 设置为一个数字，则在达到 **fast_io_fail_tmo** 设置的超时会立即触发 I/O 失败。

主机设置 /sys/class/fc_host/host*H*/

- **port_id**
- **node_name**
- **port_name**
- **issue_lip**
  指示驱动重新发现远程端口。

9.4. DM 多路径覆盖设备超时

restore_tmo sysfs 选项控制一个特定 iSCSI 设备的超时时间。以下选项全局覆盖 **recovery_tmo** 值：

- **replacement_timeout** 配置选项会全局覆盖所有 iSCSI 设备的 **recovery_tmo** 值。
- 对于由 DM 多路径管理的所有 iSCSI 设备，DM 多路径中的 **fast_io_fail_tmo** 选项会全局覆盖 **recovery_tmo** 值。
  DM 多路径中的 **fast_io_fail_tmo** 选项会覆盖光纤通道设备的 **fast_io_fail_tmo** 选项。

DM 多路径 **fast_io_fail_tmo** 选项优先于 **replacement_timeout**。红帽不推荐使用 **replacement_timeout** 覆盖由 DM 多路径管理的设备中的 **recovery_tmo**，因为在 **multipathd** 服务重新载入时 DM 多路径总是重置 **recovery_tmo**。

第 10 章 使用快照管理系统升级

执行 Red Hat Enterprise Linux 系统的可回滚升级，以返回到操作系统的早期版本。您可以使用 **Boom Boot Manager** 和 Leapp 操作系统现代化框架。

在执行操作系统升级前，请考虑以下方面：

- 使用快照的系统升级不适用于系统树中的多个文件系统，例如，独立的 **/var** 或 **/usr** 分区。
- 使用快照的系统升级不适用于 Red Hat Update Infrastructure (RHUI)系统。不使用 Boom 工具，考虑创建虚拟机的快照(VM)。

10.1. BOOM 过程概述

使用 Boom 引导管理器创建引导条目，以便您可以从 GRUB 引导装载程序菜单中选择和访问这些条目。创建引导条目简化了准备可回滚升级的过程。

以下引导条目是升级和回滚过程的一部分：

- **升级引导条目**
  引导 Leapp 升级环境。使用 **leapp** 实用程序创建和管理此引导条目。**leapp** 升级过程会自动删除这个条目。
- **Red Hat Enterprise Linux 9 引导条目**
  引导升级系统环境。使用 **leapp** 实用程序在成功升级后创建此引导条目。
- **快照引导条目**
  引导原始系统的快照。在成功或成功升级尝试后，使用它来检查并测试之前的操作系统状态。在升级操作系统前，请使用 **boom** 命令创建此引导条目。
- **回滚引导条目**
  引导原始系统环境，并将任何升级回滚到以前的系统状态。在启动升级过程的回滚时，请使用 **boom** 命令创建此引导条目。

其他资源

- **boom (1)** 手册页

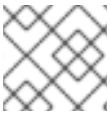
10.2. 使用 BOOM BOOT MANAGER 升级至另一个版本

使用 Boom 引导管理器执行 Red Hat Enterprise Linux 操作系统的升级。

先决条件

- 您正在运行 Red Hat Enterprise Linux 的当前版本。
- 您已安装 **boom-boot** 软件包的当前版本（版本 **boom-1.3-3.el9**，最好是 **boom-1.4-4.el9** 或更高版本）。
- 您有足够的空间用于快照。根据原始安装的大小估计大小。列出所有挂载的逻辑卷。
- 您已安装 **leapp** 软件包。
- 您已启用了软件存储库。

- 您已激活了快照卷。如果它没有激活，则 **boom** 命令会失败。



注意

其他引导条目可能包括 **/usr** 或 **/var**。

流程

1. 创建您的 *root* 逻辑卷快照：

   - 如果您的 root 文件系统使用精简配置，请创建一个精简快照：

     ```
     # lvcreate -s rhel/root -kn -n root_snapshot_before_changes
     ```

     在这里：

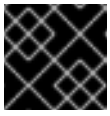
     - **-s** 创建快照。
     - **rhel/root** 将文件系统复制到逻辑卷。
     - **-n** *root_snapshot_before_changes* 显示快照的名称。
       在创建精简快照时，不要定义快照大小。快照从精简池中分配。

   - 如果您的 root 文件系统使用 thick 置备，请创建一个 thick 快照：

     ```
     # lvcreate -s  rhel/root -n root_snapshot_before_changes -L 25g
     ```

     在这里：

     - **-s** 创建快照。
     - **rhel/root** 将文件系统复制到逻辑卷。
     - **-n** *root_snapshot_before_changes* 显示快照的名称。
     - **-L** *25g* 是快照大小。根据原始安装的大小估计大小。
       在创建厚快照时，定义可保存升级过程中所有更改的快照大小。

       重要

       创建的快照不包括任何其他系统更改。

2. 创建配置集：

   ```
   # boom profile create --from-host --uname-pattern el9
   ```

3. 使用原始引导镜像的备份副本创建原始系统的快照引导条目：

   ```
   # boom create --backup --title "Root LV snapshot before changes" --rootlv
   rhel/root_snapshot_before_changes
   ```

   在这里：

- **--title** *Root LV snapshot before changes* 是引导条目的名称，它会在系统启动期间在引导条目列表中显示。
- **--rootlv** 是与新引导条目对应的根逻辑卷。
- 完成上一步后，您有一个引导条目，允许在升级前访问原始系统。

4. 使用 Leapp 升级到 Red Hat Enterprise Linux 9：

```
# leapp upgrade
```

- 检查并解决 **leapp upgrade** 命令报告中任何指示的阻塞。

5. 重启到升级的引导条目：

```
# leapp upgrade --reboot
```

- 从 GRUB 引导屏幕中选择 **Red Hat Enterprise Linux Upgrade Initramfs** 条目。
- **leapp** 工具会创建升级引导条目。运行上述命令以重启到升级引导条目，并继续执行到 Red Hat Enterprise Linux 9 的原位升级。升级过程后，reboot 参数将启动自动系统重启。GRUB 屏幕在重启过程中显示。

注意

Red Hat Enterprise Linux 9 不提供 GRUB 引导屏幕中的 Snapshots 子菜单。

验证步骤

- 继续升级并安装新的 Red Hat Enterprise Linux 9 RPM 软件包。完成升级后，系统会自动重启。GRUB 屏幕显示可用操作系统的升级版本以及旧版本。升级的系统版本是默认选择。
- 检查 **Root LV snapshot before changes** 引导条目是否在 GRUB 菜单中。如果存在，它提供升级前对操作系统状态的即时访问。

其他资源

- **boom (1)** 手册页
- 什么是 BOOM 以及如何安装它？
- 如何创建 BOOM 引导条目

10.3. 在 RED HAT ENTERPRISE LINUX 版本间切换

在您的机器上同时访问当前和以前的 Red Hat Enterprise Linux 版本。使用 **Boom Boot Manager** 访问不同的操作系统版本会降低与升级操作系统相关的风险，并有助于减少硬件停机时间。通过能够在环境间进行切换，您可以：

- 以并排的方式快速比较两个环境。
- 以最少的开销在环境之间进行切换。
- 恢复文件系统的旧内容。

- 在升级的主机运行期间，继续访问旧的系统。
- 随时停止并恢复更新过程，即使更新本身正在运行。

先决条件

- 您正在运行 Red Hat Enterprise Linux 的当前版本。

流程

1. 重启系统：

```
# reboot
```

2. 从 GRUB 引导装载程序屏幕中选择所需的引导条目。

验证步骤

- 验证所选引导卷是否已显示：

```
# cat /proc/cmdline

root=/dev/rhel/root_snapshot_before_changes ro
rd.lvm.lv=rhel/root_snapshot_before_changes rd.lvm.lv=vg_root/swap rhgb quiet
```

其他资源

- **boom (1)** 手册页
- 使用 Boom Boot Manager 升级至另一个版本

10.4. 删除逻辑卷快照

通过创建当前操作系统的快照，您可以访问、审核和测试操作系统之前的状态。在完成操作系统快照后，您可以将其删除来释放存储空间。

重要

删除后，您无法使用逻辑卷(LV)快照执行进一步的操作。

先决条件

- 您正在运行 Red Hat Enterprise Linux 的当前版本。

流程

1. 从 GRUB 条目引导到 Red Hat Enterprise Linux 9。以下输出确认选择了新的快照：

```
# boom list
BootID    Version                 Name                        RootDevice
6d2ec72 3.10.0-957.21.3.el8.x86_64 Red Hat Enterprise Linux Server
/dev/rhel/root_snapshot_before_changes
```

2. 使用 **BootID** 值删除快照条目：

```
# boom delete --boot-id 6d2ec72
```

- 这会从 GRUB 菜单中删除引导条目。

3. 删除 LV 快照：

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

其他资源

- **boom (1)** 手册页
- 使用 Boom Boot Manager 升级至另一个版本

10.5. 创建回滚引导条目

使用回滚引导条目访问升级前状态的操作系统环境。另外，您可以恢复任何操作系统升级。

从升级的系统或快照环境准备回滚引导条目。

先决条件

- 您正在运行 Red Hat Enterprise Linux 的当前版本。

流程

1. 将快照与原始卷（源）合并：

```
# lvconvert --merge rhel/root_snapshot_before_changes
```

警告

合并快照后，您必须继续这个流程中的所有剩余步骤来防止数据丢失。

2. 为合并的快照创建一个回滚引导条目：

- 对于 **boom-0.9** ：

```
boom create --title "RHEL Rollback" --rootlv rhel/root
```

- 对于 **boom-1.2**，或更新版本：

```
boom create --backup --title "RHEL Rollback" --rootlv rhel/root
```

3. 可选：重启机器以恢复操作系统状态：

```
# reboot
```

- 系统重启后，从 GRUB 屏幕中选择 Red Hat Enterprise Linux Rollback 引导条目。
- 当 **root** 逻辑卷激活后，系统会自动启动快照合并操作。

重要

合并操作启动后，快照卷将不再可用。成功引导 Red Hat Enterprise Linux Rollback 引导条目后，**Root LV 快照引导条目** 将不再工作。合并快照逻辑卷会破坏 Root LV 快照，并恢复原始卷的之前状态。

4. 可选：完成合并操作后，删除未使用的条目，并恢复原始引导条目：

   a. 从 **/boot** 文件系统中删除未使用的 Red Hat Enterprise Linux 9 引导条目，并重建 **grub.cfg** 文件以使更改生效：

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   b. 恢复原始 Red Hat Enterprise Linux 引导条目：

   ```
   # new-kernel-pkg --update $(uname -r)
   ```

5. 成功回滚系统后，删除 **boom** 引导条目：

```
# boom list
# boom delete boot-id
```

其他资源

- **boom (1)** 手册页
- 使用 Boom Boot Manager 升级至另一个版本

第 11 章 使用 NVME/RDMA 配置 NVME OVER FABRICS

在通过 RDMA (NVMe™/RDMA)设置的 Non-volatile Memory Express™(NVMe™/RDMA)中，您可以配置 NVMe 控制器和 NVMe 启动器。

作为系统管理员，完成以下任务来部署 NVMe/RDMA 设置：

- 使用 configfs 设置 NVMe/RDMA 控制器
- 使用 nvmetcli 设置 NVMe/RDMA 控制器
- 配置 NVMe/RDMA 主机

11.1. NVME OVER FABRIC 设备概述

Non-volatile Memory Express™(NVMe™)是一个接口，允许主机软件工具与固态硬盘进行通信。

使用以下类型的光纤传输来通过光纤设备配置 NVMe：

NVMe over Remote Direct Memory Access(NVMe/RDMA)

有关如何配置 NVMe™/RDMA 的详情，请参考 使用 NVMe/RDMA 配置光纤上的 NVMe。

NVMe over Fibre Channel (NVMe/FC)

有关如何配置 NVMe™/FC 的详情，请参考 使用 NVMe/FC 配置光纤上的 NVMe。

NVMe over TCP (NVMe/TCP)

有关如何配置 NVMe/FC 的详情，请参考 使用 NVMe/TCP 配置光纤上的 NVMe。

当使用 NVMe over fabrics 时，固态驱动器不必位于您的系统本地，它可以通过 NVMe over fabrics 设备远程配置。

11.2. 使用 CONFIGFS 设置 NVME/RDMA 控制器

使用此流程，使用 **configfs** 配置 RDMA (NVMe™/RDMA)控制器上的 Non-volatile Memory Express™（NVMe™™(NVMe™)。

先决条件

- 验证您有一个要分配给 **nvmet** 子系统的块设备。

步骤

1. 创建 **nvmet-rdma** 子系统：

```
# modprobe nvmet-rdma

# mkdir /sys/kernel/config/nvmet/subsystems/testnqn

# cd /sys/kernel/config/nvmet/subsystems/testnqn
```

使用子系统名称替换 *testnqn*。

2. 允许任何主机连接到此控制器：

```
# echo 1 > attr_allow_any_host
```

3. 配置命名空间：

```
# mkdir namespaces/10

# cd namespaces/10
```

使用命名空间号替换 *10*

4. 设置 NVMe 设备的路径：

```
# echo -n /dev/nvme0n1 > device_path
```

5. 启用命名空间：

```
# echo 1 > enable
```

6. 创建带有 NVMe 端口的目录：

```
# mkdir /sys/kernel/config/nvmet/ports/1

# cd /sys/kernel/config/nvmet/ports/1
```

7. 显示 *mlx5_ib0* 的 IP 地址：

```
# ip addr show mlx5_ib0

8: mlx5_ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 4092 qdisc mq state UP
group default qlen 256
    link/infiniband 00:00:06:2f:fe:80:00:00:00:00:00:00:e4:1d:2d:03:00:e7:0f:f6 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    inet 172.31.0.202/24 brd 172.31.0.255 scope global noprefixroute mlx5_ib0
       valid_lft forever preferred_lft forever
    inet6 fe80::e61d:2d03:e7:ff6/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

8. 为控制器设置传输地址：

```
# echo -n 172.31.0.202 > addr_traddr
```

9. 将 RDMA 设置为传输类型：

```
# echo rdma > addr_trtype

# echo 4420 > addr_trsvcid
```

10. 为端口设置地址系列：

```
# echo ipv4 > addr_adrfam
```

11. 创建软链接：

-

```
# ln -s /sys/kernel/config/nvmet/subsystems/testnqn
/sys/kernel/config/nvmet/ports/1/subsystems/testnqn
```

验证

- 验证 NVMe 控制器是否在指定端口上侦听并准备好连接请求：

```
# dmesg | grep "enabling port"
[ 1091.413648] nvmet_rdma: enabling port 1 (172.31.0.202:4420)
```

其他资源

- **nvme(1)** 手册页

11.3. 使用 NVMETCLI 设置 NVME/RDMA 控制器

使用 **nvmetcli** 工具编辑、查看和启动 Non-volatile Memory Express™(NVMe™)控制器。 **nvmetcli** 实用程序提供命令行和交互式 shell 选项。使用这个流程通过 **nvmetcli** 配置 NVMe™/RDMA 控制器。

先决条件

- 验证您有一个要分配给 **nvmet** 子系统的块设备。
- 以 root 用户身份执行以下 **nvmetcli** 操作。

步骤

1. 安装 **nvmetcli** 软件包：

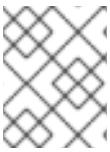
```
# dnf install nvmetcli
```

2. 下载 **rdma.json** 文件：

```
# wget
http://git.infradead.org/users/hch/nvmetcli.git/blob_plain/0a6b088db2dc2e5de11e6f23f1e890e4
b54fee64:/rdma.json
```

3. 编辑 **rdma.json** 文件，并将 **traddr** 值更改为 **172.31.0.202**。
4. 通过载入 NVMe 控制器配置文件来设置控制器：

```
# nvmetcli restore rdma.json
```



注意

如果没有指定 NVMe 控制器配置文件名称，则 **nvmetcli** 使用 **/etc/nvmet/config.json** 文件。

验证

- 验证 NVMe 控制器是否在指定端口上侦听并准备好连接请求：

```
# dmesg | tail -1
[ 4797.132647] nvmet_rdma: enabling port 2 (172.31.0.202:4420)
```

- 可选：清除当前 NVMe 控制器：

```
# nvmetcli clear
```

其他资源

- **nvmetcli** 和 **nvme(1)** man page

11.4. 配置 NVME/RDMA 主机

使用这个流程，使用 NVMe 管理命令行界面(**nvme-cli**)工具配置 RDMA (NVMe™/RDMA)主机上的 Non-volatile Memory Express™(NVMe™)。

流程

1. 安装 **nvme-cli** 工具：

```
# dnf install nvme-cli
```

2. 如果没有加载，则加载 **nvme-rdma** 模块：

```
# modprobe nvme-rdma
```

3. 在 NVMe 控制器中发现可用的子系统：

```
# nvme discover -t rdma -a 172.31.0.202 -s 4420

Discovery Log Number of Records 1, Generation counter 2
=====Discovery Log Entry 0======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified, sq flow control disable supported
portid:  1
trsvcid: 4420
subnqn:  testnqn
traddr:  172.31.0.202
rdma_prtype: not specified
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
```

4. 连接到发现的子系统：

```
# nvme connect -t rdma -n testnqn -a 172.31.0.202 -s 4420

# lsblk
NAME                  MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                     8:0    0 465.8G  0 disk
├─sda1                  8:1    0     1G  0 part /boot
```

```
└─sda2                        8:2    0 464.8G  0 part
  ├─rhel_rdma--virt--03-root 253:0    0    50G  0 lvm  /
  ├─rhel_rdma--virt--03-swap 253:1    0     4G  0 lvm  [SWAP]
  └─rhel_rdma--virt--03-home 253:2    0 410.8G  0 lvm  /home
nvme0n1

# cat /sys/class/nvme/nvme0/transport
rdma
```

使用 NVMe 子系统名称替换 *testnqn*。

将 *172.31.0.202* 替换为控制器 IP 地址。

使用端口号替换 *4420*。

验证

- 列出当前连接的 NVMe 设备：

```
# nvme list
```

- 可选：从控制器断开连接：

```
# nvme disconnect -n testnqn
NQN:testnqn disconnected 1 controller(s)

# lsblk
NAME                        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                           8:0    0 465.8G  0 disk
├─sda1                        8:1    0     1G  0 part /boot
└─sda2                        8:2    0 464.8G  0 part
  ├─rhel_rdma--virt--03-root 253:0    0    50G  0 lvm  /
  ├─rhel_rdma--virt--03-swap 253:1    0     4G  0 lvm  [SWAP]
  └─rhel_rdma--virt--03-home 253:2    0 410.8G  0 lvm  /home
```

其他资源

- **nvme(1)** 手册页
- Nvme-cli Github 软件仓库

11.5. 后续步骤

- 在 NVMe 设备 上启用多路径。

第 12 章 使用 NVME/FC 配置 NVME OVER FABRICS

当与特定 Broadcom Emulex 和 Marvell Qlogic Fibre Channel 适配器一起使用时，在主机模式下完全支持光纤通道上的 Non-volatile Memory Express™(NVMe™)。作为系统管理员，完成以下部分中的任务来部署 NVMe/FC 设置：

- 为广播适配器配置 NVMe 主机
- 为 QLogic 适配器配置 NVMe 主机

12.1. NVME OVER FABRIC 设备概述

Non-volatile Memory Express™(NVMe™)是一个接口，允许主机软件工具与固态硬盘进行通信。

使用以下类型的光纤传输来通过光纤设备配置 NVMe：

NVMe over Remote Direct Memory Access(NVMe/RDMA)

有关如何配置 NVMe™/RDMA 的详情，请参考 使用 NVMe/RDMA 配置光纤上的 NVMe。

NVMe over Fibre Channel (NVMe/FC)

有关如何配置 NVMe™/FC 的详情，请参考 使用 NVMe/FC 配置光纤上的 NVMe。

NVMe over TCP (NVMe/TCP)

有关如何配置 NVMe/FC 的详情，请参考 使用 NVMe/TCP 配置光纤上的 NVMe。

当使用 NVMe over fabrics 时，固态驱动器不必位于您的系统本地，它可以通过 NVMe over fabrics 设备远程配置。

12.2. 为广播适配器配置 NVME 主机

使用这个流程，使用 NVMe 管理命令行界面(**nvme-cli**)工具为 Broadcom 适配器客户端配置 Non-volatile Memory Express™(NVMe™)主机。

流程

1. 安装 **nvme-cli** 工具：

```
# dnf install nvme-cli
```

这会在 **/etc/nvme/** 目录中创建 **hostnqn** 文件。**hostn** 文件标识 NVMe 主机。

2. 查找本地和远程端口的全球节点名称(WWNN)和全球端口名称(WWPN)标识符：

```
# cat /sys/class/scsi_host/host*/nvme_info

NVME Host Enabled
XRI Dist lpfc0 Total 6144 IO 5894 ELS 250
NVME LPORT lpfc0 WWPN x10000090fae0b5f5 WWNN x20000090fae0b5f5 DID x010f00
ONLINE
NVME RPORT       WWPN x204700a098cbcac6 WWNN x204600a098cbcac6 DID x01050e
TARGET DISCSRVC ONLINE

NVME Statistics
LS: Xmt 000000000e Cmpl 000000000e Abort 00000000
```

```
LS XMIT: Err 00000000  CMPL: xb 00000000 Err 00000000
Total FCP Cmpl 00000000000008ea Issue 00000000000008ec OutIO 0000000000000002
    abort 00000000 noxri 00000000 nondlp 00000000 qdepth 00000000 wqerr 00000000 err
00000000
FCP CMPL: xb 00000000 Err 00000000
```

使用这些 **host-traddr** 和 **traddr** 值，找到子系统 NVMe 限定名称(NQN)：

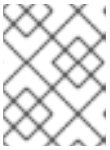
```
# nvme discover --transport fc \ --traddr nn-0x204600a098cbcac6:pn-
0x204700a098cbcac6 \ --host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5

Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0======
trtype:  fc
adrfam:  fibre-channel
subtype: nvme subsystem
treq:    not specified
portid:  0
trsvcid: none
subnqn:  nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
traddr:  nn-0x204600a098cbcac6:pn-0x204700a098cbcac6
```

将 *nn-0x204600a098cbcac6:pn-0x204700a098cbcac6* 替换为 **traddr**。

将 *nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5* 替换为 **host-traddr**。

3. 使用 **nvme-cli** 连接到 NVMe 控制器：

```
# nvme connect --transport fc \ --traddr nn-0x204600a098cbcac6:pn-
0x204700a098cbcac6 \ --host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 \ -
n nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_
1 \ -k 5
```



注意

如果您在连接时间超过默认的 keep-alive timeout 值时看到 **keep-alive timer (5 seconds) expired!** 错误，请使用 **-k** 选项增加它。例如，您可以使用 **-k 7**。

在这里，

将 *nn-0x204600a098cbcac6:pn-0x204700a098cbcac6* 替换为 **traddr**。

将 *nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5* 替换为 **host-traddr**。

将 *nqn.1992-08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1* 替换为 ***subnqn***。

将 *5* 替换为 keep-alive timeout 值（以秒为单位）。

验证

- 列出当前连接的 NVMe 设备：

```
# nvme list
Node            SN                   Model                                    Namespace Usage
Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------------------------
- ---------------- --------
/dev/nvme0n1     80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller                   1
107.37  GB / 107.37  GB      4 KiB +  0 B   FFFFFFFF


# lsblk |grep nvme
nvme0n1                 259:0    0   100G  0 disk
```

其他资源

- **nvme(1)** 手册页
- Nvme-cli Github 软件仓库

12.3. 为 QLOGIC 适配器配置 NVME 主机

使用这个流程，使用 NVMe 管理命令行界面(**nvme-cli**)工具为 Qlogic 适配器客户端配置 Non-volatile Memory Express™(NVMe™)主机。

流程

1. 安装 **nvme-cli** 工具：

```
# dnf install nvme-cli
```

这会在 **/etc/nvme/** 目录中创建 **hostnqn** 文件。**hostn** 文件标识 NVMe 主机。

2. 重新载入 **qla2xxx** 模块：

```
# modprobe -r qla2xxx
# modprobe qla2xxx
```

3. 查找本地和远程端口的全球节点名称(WWNN)和全球端口名称(WWPN)标识符：

```
# dmesg |grep traddr

[    6.139862] qla2xxx [0000:04:00.0]-ffff:0: register_localport: host-traddr=nn-
0x20000024ff19bb62:pn-0x21000024ff19bb62 on portID:10700
[    6.241762] qla2xxx [0000:04:00.0]-2102:0: qla_nvme_register_remote: traddr=nn-
0x203b00a098cbcac6:pn-0x203d00a098cbcac6 PortID:01050d
```

使用这些 **host-traddr** 和 **traddr** 值，找到子系统 NVMe 限定名称(NQN)：

```
# nvme discover --transport fc \ --traddr nn-0x203b00a098cbcac6:pn-
0x203d00a098cbcac6 \ --host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62

Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0======
trtype:  fc
```

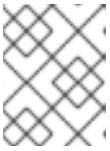
```
adrfam:  fibre-channel
subtype: nvme subsystem
treq:    not specified
portid:  0
trsvcid: none
subnqn:  nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_su
bsystem_468
traddr:  nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6
```

将 *nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6* 替换为 **traddr**。

将 *nn-0x20000024ff19bb62:pn-0x21000024ff19bb62* 替换为 **host-traddr**。

4. 使用 **nvme-cli** 工具连接到 NVMe 控制器：

```
# nvme connect --transport fc \ --traddr nn-0x203b00a098cbcac6:pn-
0x203d00a098cbcac6 \ --host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 \ -
n nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipat
h_1_subsystem_468\ -k 5
```



注意

如果您在连接时间超过默认的 keep-alive timeout 值时看到 **keep-alive timer (5 seconds) expired!** 错误，请使用 **-k** 选项增加它。例如，您可以使用 **-k 7**。

在这里，

将 *nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6* 替换为 **traddr**。

将 *nn-0x20000024ff19bb62:pn-0x21000024ff19bb62* 替换为 **host-traddr**。

将 *nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsyste*
替换为 ***subnqn***。

将 *5* 替换为 keep-live timeout 值（以秒为单位）。

验证

- 列出当前连接的 NVMe 设备：

```
# nvme list
Node          SN                   Model                                    Namespace Usage
Format          FW Rev
----------------- -------------------- ---------------------------------------- --------- -------------------------
- ---------------- --------
/dev/nvme0n1     80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller                   1
107.37  GB / 107.37  GB      4 KiB +  0 B   FFFFFFFF

# lsblk |grep nvme
nvme0n1                 259:0    0   100G  0 disk
```

其他资源

- **nvme(1)** 手册页
- Nvme-cli Github 软件仓库

12.4. 后续步骤

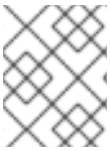
- 在 NVMe 设备 上启用多路径。

第 13 章 使用 NVME/TCP 配置 NVME OVER FABRICS

在 TCP (NVMe/TCP)设置上的 Non-volatile Memory Express™(NVMe/TCP)中，主机模式被完全支持，控制器设置不被支持。

作为系统管理员，完成以下部分中的任务来部署 NVMe/TCP 设置：

- 配置 NVMe/TCP 主机
- 将 NVMe/TCP 主机连接到 NVMe/TCP 控制器



注意

在 Red Hat Enterprise Linux 9 中，默认启用原生 NVMe 多路径。NVMe/TCP 不支持启用 DM 多路径。

13.1. NVME OVER FABRIC 设备概述

Non-volatile Memory Express™(NVMe™)是一个接口，允许主机软件工具与固态硬盘进行通信。

使用以下类型的光纤传输来通过光纤设备配置 NVMe：

NVMe over Remote Direct Memory Access(NVMe/RDMA)

有关如何配置 NVMe™/RDMA 的详情，请参考 使用 NVMe/RDMA 配置光纤上的 NVMe。

NVMe over Fibre Channel (NVMe/FC)

有关如何配置 NVMe™/FC 的详情，请参考 使用 NVMe/FC 配置光纤上的 NVMe。

NVMe over TCP (NVMe/TCP)

有关如何配置 NVMe/FC 的详情，请参考 使用 NVMe/TCP 配置光纤上的 NVMe。

当使用 NVMe over fabrics 时，固态驱动器不必位于您的系统本地，它可以通过 NVMe over fabrics 设备远程配置。

13.2. 配置 NVME/TCP 主机

使用 Non-volatile Memory Express™(NVMe™)管理命令行界面(nvme-cli)工具配置 NVMe/TCP 主机。

流程

1. 安装 **nvme-cli** 工具：

```
# dnf install nvme-cli
```

此工具在 **/etc/nvme/** 目录中创建 **hostnqn** 文件，该文件标识 NVMe 主机。

2. 找到 nvme **hostid** 和 **hostnqn** ：

```
# cat /etc/nvme/hostnqn
nqn.2014-08.org.nvmexpress:uuid:8ae2b12c-3d28-4458-83e3-658e571ed4b8

# cat /etc/nvme/hostid
09e2ce17-ccc9-412d-8dcf-2b0a1d581ee3
```

使用 **hostid** 和 **hostn** 值配置 NVMe/TCP 控制器。

3. 检查控制器的状态：

```
# nmcli device show ens6
GENERAL.DEVICE:                         ens6
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         52:57:02:12:02:02
GENERAL.MTU:                            1500
GENERAL.STATE:                          30 (disconnected)
GENERAL.CONNECTION:                     --
GENERAL.CON-PATH:                       --
WIRED-PROPERTIES.CARRIER:               on
```

4. 使用静态 IP 地址为新安装的以太网控制器配置主机网络：

```
# nmcli connection add con-name ens6 ifname ens6 type ethernet ip4 192.168.101.154/24
gw4 192.168.101.1
```

此处，将 *192.168.101.154* 替换为主机 IP 地址。

```
# nmcli connection mod ens6 ipv4.method manual
# nmcli connection up ens6
```

因为创建了一个新网络来将 NVMe/TCP 主机连接到 NVMe/TCP 控制器，因此也在控制器上执行这个步骤。

验证

- 验证新创建的主机网络是否正常工作：

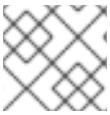
```
# nmcli device show ens6
GENERAL.DEVICE:                         ens6
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         52:57:02:12:02:02
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     ens6
GENERAL.CON-PATH:                       /org/freedesktop/NetworkManager/ActiveConnection/5
WIRED-PROPERTIES.CARRIER:               on
IP4.ADDRESS[1]:                         192.168.101.154/24
IP4.GATEWAY:                            192.168.101.1
IP4.ROUTE[1]:                           dst = 192.168.101.0/24, nh = 0.0.0.0, mt = 101
IP4.ROUTE[2]:                           dst = 192.168.1.1/32, nh = 0.0.0.0, mt = 101
IP4.ROUTE[3]:                           dst = 0.0.0.0/0, nh = 192.168.1.1, mt = 101
IP6.ADDRESS[1]:                         fe80::27ce:dde1:620:996c/64
IP6.GATEWAY:                            --
IP6.ROUTE[1]:                           dst = fe80::/64, nh = ::, mt = 101
```

其他资源

- **nvme (1)** man page

13.3. 将 NVME/TCP 主机连接到 NVME/TCP 控制器

连接 TCP (NVMe/TCP)主机上的 NVMe/TCP 控制器系统，以验证 NVMe/TCP 主机现在是否可以访问命名空间。



注意

不支持 NVMe/TCP 控制器 (**nvmet_tcp**) 模块。

先决条件

- 您已配置了 NVMe/TCP 主机。如需更多信息，请参阅配置 NVMe/TCP 主机。
- 已使用外部存储软件配置了 NVMe/TCP 控制器，并配置了控制器中的网络。在此过程中，*192.168.101.55* 是 NVMe/TCP 控制器的 IP 地址。

流程

1. 如果还没有载入 **nvme_tcp** 模块：

```
# modprobe nvme_tcp
```

2. 在 NVMe 控制器中发现可用的子系统：

```
# nvme discover --transport=tcp --traddr=192.168.101.55 --host-traddr=192.168.101.154 --
trsvcid=8009

Discovery Log Number of Records 2, Generation counter 7
=====Discovery Log Entry 0======
trtype:  tcp
adrfam:  ipv4
subtype: current discovery subsystem
treq: not specified, sq flow control disable supported
portid:  2
trsvcid: 8009
subnqn:  nqn.2014-08.org.nvmexpress.discovery
traddr:  192.168.101.55
eflags:  not specified
sectype: none
=====Discovery Log Entry 1======
trtype:  tcp
adrfam:  ipv4
subtype: nvme subsystem
treq: not specified, sq flow control disable supported
portid:  2
trsvcid: 8009
subnqn:  nqn.2014-08.org.nvmexpress:uuid:0c468c4d-a385-47e0-8299-6e95051277db
traddr:  192.168.101.55
eflags:  not specified
sectype: none
```

   在这里，*192.168.101.55* 是 NVMe/TCP 控制器 IP 地址， *192.168.101.154* 是 NVMe/TCP 主机 IP 地址。

3. 配置 **/etc/nvme/discovery.conf** 文件，以添加 **nvme discover** 命令中使用的参数：

```
# echo "--transport=tcp --traddr=192.168.101.55 --host-traddr=192.168.101.154 --
trsvcid=8009" >> /etc/nvme/discovery.conf
```

4. 将 NVMe/TCP 主机连接到控制器系统：

```
# nvme connect-all
```

验证

- 验证 NVMe/TCP 主机是否可以访问命名空间：

```
# nvme list-subsys

nvme-subsys3 - NQN=nqn.2014-08.org.nvmexpress:uuid:0c468c4d-a385-47e0-8299-
6e95051277db
\
 +- nvme3 tcp traddr=192.168.101.55,trsvcid=8009,host_traddr=192.168.101.154 live
optimized

# nvme list
Node             Generic          SN               Model                                    Namespace Usage
Format       FW Rev
---------------------- ---------------------- -------------------- ---------------------------------------- --------
- ---------------------------- ---------------- --------
/dev/nvme3n1     /dev/ng3n1       d93a63d394d043ab4b74 Linux                                    1
21.47  GB /  21.47  GB    512   B +  0 B   5.18.5-2
```

其他资源

- **nvme (1)** man page

第 14 章 在 NVME 设备中启用多路径

您可以通过光纤传输（如光纤通道(FC)）将连接到您的系统的 Non-volatile Memory Express™(NVMe™)设备多路径化。您可以在多个多路径解决方案之间进行选择。

14.1. 本地 NVME 多路径和 DM 多路径

Non-volatile Memory Express™(NVMe™)设备支持原生多路径功能。当在 NVMe 中配置多路径时，您可以在标准 DM 多路径和原生 NVMe 多路径之间进行选择。

DM 多路径和原生 NVMe 多路径都支持 NVMe 设备的 Asymmetric Namespace Access(ANA)多路径方案。ANA 识别控制器和主机之间的优化路径，并提高性能。

当启用原生 NVMe 多路径时，它会全局地应用于所有 NVMe 设备。它可以提供更高的性能，但不包含 DM 多路径提供的所有功能。例如，原生 NVMe 多路径只支持 **numa** 和 **round-robin** 路径选择方法。

默认情况下，Red Hat Enterprise Linux 9 中启用了 NVMe 多路径，也是推荐的多路径解决方案。

14.2. 在 NVME 设备中启用 DM 多路径

nvme_core.multipath 选项的默认内核设置被设置为 **Y**，这意味着启用了原生 Non-volatile Memory Express™(NVMe™)多路径。您可以通过禁用原生 NVMe 多路径在连接的 NVMe 设备上启用 DM 多路径。

先决条件

- NVMe 设备连接到您的系统。如需更多信息，请参阅 光纤设备上 NVMe 的概述。

流程

1. 检查是否启用了原生 NVMe 多路径：

```
# cat /sys/module/nvme_core/parameters/multipath
```

这个命令显示以下之一：

N
禁用原生 NVMe 多路径。

Y
启用原生 NVMe 多路径。

2. 如果启用了原生 NVMe 多路径，使用以下方法之一禁用它：

- 使用内核选项：

a. 在命令行中添加 **nvme_core.multipath=N** 选项：

```
# grubby --update-kernel=ALL --args="nvme_core.multipath=N"
```

b. 在 64 位 IBM Z 构架中更新引导菜单：

```
# zipl
```

c. 重启系统：

- 使用内核模块配置文件：

  a. 使用以下内容创建 **/etc/modprobe.d/nvme_core.conf** 配置文件：

  ```
  options nvme_core multipath=N
  ```

  b. 备份 **initramfs** 文件：

  ```
  # cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname r).bak.$(date
  +%m%d-%H%M%S).img
  ```

  c. 重建 **initramfs** ：

  ```
  # cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date
  +%m-%d-%H%M%S).img
  # dracut --force --verbose
  ```

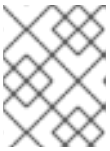
  d. 重启系统：

3. 启用 DM 多路径：

   ```
   # systemctl enable --now multipathd.service
   ```

4. 在所有可用路径中分发 I/O。在 **/etc/multipath.conf** 文件中添加以下内容：

   ```
   devices {
         device {
               vendor "NVME"
               product ".*"
               path_grouping_policy group_by_prio
         }
   }
   ```

   注意

   当 DM 多路径管理 NVMe 设备时，**/sys/class/nvme-subsys0/iopolicy** 配置文件不会影响 I/O 分发。

5. 重新载入 **multipathd** 服务以应用配置更改：

   ```
   # multipath -r
   ```

验证

- 验证是否已禁用了原生 NVMe 多路径：

  ```
  # cat /sys/module/nvme_core/parameters/multipath
  N
  ```

- 验证 DM 多路径是否可以识别 nvme 设备：

  ```
  # multipath -l
  ```

```
eui.00007a8962ab241100a0980000d851c8 dm-6 NVME,NetApp E-Series
size=20G features='0' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 0:10:2:2 nvme0n2 259:3 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
  |- 4:11:2:2 nvme4n2 259:28 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
  |- 5:32778:2:2 nvme5n2 259:38 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
  |- 6:32779:2:2 nvme6n2 259:44 active undef running
```

其他资源

- 配置内核命令行参数
- 配置 DM 多路径

14.3. 启用原生 NVME 多路径

如果禁用了原生 NVMe 多路径，您可以使用以下解决方案启用它。

先决条件

- NVMe 设备连接到您的系统。如需更多信息，请参阅 光纤设备上 NVMe 的概述。

流程

1. 检查内核中是否启用了原生 NVMe 多路径：

   ```
   # cat /sys/module/nvme_core/parameters/multipath
   ```

   这个命令显示以下之一：

   N
   禁用原生 NVMe 多路径。

   Y
   启用原生 NVMe 多路径。

2. 如果禁用了原生 NVMe 多路径，请使用以下方法之一启用它：

   - 使用内核选项：

     a. 从内核命令行中删除 **nvme_core.multipath=N** 选项：

     ```
     # grubby --update-kernel=ALL --remove-args="nvme_core.multipath=N"
     ```

     b. 在 64 位 IBM Z 构架中更新引导菜单：

     ```
     # zipl
     ```

     c. 重启系统：

- 使用内核模块配置文件：

  a. 删除 **/etc/modprobe.d/nvme_core.conf** 配置文件：

  ```
  # rm /etc/modprobe.d/nvme_core.conf
  ```

  b. 备份 **initramfs** 文件：

  ```
  # cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
  ```

  c. 重建 **initramfs** ：

  ```
  # dracut --force --verbose
  ```

  d. 重启系统：

3. 可选： 在运行的系统中，更改 NVMe 设备中的 I/O 策略，以便在所有可用路径中分发 I/O：

   ```
   # echo "round-robin" > /sys/class/nvme-subsystem/nvme-subsys0/iopolicy
   ```

4. 可选：使用 **udev** 规则永久设置 I/O 策略。使用以下内容创建 **/etc/udev/rules.d/71-nvme-io-policy.rules** 文件：

   ```
   ACTION=="add|change", SUBSYSTEM=="nvme-subsystem", ATTR{iopolicy}="round-robin"
   ```

验证

1. 验证您的系统是否可以识别 NVMe 设备。以下示例假设您有一个通过光纤存储子系统连接的 NVMe，它有两个 NVMe 命名空间：

   ```
   # nvme list

   Node             SN                   Model                                    Namespace Usage
   Format           FW Rev
   ---------------- -------------------- ---------------------------------------- --------- --------------------------
   - ---------------- --------
   /dev/nvme0n1     a34c4f3a0d6f5cec     Linux                                    1         250.06  GB /
   250.06  GB    512   B +  0 B   4.18.0-2
   /dev/nvme0n2     a34c4f3a0d6f5cec     Linux                                    2         250.06  GB /
   250.06  GB    512   B +  0 B   4.18.0-2
   ```

2. 列出所有连接的 NVMe 子系统：

   ```
   # nvme list-subsys

   nvme-subsys0 - NQN=testnqn
   \
    +- nvme0 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
    +- nvme1 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
   ```

```
 +- nvme2 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
 +- nvme3 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
```

检查活动传输类型。例如，**nvme0 fc** 表示设备通过光纤通道传输连接，**nvme tcp** 则表示设备通过 TCP 连接。

3. 如果您编辑了内核选项，请验证原生 NVMe 多路径是否在内核命令行上启用了：

```
# cat /proc/cmdline

BOOT_IMAGE=[...] nvme_core.multipath=Y
```

4. 如果您更改了 I/O 策略，请验证 **round-robin** 是否是 NVMe 设备上活跃的 I/O 策略：

```
# cat /sys/class/nvme-subsystem/nvme-subsys0/iopolicy

round-robin
```

其他资源

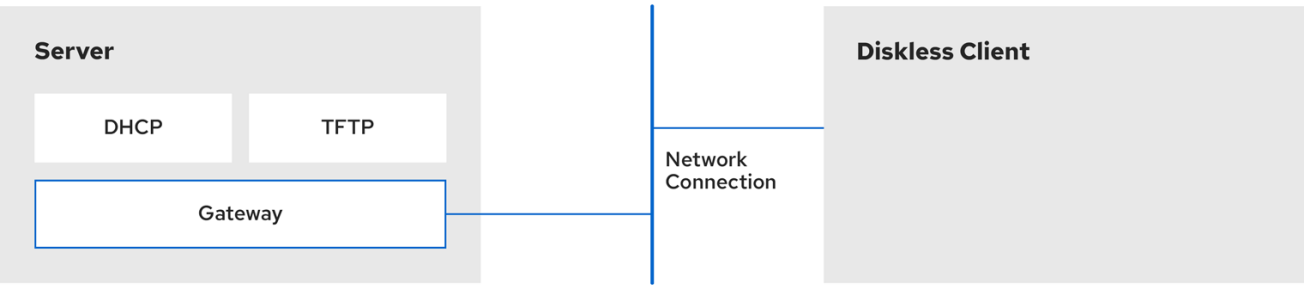
- 配置内核命令行参数

第 15 章 设置远程无盘系统

在网络环境中，您可以通过部署远程无盘系统来使用相同的配置设置多个客户端。通过使用当前的 Red Hat Enterprise Linux 服务器版本，您可以节省这些客户端的硬盘驱动器成本，并在单独的服务器上配置网关。

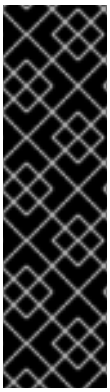
下图描述了通过动态主机配置协议(DHCP)和普通文件传输协议(TFTP)服务，无盘客户端与服务器的连接。

图 15.1. 远程无盘系统设置图



15.1. 为远程无盘系统准备环境

准备您的环境，以便继续远程无盘系统实现。远程无盘系统引导需要一个普通文件传输协议(TFTP)服务（由 **tftp-server**提供），以及一个动态主机配置协议(DHCP)服务（由 **dhcp** 提供）。系统使用 **tftp** 服务通过预启动执行环境(PXE)加载程序，通过网络检索内核镜像和初始 RAM 磁盘 **initrd**。



重要

要确保环境中远程无盘系统的正确功能，请按照以下顺序配置服务：

1. 用于无盘客户端的 **tftp** 服务
2. DHCP 服务器
3. 网络文件系统(NFS)
4. 导出的文件系统。

先决条件

- 您已设置了网络连接。

流程

1. 安装 **dracut-network** 软件包：

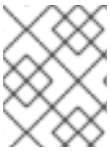
```
# dnf install dracut-network
```

2. 在 **/etc/dracut.conf.d/network.conf** 文件中添加以下行：

```
add_dracutmodules+=" nfs "
```

15.2. 为无盘客户端配置 TFTP 服务

要使远程无盘系统在您的环境中正常工作，您需要首先为无盘客户端配置普通文件传输协议(TFTP)服务。



注意

此配置不会通过统一可扩展固件接口(UEFI)引导。对于基于 UEFI 的安装，请参阅 为基于 UEFI 的客户端配置 TFTP 服务器。

先决条件

- 您已安装了以下软件包：
  - **tftp-server**
  - **syslinux**

流程

1. 启用 **tftp** 服务：

```
# systemctl enable --now tftp
```

2. 在 **tftp** 根目录中创建 **pxelinux** 目录：

```
# mkdir -p /var/lib/tftpboot/pxelinux/
```

3. 将 **/usr/share/syslinux/pxelinux.0** 文件复制到 **/var/lib/tftpboot/pxelinux/** 目录中：

```
# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/pxelinux/
```

   - 您可以在 **/var/lib/tftpboot** 目录中找到 tftp 根目录(**chroot**)。

4. 将 **/usr/share/syslinux/ldlinux.c32** 复制到 **/var/lib/tftpboot/pxelinux/** ：

```
# cp /usr/share/syslinux/ldlinux.c32 /var/lib/tftpboot/pxelinux/
```

5. 在 **tftp** 根目录中创建 **pxelinux.cfg** 目录：

```
# mkdir -p /var/lib/tftpboot/pxelinux/pxelinux.cfg/
```

   此配置不会通过统一可扩展固件接口(UEFI)引导。要为 UEFI 执行安装，请按照 为基于 UEFI 的客户端配置 TFTP 服务器 中的流程操作。

验证

- 检查服务 **tftp** 的状态：

```
# systemctl status tftp
...
Active: active (running)
...
```

15.3. 为无盘客户端配置 DHCP 服务器

远程无盘系统需要几个预安装的服务以启用正确的功能。首先，您需要安装普通文件传输协议(TFTP)服务，然后配置动态主机配置协议(DHCP)服务器。

先决条件

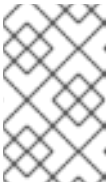
- 您已安装了以下软件包：
  - **dhcp-server**
- 您已为无盘客户端配置了 **tftp** 服务。请参阅 为无盘客户端配置 TFTP 服务 部分。

流程

1. 将配置添加到 **/etc/dhcp/dhcpd.conf** 文件中，以为引导设置 DHCP 服务器并启用预启动执行环境(PXE)：

```
option space pxelinux;
option pxelinux.magic code 208 = string;
option pxelinux.configfile code 209 = text;
option pxelinux.pathprefix code 210 = text;
option pxelinux.reboottime code 211 = unsigned integer 32;
option architecture-type code 93 = unsigned integer 16;

subnet 192.168.205.0 netmask 255.255.255.0 {
 option routers 192.168.205.1;
 range 192.168.205.10 192.168.205.25;

 class "pxeclients" {
  match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
  next-server 192.168.205.1;

  if option architecture-type = 00:07 {
   filename "BOOTX64.efi";
   } else {
   filename "pxelinux/pxelinux.0";
  }
 }
}
```

   - 您的 DHCP 配置可能随您的环境而异，如设置租期时间或固定地址。详情请参阅 提供 DHCP 服务。



注意

虽然将 **libvirt** 虚拟机用作无盘客户端，但 **libvirt** 守护进程回提供 DHCP 服务，并且不使用独立 DHCP 服务器。在这种情况下，必须使用 **libvirt** 网络配置中的 **bootp file=<filename>** 选项启用网络引导，**virsh net-edit**。

2. 启用 **dhcpd.service** ：

```
# systemctl enable --now dhcpd.service
```

验证

- 检查服务 **dhcpd.service** 的状态：

```
# systemctl status dhcpd.service
...
Active: active (running)
...
```

15.4. 为无盘客户端配置导出的文件系统

作为在您的环境中配置远程无盘系统的一部分，您必须为无盘客户端配置导出的文件系统。

先决条件

- 您已为无盘客户端配置了 **tftp** 服务。请参阅 为无盘客户端配置 TFTP 服务 部分。
- 您已配置了动态主机配置协议(DHCP)服务器。请参阅 为无盘客户端配置 DHCP 服务器 部分。

流程

1. 通过将根目录添加到 **/etc/exports** 目录，将网络文件系统(NFS)服务器配置为导出根目录。有关完整的说明集，请参阅
   - 部署 NFS 服务器
2. 将 Red Hat Enterprise Linux 的完整版本安装到根目录，以容纳整个无盘客户端。为此，您可以安装新的基础系统或克隆现有的安装。
   - 通过使用导出的文件系统的路径替换导出的根目录，来将 Red Hat Enterprise Linux 安装到导出的位置：

     ```
     # dnf install @Base kernel dracut-network nfs-utils --installroot=exported-root-directory --releasever=/
     ```

     通过将 **releasever** 选项设置为 /，可以从主机(/)系统中检测到 releasever。
   - 使用 **rsync** 工具与正在运行的系统同步：

     ```
     # rsync -a -e ssh --exclude='/proc/' --exclude='/sys/' example.com:/ exported-root-directory
     ```

     - 使用要通过 **rsync** 工具同步的正在运行的系统的主机名替换 *example.com*。
     - 使用导出的文件系统的路径替换 *exported-root-directory*。
       请注意，对于这个选项，您必须有一个单独的现有运行系统，通过以上命令将其克隆到服务器。

您需要完全配置准备用于导出的文件系统，然后才能将其用于无盘客户端。按照以下流程完成配置。

配置文件系统

1. 将无盘客户端支持的内核(**vmlinuz-_kernel-version_pass:attributes**)复制到 **tftp** 引导目录中：

   ```
   # cp /exported-root-directory/boot/vmlinuz-kernel-version /var/lib/tftpboot/pxelinux/
   ```

2. 在本地创建 **initramfs-*kernel-version*.img** 文件，并将其移到具有 NFS 支持的导出的根目录中：

```
# dracut --add nfs initramfs-kernel-version.img kernel-version
```

例如：

```
# dracut --add nfs /exports/root/boot/initramfs-5.14.0-202.el9.x86_64.img 5.14.0-202.el9.x86_64
```

创建 initrd 的示例，使用当前运行的内核版本，并覆盖现有镜像：

```
# dracut -f --add nfs "boot/initramfs-$(uname -r).img" "$(uname -r)"
```

3. 将 **initrd** 的文件权限改为 **0644** ：

```
# chmod 0644 /exported-root-directory/boot/initramfs-kernel-version.img
```

警告

如果您不更改 **initrd** 文件权限，则 **pxelinux.0** 引导装载程序会失败，并显示 "file not found" 错误。

4. 将生成的 **initramfs-*kernel-version*.img** 文件复制到 **tftp** 引导目录中：

```
# cp /exported-root-directory/boot/initramfs-kernel-version.img
/var/lib/tftpboot/pxelinux/
```

5. 在 **/var/lib/tftpboot/pxelinux/pxelinux.cfg/default** 文件中添加以下配置，来编辑默认引导配置，以使用 **initrd** 和内核：

```
default rhel9

label rhel9
  kernel vmlinuz-kernel-version
  append initrd=initramfs-kernel-version.img root=nfs:_server-ip_:/exported-root-directory rw
```

- 此配置指示无盘客户端 root 以读/写格式挂载导出的文件系统(**/exported-root-directory**)。

6. 可选：使用以下配置编辑 **/var/lib/tftpboot/pxelinux/pxelinux.cfg/default** 文件，来以 *只读* 格式挂载文件系统：

```
default rhel9

label rhel9
  kernel vmlinuz-kernel-version
  append initrd=initramfs-kernel-version.img root=nfs:server-ip:/exported-root-directory ro
```

7. 重启 NFS 服务器：

```
# systemctl restart nfs-server.service
```

现在，您可以将 NFS 共享导出到无盘客户端。这些客户端可以通过预引导执行环境(PXE)通过网络引导。

15.5. 重新配置远程无盘系统

如果要安装软件包更新、服务重启或调试问题，您可以重新配置系统。以下步骤演示了如何更改用户的密码、如何在系统上安装软件，描述如何将系统分成只读模式的 **/usr**，以及读写模式的 **/var**。

先决条件

- 您已在导出的文件系统中启用了 **no_root_squash** 选项。

流程

1. 要更改用户密码，请按照以下步骤执行：
   - 将命令行改为 */exported/root/directory*:

     ```
     # chroot /exported/root/directory /bin/bash
     ```

   - 更改您想要的用户的密码：

     ```
     # passwd <username>
     ```

     使用您要更改密码的实际用户替换 *<username>*。

   - 退出命令行。
2. 在远程无盘系统上安装软件：

   ```
   # dnf install <package> --installroot=/exported/root/directory --releasever=/ --config
   /etc/dnf/dnf.conf --setopt=reposdir=/etc/yum.repos.d/
   ```

   - 用要安装的实际软件包替换 *<package>*。
3. 配置两个单独的导出，来将远程无盘系统分成 **/usr** 和 **/var**。查看
   - 部署 NFS 服务器

15.6. 处理加载远程无盘系统的常见问题

根据之前的配置，在加载远程无盘系统时可能会出现一些问题。以下是在 Red Hat Enterprise Linux 服务器上最常见问题以及故障排除方法的一些示例。

例 15.1. 客户端无法获得 IP 地址

- 检查是否在服务器上启用了动态主机配置协议(DHCP)服务。
  - 检查 **dhcp.service** 是否在运行：

    ```
    # systemctl status dhcpd.service
    ```

- 如果 **dhcp.service** 不活跃，您必须启用并启动它：

```
# systemctl enable dhcpd.service
# systemctl start dhcpd.service
```

- 重启无磁盘客户端。
- 检查 DHCP 配置文件 **/etc/dhcp/dhcpd.conf**。详情请参阅为 为无盘客户端配置 DHCP 服务器。

- 检查防火墙端口是否已打开。
  - 检查 **dhcp.service** 是否在活跃服务中列出：

```
# firewall-cmd --get-active-zones
# firewall-cmd --info-zone=public
```

  - 如果 **dhcp.service** 没有列在活跃的服务中，请将其添加到列表中：

```
# firewall-cmd --add-service=dhcp --permanent
```

  - 检查 **nfs.service** 是否在活跃服务中列出：

```
# firewall-cmd --get-active-zones
# firewall-cmd --info-zone=public
```

  - 如果 **nfs.service** 没有列在活跃的服务中，将其添加到列表中：

```
# firewall-cmd --add-service=nfs --permanent
```

例 15.2. 在引导远程无盘系统的过程中，该文件不可用

1. 检查该文件是否在 **/var/lib/tftpboot/** 目录中。
2. 如果该文件在目录中，请检查权限：

```
# chmod 644 pxelinux.0
```

3. 检查防火墙端口是否已打开。

例 15.3. 载入 内核/initrd后系统引导失败

1. 检查是否在服务器上启用了 NFS 服务。
   a. 检查 **nfs.service** 是否正在运行：

```
# systemctl status nfs.service
```

   b. 如果 **nfs.service** 不活跃，您必须启动并启用它：

```
# systemctl start nfs.service
# systemctl enable nfs.service
```

2. 检查 **/var/lib/tftpboot/pxelinux.cfg/** 目录中的参数是否正确。详情请参阅 为无盘客户端配置导出的文件系统。

3. 检查防火墙端口是否已打开。

第 16 章 SWAP 入门

使用交换空间为不活跃的进程和数据提供临时存储，并在物理内存已满时防止出现内存不足的错误。交换空间充当物理内存的扩展，并允许系统平稳运行，即使物理内存已耗尽。请注意，使用交换空间可能会降低系统性能，因此在依赖交换空间之前优化物理内存的使用可能更好。

16.1. SWAP 空间概述

当物理内存(RAM)已满时，将使用 Linux 中的 *交换空间*。如果系统需要更多的内存资源并且 RAM 已满，内存中的不活动页面将移到交换空间。虽然交换空间可以帮助具有少量 RAM 的计算机，但不应将其视为更多 RAM 的替代品。

交换空间位于硬盘驱动器上，其访问时间比物理内存要慢。交换空间可以是专用的交换分区（推荐）、交换文件，或者交换分区和交换文件的组合。

过去数年，推荐的 swap 空间会随系统中的 RAM 量增加而线性增大。然而，现代系统通常包含了成百 GB 内存。因此，推荐的交换空间被视为系统内存工作负载的功能，而不是系统内存的功能。

添加 swap 空间

以下是添加 swap 空间的不同方法：

- 在 LVM2 逻辑卷中扩展 swap
- 为 swap 创建 LVM2 逻辑卷
- 创建交换文件
  例如，您可以将系统中的 RAM 量从 1 GB 升级到 2 GB，但只有 2 GB 的交换空间。如果您执行内存密集型操作或运行需要大量内存的应用程序，则最好将交换空间大小增加到 4 GB。

删除 swap 空间

以下是删除 swap 空间的不同方法：

- 在 LVM2 逻辑卷中减少 swap
- 为 swap 删除 LVM2 逻辑卷
- 删除交换文件
  例如：您将系统中的 RAM 大小从 1GB 降到 512MB，但仍分配了 2GB swap 空间。最好将交换空间大小减少到 1 GB，因为较大的 2 GB 可能会浪费磁盘空间。

16.2. 推荐的系统 SWAP 空间

推荐的 swap 分区的大小取决于系统中的 RAM 量，以及是否有足够的内存供系统休眠。推荐的 swap 分区大小在安装过程中自动设置。但是，为了允许休眠，您需要在自定义分区阶段编辑交换空间。

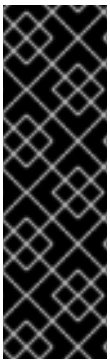
以下建议对于有较少内存（如 1 GB 或更小）的系统特别重要。无法在这些系统上分配足够的 swap 空间可能会导致问题，如不稳定，甚至会导致安装的系统无法引导。

表 16.1. 推荐的 swap 空间

系统中的 RAM 量	推荐的 swap 空间	如果允许休眠则推荐使用 swap 空间
⩽ 2 GB	RAM 量的 2 倍	RAM 量的 3 倍
> 2 GB – 8 GB	与 RAM 量相等	RAM 量的 2 倍
> 8 GB – 64 GB	至少 4 GB	RAM 量的 1.5 倍
> 64 GB	至少 4 GB	不推荐休眠

对于边界值，如 2 GB、8 GB 或 64 GB 系统 RAM，请根据您的需要或偏好选择 swap 大小。如果您的系统资源允许此操作，增加 swap 空间可提高性能。

请注意，将交换空间分布到多个存储设备也可以提高交换空间的性能，特别是在具有快速驱动器、控制器和接口的系统上。



重要

在修改时，*不应* 使用分配被为交换空间的文件系统和 LVM2 卷。如果系统进程或内核正在使用交换空间，则任何修改交换的尝试都会失败。使用 **free** 和 **cat /proc/swaps** 命令验证交换的使用量以及位置。

重新调整 swap 空间大小需要临时将其从系统中删除。如果运行的应用程序依赖于额外的 swap 空间，且可能会遇到低内存情况，这可能会出现问题。最好是，在救援模式中执行 swap 大小调整，请参阅*执行高级 RHEL 9 安装中的* 的 Debug 引导选项。当提示挂载文件系统时，请选择 **Skip**。

16.3. 为 SWAP 创建 LVM2 逻辑卷

您可以为 swap 创建一个 LVM2 逻辑卷。假设 */dev/VolGroup00/LogVol02* 是您要添加的交换卷。

先决条件

- 您有足够的磁盘空间。

流程

1. 创建大小为 *2 GB* 的 LVM2 逻辑卷：

```
# lvcreate VolGroup00 -n LogVol02 -L 2G
```

2. 格式化新 swap 空间：

```
# mkswap /dev/VolGroup00/LogVol02
```

3. 在 **/etc/fstab** 文件中添加以下条目：

```
/dev/VolGroup00/LogVol02 none swap defaults 0 0
```

4. 重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

5. 在逻辑卷中激活 swap：

```
# swapon -v /dev/VolGroup00/LogVol02
```

验证

- 要测试是否成功创建并激活 swap 逻辑卷，请使用以下命令检查活跃 swap 空间：

```
# cat /proc/swaps
              total        used        free      shared  buff/cache   available
Mem:           30Gi       1.2Gi        28Gi        12Mi       994Mi        28Gi
Swap:          22Gi          0B        22Gi
```

```
# free -h
              total        used        free      shared  buff/cache   available
Mem:           30Gi       1.2Gi        28Gi        12Mi       995Mi        28Gi
Swap:          17Gi          0B        17Gi
```

16.4. 创建交换文件

当系统内存不足时，您可以创建一个交换文件来在固态驱动器或硬盘上创建一个临时存储空间。

先决条件

- 您有足够的磁盘空间。

流程

1. 以 MB 为单位确定新交换文件的大小，再乘以 1024 来确定块的数量。例如： 64MB swap 文件的块大小为 65536。
2. 创建一个空文件：

```
# dd if=/dev/zero of=/swapfile bs=1024 count=65536
```

将 *65536* 替换为等于所需块大小的值。

3. 使用以下命令设定 swap 文件：

```
# mkswap /swapfile
```

4. 更改交换文件的安全性，使其不可读。

```
# chmod 0600 /swapfile
```

5. 使用以下条目编辑 **/etc/fstab** 文件，以在引导时启用交换文件：

```
/swapfile none swap defaults 0 0
```

下次系统引导时，它会激活新的 swap 文件。

6. 重新生成挂载单元，以便您的系统注册新的 **/etc/fstab** 配置：

```
# systemctl daemon-reload
```

7. 立即激活 swap 文件：

```
# swapon /swapfile
```

验证

- 要测试新 swap 文件是否已成功创建并激活，请使用以下命令检查活跃 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

16.5. 在 LVM2 逻辑卷中扩展 SWAP

您可以在现有 LVM2 逻辑卷上扩展 swap 空间。假设 */dev/VolGroup00/LogVol01* 是您要将其扩展为 *2 GB* 的卷。

先决条件

- 您有足够的磁盘空间。

流程

1. 为关联的逻辑卷禁用交换：

```
# swapoff -v /dev/VolGroup00/LogVol01
```

2. 将 LVM2 逻辑卷调整为 *2 GB* ：

```
# lvresize /dev/VolGroup00/LogVol01 -L +2G
```

3. 格式化新 swap 空间：

```
# mkswap /dev/VolGroup00/LogVol01
```

4. 启用扩展的逻辑卷：

```
# swapon -v /dev/VolGroup00/LogVol01
```

验证

- 要测试是否成功扩展并激活 swap 逻辑卷，请检查活跃 swap 空间：

```
# cat /proc/swaps
Filename                Type        Size        Used        Priority
/dev/dm-1               partition   16322556    0           -2
/dev/dm-4               partition   7340028     0           -3
```

```
# free -h
              total        used        free      shared  buff/cache   available
Mem:           30Gi       1.2Gi        28Gi        12Mi       994Mi        28Gi
Swap:          22Gi          0B        22Gi
```

16.6. 在 LVM2 逻辑卷中减少 SWAP

您可以在 LVM2 逻辑卷上减少 swap。假设 */dev/VolGroup00/LogVol01* 是您要缩小的卷。

流程

1. 为关联的逻辑卷禁用交换：

```
# swapoff -v /dev/VolGroup00/LogVol01
```

2. 清除交换签名：

```
# wipefs -a /dev/VolGroup00/LogVol01
```

3. 将 LVM2 逻辑卷减少 512 MB：

```
# lvreduce /dev/VolGroup00/LogVol01 -L -512M
```

4. 格式化新 swap 空间：

```
# mkswap /dev/VolGroup00/LogVol01
```

5. 在逻辑卷中激活 swap：

```
# swapon -v /dev/VolGroup00/LogVol01
```

验证

- 要测试是否成功缩小 swap 逻辑卷，请使用以下命令检查活跃的 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

16.7. 删除 SWAP 的 LVM2 逻辑卷

您可以删除 swap 的 LVM2 逻辑卷。假设 */dev/VolGroup00/LogVol02* 是您要删除的交换卷。

流程

1. 为关联的逻辑卷禁用交换：

```
# swapoff -v /dev/VolGroup00/LogVol02
```

2. 删除 LVM2 逻辑卷：

```
# lvremove /dev/VolGroup00/LogVol02
```

3. 从 **/etc/fstab** 文件中删除以下关联的条目：

```
/dev/VolGroup00/LogVol02 none swap defaults 0 0
```

4. 重新生成挂载单元以注册新配置：

```
# systemctl daemon-reload
```

验证

- 使用以下命令测试逻辑卷是否已成功删除，检查活跃的 swap 空间：

```
$ cat /proc/swaps
$ free -h
```

16.8. 删除 SWAP 文件

您可以删除 swap 文件。

流程

1. 禁用 **/swapfile** swap 文件：

```
# swapoff -v /swapfile
```

2. 相应地从 **/etc/fstab** 文件中删除其条目。
3. 重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

4. 删除实际的文件：

```
# rm /swapfile
```

第 17 章 使用以太网配置光纤

根据 IEEE T11 FC-BB-5 标准，使用以太网（FCoE）的光纤通道是通过以太网传输光纤通道帧的协议。通常数据中心有一个专用的 LAN 和 Storage Area Network（SAN），它和它们自己的配置是相互分开的。FCoE 将这些网络合并为一个整合的网络结构。例如 FCoE 的优点是降低硬件和能源成本。

17.1. 在 RHEL 中使用硬件 FCOE HBA

在 RHEL 中，您可以使用以太网的硬件光纤通道(FCoE)主机总线适配器(HBA)，这些驱动程序支持以下驱动程序：

- **qedf**
- **bnx2fc**
- **fnic**

如果您使用这样的 HBA，在 HBA 设置中配置 FCoE 设置。如需更多信息，请参阅适配器文档。

配置 HBA 后，从 Storage Area Network(SAN)中导出的逻辑单元号(LUN)将自动用于 RHEL 作为 **/dev/sd*** 设备。您可以使用类似本地存储设备的设备。

17.2. 设置软件 FCOE 设备

使用软件 FCoE 设备通过 FCoE 访问逻辑单元号(LUN)，它使用部分支持 FCoE 卸载的以太网适配器。

重要

RHEL 不支持需要 **fcoe.ko** 内核模块的软件 FCoE 设备。

完成此步骤后，RHEL 会自动从 Storage Area Network(SAN)导出的 LUN 作为 **/dev/sd*** 设备。您可以使用类似本地存储设备的方法使用这些设备。

先决条件

- 您已将网络交换机配置为支持 VLAN。
- SAN 使用 VLAN 来将存储流量与普通以太网流量分开。
- 您已在其 BIOS 中配置了服务器的 HBA。
- HBA 连接至网络，连接是在线的。如需更多信息，请参阅您的 HBA 文档。

步骤

1. 安装 **fcoe-utils** 软件包：

```
# dnf install fcoe-utils
```

2. 将 **/etc/fcoe/cfg-ethx** 模板文件复制到 **/etc/fcoe/cfg-*interface_name***。例如，如果要将 **enp1s0** 接口配置为使用 FCoE，请输入以下命令：

```
# cp /etc/fcoe/cfg-ethx /etc/fcoe/cfg-enp1s0
```

3. 启用并启动 **fcoe** 服务：

```
# systemctl enable --now fcoe
```

4. 在接口 **enp1s0** 中发现 FCoE VLAN，为发现的 VLAN 创建网络设备，并启动启动器：

```
# fipvlan -s -c enp1s0
Created VLAN device enp1s0.200
Starting FCoE on interface enp1s0.200
Fibre Channel Forwarders Discovered
interface       | VLAN | FCF MAC
------------------------------------------
enp1s0          | 200  | 00:53:00:a7:e7:1b
```

5. 可选：显示发现的目标、LUN 和与 LUN 关联的设备详情：

```
# fcoeadm -t
Interface:       enp1s0.200
Roles:           FCP Target
Node Name:       0x500a0980824acd15
Port Name:       0x500a0982824acd15
Target ID:       0
MaxFrameSize:    2048 bytes
OS Device Name:  rport-11:0-1
FC-ID (Port ID): 0xba00a0
State:           Online

LUN ID  Device Name   Capacity   Block Size  Description
------  -----------  ----------  ----------  ----------------------
    0  sdb           28.38 GiB      512     NETAPP LUN (rev 820a)
    ...
```

本例显示 SAN 中的 LUN 0 已作为 **/dev/sdb** 设备附加到主机中。

验证

- 显示所有活跃 FCoE 接口的信息：

```
# fcoeadm -i
Description:     BCM57840 NetXtreme II 10 Gigabit Ethernet
Revision:        11
Manufacturer:    Broadcom Inc. and subsidiaries
Serial Number:   000AG703A9B7

Driver:          bnx2x Unknown
Number of Ports: 1

   Symbolic Name:    bnx2fc (QLogic BCM57840) v2.12.13 over enp1s0.200
   OS Device Name:   host11
   Node Name:        0x2000000af70ae935
   Port Name:        0x2001000af70ae935
   Fabric Name:      0x20c8002a6aa7e701
   Speed:            10 Gbit
   Supported Speed:  1 Gbit, 10 Gbit
```

```
MaxFrameSize:      2048 bytes
FC-ID (Port ID):   0xba02c0
State:             Online
```

其他资源

- **fcoeadm(8)** 手册页
- **/usr/share/doc/fcoe-utils/README**
- 使用光纤通道设备

第 18 章 管理磁带设备

磁带设备是保存数据并按顺序访问的磁带。使用磁带驱动器将数据写入此磁带设备。不需要创建文件系统来存储磁带设备中的数据。可以使用各种接口（如 SCSI、FC、USB、SATA 和其他接口）连接到主机计算机。

18.1. 磁带设备的类型

以下是不同类型的磁带设备列表：

- **/dev/st0** 是一个回卷磁带设备。
- **/dev/nst0** 是一个非回卷磁带设备。使用非缓解设备进行日常备份。

使用磁带设备有几个优点。它们成本效益高且稳定。磁带设备也对数据崩溃具有弹性，并适合长久保存数据。

18.2. 安装磁带驱动器管理工具

使用 **mt** 命令返回数据。**mt** 实用程序控制 magnetic 磁带驱动器操作，而 **st** 程序则用于 SCSI 磁带驱动程序。这个步骤描述了如何为磁带驱动器操作安装 **mt-st** 软件包。

流程

- 安装 **mt-st** 软件包：

```
# dnf install mt-st
```

其他资源

- **mt(1)** 和 **st(4)** man pages

18.3. 写入如回卷磁带设备

在每次操作后，回卷磁带设备会进行回卷。要备份数据，您可以使用 **tar** 命令。默认情况下，在磁带设备中，块大小为 10KB(**bs=10k**)。您可以使用 **export TAPE=*/dev/st0*** 属性设置 **TAPE** 环境变量。使用 **-f** 设备选项指定磁带设备文件。当您使用多个磁带设备时，这个选项很有用。

先决条件

1. 已安装 **mt-st** 软件包。如需更多信息，请参阅安装磁带驱动器管理工具。
2. 加载磁带驱动器：

```
# mt -f /dev/st0 load
```

流程

1. 检查磁带头：

```
# mt -f /dev/st0 status
```

```
SCSI 2 tape drive:
File number=-1, block number=-1, partition=0.
Tape block size 0 bytes. Density code 0x0 (default).
Soft error count since last status=0
General status bits on (50000):
 DR_OPEN IM_REP_EN
```

在这里：

- 当前**文件号**为 -1。
- **块编号** 定义磁带头。默认情况下，它被设置为 -1。
- **块大小** 0 表示磁带设备没有固定的块大小。
- **软错误数**表示在执行 mt status 命令后遇到的错误数量。
- **General status 位**解释了磁带设备的状态。
- **DR_OPEN** 表示公开，磁带设备为空。**IM_REP_EN** 是即时报告模式。

2. 如果磁带设备不是空的，覆盖它：

```
# tar -czf /dev/st0 _/source/directory
```

该命令使用 ***/source/directory*** 目录的内容覆盖 tape 设备中的数据

3. 将 ***/source/directory*** 目录备份到磁带设备中：

```
# tar -czf /dev/st0 _/source/directory
tar: Removing leading `/' from member names
/source/directory
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

4. 查看磁带设备的状态：

```
# mt -f /dev/st0  status
```

验证步骤

- 查看磁带设备上的所有文件列表：

```
# tar -tzf /dev/st0
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

其他资源

- **mt(1)**, **st(4)**, 和 **tar(1)** man pages
- 磁带驱动器介质被识别为带有写保护 Red Hat 知识库文章
- 如何在系统 Red Hat Knowlegebase 文章中探测到磁带驱动器

18.4. 写入非回卷解磁带设备

在完成特定命令后，非回卷磁带设备会将磁带保持在当前位置。例如，备份后，您可以将更多数据附加到非回卷的磁带设备中。您还可以使用它来避免任何意外的回卷。

先决条件

1. 已安装 **mt-st** 软件包。如需更多信息，请参阅安装磁带驱动器管理工具。
2. 加载磁带驱动器：

```
# mt -f /dev/nst0 load
```

流程

1. 检查非回卷磁带设备 **/dev/nst0** 的磁带头：

```
# mt -f /dev/nst0 status
```

2. 指定位于头或磁带末尾的指针：

```
# mt -f /dev/nst0 rewind
```

3. 附加磁带设备中的数据：

```
# mt -f /dev/nst0 eod
# tar -czf /dev/nst0 /source/directory/
```

4. 将 ***/source/directory/*** 备份到磁带设备中：

```
# tar -czf /dev/nst0 /source/directory/
tar: Removing leading `/' from member names
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

5. 查看磁带设备的状态：

```
# mt -f /dev/nst0  status
```

验证步骤

- 查看磁带设备上的所有文件列表：

```
# tar -tzf /dev/nst0
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

其他资源

- **mt(1)**, **st(4)**, 和 **tar(1)** man pages
- 磁带驱动器介质被识别为带有写保护 Red Hat 知识库文章
- 如何在系统 Red Hat Knowlegebase 文章中探测到磁带驱动器

18.5. 在磁带设备中切换磁带头

使用以下步骤切换磁带设备中的磁带头。

先决条件

1. 已安装 **mt-st** 软件包。如需更多信息，请参阅安装磁带驱动器管理工具。
2. 数据被写入磁带设备。如需更多信息，请参阅写入回卷磁带设备，或写入非回卷磁带设备。

流程

- 查看磁带指针的当前位置：

```
# mt -f /dev/nst0 tell
```

- 在将数据附加到磁带设备时切换磁带头：

```
# mt -f /dev/nst0 eod
```

- 使用之前的记录：

```
# mt -f /dev/nst0 bsfm 1
```

- 转至正向记录：

```
# mt -f /dev/nst0 fsf 1
```

其他资源

- **mt(1)** man page

18.6. 从磁带设备中恢复数据

要从磁带设备中恢复数据，请使用 **tar** 命令。

先决条件

1. 已安装 **mt-st** 软件包。如需更多信息，请参阅安装磁带驱动器管理工具。
2. 数据被写入磁带设备。如需更多信息，请参阅写入回卷磁带设备，或写入非回卷磁带设备。

流程

- 对于回卷磁带设备 **/dev/st0**：
  - 恢复 ***/source/directory/*** ：

```
# tar -xzf /dev/st0 /source/directory/
```

- 对于非回卷磁带设备 **/dev/nst0** ：
  - 重新构建磁带设备：

```
# mt -f /dev/nst0 rewind
```

  - 恢复 **etc** 目录：

```
# tar -xzf /dev/nst0 /source/directory/
```

其他资源

- **mt(1)** 和 **tar(1)** man pages

18.7. 从磁带设备中删除数据

要从磁带设备中删除数据，使用 **erase** 选项。

先决条件

1. 已安装 **mt-st** 软件包。如需更多信息，请参阅安装磁带驱动器管理工具。
2. 数据被写入磁带设备。如需更多信息，请参阅写入回卷磁带设备，或写入非回卷磁带设备。

流程

1. 从磁带设备中删除数据：

```
# mt -f /dev/st0 erase
```

2. 卸载磁带设备：

```
mt -f /dev/st0 offline
```

其他资源

- **mt(1)** man page

18.8. 磁带命令

以下是常见的 **mt** 命令：

表 18.1. mt 命令

命令	描述
mt -f /dev/*st0* status	显示磁带设备的状态。
mt -f /dev/*st0* erase	擦除整个磁带。
mt -f /dev/*nst0* rewind	插入磁带设备。
mt -f /dev/*nst0* fsf *n*	将磁带头切换到转发记录。在这里，*n* 是一个可选的文件数。如果指定了文件计数，磁带头将跳过 *n* 个记录。
mt -f /dev/*nst0* bsfm *n*	将磁头切换到之前的记录。
mt -f /dev/*nst0* eod	将磁带头切换到数据的末尾。

第 19 章 管理 RAID

您可以使用独立磁盘的冗余阵列 (RAID) 在多个驱动器间存储数据。它可以帮助避免驱动器失败时数据丢失。

19.1. RAID 概述

在 RAID 中，多个设备（如 HDD、SSD 或 NVMe）组合成一个阵列来实现性能或冗余目标，而不是使用一个大而昂贵的驱动器。这组设备以单一逻辑存储单元或驱动器在计算机中出现。

RAID 支持各种配置，包括 0、1、4、5、6、10 和 linear。RAID 使用磁盘条带 (RAID 0)、磁盘镜像 (RAID 1)和带有奇偶校验 (RAID 4 4、5 和 6) 的磁盘条带等技术来实现冗余、较低延迟、增大带宽以及最大程度从硬盘崩溃中恢复的能力。

RAID 通过将数据分解为一致的块（通常为 256 KB 或 512 KB）来跨阵列中的每个设备分发数据，但其他值可以接受。根据所使用的 RAID 级别，它会将这些块写入 RAID 阵列中的硬盘中。在读取数据时，进程是相反的顺序，实现阵列中的多个设备实际上是一个大驱动器的效果。

对于需要管理大量数据的人员，RAID 技术非常有用。以下是部署 RAID 的主要原因：

- 它可以提高速度
- 它使用单个虚拟磁盘增加存储容量
- 尽可能减少磁盘出现问题时导致数据丢失的问题
- RAID 布局和级别在线转换

19.2. RAID 类型

以下是可能的 RAID 类型：

固件 RAID

固件 RAID（也称为 ATARAID）是一种软件 RAID，可以使用基于固件的菜单配置 RAID 集。此类 RAID 使用的固件也会挂接到 BIOS 中，允许您从 RAID 集启动。不同的供应商使用不同的磁盘元数据格式来标记 RAID 集成员。Intel Matrix RAID 是固件 RAID 系统的示例。

硬件 RAID

基于硬件的阵列独立于主机管理 RAID 子系统。它可能会在每个 RAID 阵列中存在多个设备。
硬件 RAID 设备可能是系统内部或外部的。内部设备通常由一个专用的控制器卡组成，它处理对操作系统透明的 RAID 任务。外部设备通常通过 SCSI、光纤、iSCSI、InfiniBand 或者其它高速网络互联连接的系统,并显示卷如逻辑单元到系统。

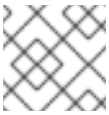
RAID 控制器卡的功能类似于操作系统的 SCSI 控制器，并处理所有实际驱动器通信。您可以将驱动器插入与普通 SCSI 控制器类似的 RAID 控制器，然后将其添加到 RAID 控制器配置中。操作系统将无法辨别它们的不同。

软件 RAID

软件 RAID 在内核块设备代码中实现各种 RAID 级别。它提供最便宜的解决方案，因为不需要昂贵的磁盘控制器卡或热插拔机箱。使用热插拔机箱时，您可以在不关闭系统的情况下删除硬盘驱动器。软件 RAID 也可用于任何由 Linux 内核支持的块存储，如 SATA、SCSI 和 NVMe。随着新 CPU 的速度加快，除非您使用高端存储设备，软件 RAID 通常会优于硬件 RAID。
由于 Linux 内核包含多个设备 (MD) 驱动程序，因此 RAID 解决方案将完全独立于硬件。基于软件的阵列的性能取决于服务器 CPU 的性能和负载。

以下是 Linux 软件 RAID 堆栈的主要功能：

- 多线程设计
- 在不同的 Linux 机器间移动磁盘阵列不需要重新构建数据
- 使用空闲系统资源进行后台阵列重构
- 支持热插拔驱动器
- 自动 CPU 检测以利用某些 CPU 功能，如流传输单一指令多个数据 (SIMD) 支持。
- 自动更正阵列磁盘中的错误扇区。
- 定期检查 RAID 数据，以确保阵列的健康状态。
- 主动监控阵列，在发生重要事件时将电子邮件报警发送到指定的电子邮件地址。
- Write-intent 位图通过允许内核准确了解磁盘的哪些部分需要重新同步，而不必在系统崩溃后重新同步整个阵列，可以大大提高了重新同步事件的速度。



注意

resync 是一个通过现有 RAID 设备同步数据以达到冗余的过程

- 重新同步检查点，以便如果您在重新同步期间重新启动计算机，则在启动时重新同步会从其停止的地方开始，而不是从头开始
- 安装后更改阵列参数的功能，称为重塑（reshaping）。例如：当有新设备需要添加时，您可以将 4 磁盘 RAID5 阵列增加成 5 磁盘 RAID5 阵列。这种增加操作是实时的，不需要您在新阵列上重新安装。
- 重塑支持更改设备数量、RAID 算法或 RAID 阵列类型的大小，如 RAID4、RAID5、RAID6 或 RAID10。
- 接管支持 RAID 级别转换，比如 RAID0 到 RAID6。
- 集群 MD 是集群的存储解决方案，可为集群提供 RAID1 镜像的冗余。目前，只支持 RAID1。

19.3. 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。

19.4. LINUX RAID 子系统

以下子系统组成了 Linux 中的 RAID 系统：

Linux 硬件 RAID 控制器驱动程序

硬件 RAID 控制器在 Linux 中没有特定的 RAID 子系统。由于它们使用特殊的 RAID 芯片组，因此硬件 RAID 控制器有自己的驱动程序。使用这些驱动程序时，系统会检测 RAID 集作为常规磁盘。

mdraid

mdraid 子系统设计为 Linux 的软件 RAID 解决方案。这也是 Red Hat Enterprise Linux 中软件 RAID 的首选解决方案。此子系统使用自己的元数据格式，称为原生 MD 元数据。
它还支持其他元数据格式，称为外部元数据。Red Hat Enterprise Linux 9 使用带有外部元数据的 **mdraid** 来访问 Intel Rapid Storage (ISW) 或 Intel Matrix Storage Manager (IMSM) 设置和存储网络行业关联 (SNIA) 磁盘驱动器格式 (DDF)。**mdraid** 子系统集通过 **mdadm** 程序进行配置和控制。

19.5. 在安装过程中创建软件 RAID

独立磁盘冗余阵列 (RAID) 设备由被安排的多个存储设备组成，以便在一些配置中提供更高的性能和容错能力。

创建 RAID 设备只需要一步，并可根据需要添加或者删除磁盘。您可以为系统中的每个物理磁盘配置一个 RAID 分区，因此安装程序可使用的磁盘数决定可用 RAID 设备的级别。例如：如果您的系统有两块磁盘，则无法创建 RAID 10 设备，因为它至少需要三块单独的磁盘。

注意

在 64 位 IBM Z 上，存储子系统透明地使用 RAID。您不必手动配置软件 RAID。

先决条件

- 您已经选择了两个或者多个磁盘，然后才能看到 RAID 配置选项。根据您要创建的 RAID 类型，至少需要两个磁盘。
- 您创建了挂载点。通过配置挂载点，您可以配置 RAID 设备。
- 您已在 **安装目的** 窗口中选择了 **自定义** 单选按钮。

流程

1. 在 **Manual Partitioning** 窗口左面地框中，选所需的分区。
2. 在 **Device(s)** 部分点 **修改**。此时会打开 **Configure Mount Point** 对话框。
3. 选择您要包含在 RAID 设备中的磁盘并点击 **选择**。
4. 点击**设备类型**下拉菜单并选择 **RAID**。

5. 点击**文件系统**下拉菜单并选择您首选的文件系统类型。
6. 点击**RAID 级别**下拉菜单并选择您需要的 RAID 级别。
7. 点击 **更新设置** 保存您的更改。
8. 点 **Done** 应用设置并返回到**按照概述**窗口。

其它资源

- 创建带有 DM 完整性的 RAID LV

19.6. 在安装的系统中创建软件 RAID

您可以使用 **mdadm** 程序在现有系统上创建一个软件独立磁盘阵列 (RAID)。

先决条件

- 已安装 **mdadm** 软件包。
- 您已在系统上创建了两个或多个分区。有关详细说明，请参阅 使用 parted 创建分区。

流程

1. 创建两个块设备的 RAID，如 */dev/sda1* 和 */dev/sdc1* ：

```
# mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sda1 /dev/sdc1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

level_value 选项定义 RAID 级别。

2. 可选：检查 RAID 的状态：

```
# mdadm --detail /dev/md0
/dev/md0:
           Version : 1.2
     Creation Time : Thu Oct 13 15:17:39 2022
        Raid Level : raid0
        Array Size : 18649600 (17.79 GiB 19.10 GB)
      Raid Devices : 2
     Total Devices : 2
       Persistence : Superblock is persistent

       Update Time : Thu Oct 13 15:17:39 2022
             State : clean
    Active Devices : 2
   Working Devices : 2
    Failed Devices : 0
     Spare Devices : 0
[...]
```

3. 可选：观察 RAID 中每个设备的详细信息：

```
# mdadm --examine /dev/sda1 /dev/sdc1
/dev/sda1:
          Magic : a92b4efc
        Version : 1.2
    Feature Map : 0x1000
     Array UUID : 77ddfb0a:41529b0e:f2c5cde1:1d72ce2c
           Name : 0
  Creation Time : Thu Oct 13 15:17:39 2022
     Raid Level : raid0
   Raid Devices : 2
[...]
```

4. 在 RAID 驱动器中创建文件系统：

```
# mkfs -t xfs /dev/md0
```

使用您选择格式化驱动器的文件系统替换 *xfs*。

5. 为 RAID 驱动器创建挂载点并挂载它：

```
# mkdir /mnt/raid1
# mount /dev/md0 /mnt/raid1
```

使用挂载点替换 */mnt/raid1*。

如果您希望 RHEL 在系统引导时自动挂载 **md0** RAID 设备，请将设备的条目添加到 **/etc/fstab 文件中** ：

```
/dev/md0   /mnt/raid1 xfs  defaults   0 0
```

19.7. 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

19.8. 扩展 RAID

您可以使用 **mdadm** 工具的 **--grow** 选项扩展 RAID。

先决条件

- 有足够的磁盘空间。
- **parted** 软件包已经安装。

流程

1. 扩展 RAID 分区。如需更多信息，请参阅使用 parted 重新定义分区。
2. 将 RAID 扩展到最大分区容量：

```
# mdadm --grow --size=max /dev/md0
```

要设置特定大小，设置 **--size** 参数的值（单位为 kB），如 **--size=*524228***。

3. 增加文件系统的大小。例如，如果卷使用 XFS 并挂载到 */mnt/*，请输入：

```
# xfs_growfs /mnt/
```

其他资源

- **mdadm (8)** 手册页
- 管理文件系统

19.9. 缩小 RAID

您可以使用 **mdadm** 工具的 **--grow** 选项缩小 RAID。

重要

XFS 文件系统不支持缩小。

先决条件

- **parted** 软件包已经安装。

流程

1. 缩小文件系统。如需更多信息，请参阅管理文件系统。
2. 减少 RAID 的大小，例如 *512 MB* ：

```
# mdadm --grow --size=524228 /dev/md0
```

写 **--size** 参数（单位为 kB）。

3. 将分区缩小到您需要的大小。

其他资源

- **mdadm (8)** 手册页
- 使用 parted 重新定义分区大小。

19.10. 支持的 RAID 转换

可以从一个 RAID 级别转换到另一个 RAID 级别。例如：您可以从 RAID5 转换到 RAID10，但不能从 RAID10 转换为 RAID5。下表描述了支持的 RAID 转换：

源级别	目标级别
RAID0	RAID4, RAID5, RAID10
RAID1	RAID0, RAID5
RAID4	RAID0, RAID5
RAID5	RAID0, RAID1, RAID4, RAID6, RAID10
RAID6	RAID5
RAID10	RAID0

注意

只有在使用 **ALGORITHM_PARITY_N** 布局时，才可以将 RAID 5 转换为 RAID0 和 RAID4。

其他资源.

- **mdadm (8)** 手册页

19.11. 转换 RAID 级别

您可以根据需要将 RAID 转换为不同的 RAID 级别。以下示例将 RAID 设备 **/dev/md0** 的级别从 0 转换为 5，并为阵列添加一个磁盘 */dev/sdd*。

先决条件

- 有足够的磁盘进行转换。
- 已安装 **mdadm** 软件包。
- 确定支持预期的转换。请参阅支持的 RAID 转换。

流程

1. 将 RAID /dev/md0 转换为 RAID 级别 *5*：

```
# mdadm --grow --level=5 -n 3 /dev/md0 --force
```

2. 在阵列中添加新磁盘：

```
# mdadm --manage /dev/md0 --add /dev/sdd
```

验证

- 验证 RAID 级别是否已转换：

```
# mdadm --detail /dev/md0
/dev/md0:
           Version : 1.2
     Creation Time : Thu Oct 13 15:17:39 2022
        Raid Level : raid0
        Array Size : 18649600 (17.79 GiB 19.10 GB)
      Raid Devices : 5
[...]
```

其他资源

- **mdadm (8)** 手册页

19.12. 安装后将根磁盘转换为 RAID1

这部分论述了如何在安装 Red Hat Enterprise Linux 9 后将非 RAID 根磁盘转换为 RAID1 镜像。

在 PowerPC (PPC) 构架中，执行以下步骤：

先决条件

- 以下红帽知识库中的说明已完成： 安装 Red Hat Enterprise Linux 7 后如何将根磁盘转换为 RAID1？

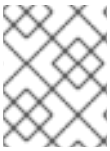
流程

1. 将 PowerPC Reference Platform (PReP) 引导分区从 */dev/sda1* 复制到 */dev/sdb1* ：

```
# dd if=/dev/sda1 of=/dev/sdb1
```

2. 在两个磁盘的第一个分区中更新 **prep** 和 **boot** 标志：

```
$ parted /dev/sda set 1 prep on
$ parted /dev/sda set 1 boot on

$ parted /dev/sdb set 1 prep on
$ parted /dev/sdb set 1 boot on
```



注意

执行 **grub2-install *dev/sda*** 命令无法在 PowerPC 机器上工作并返回错误，但系统会按预期引导。

19.13. 创建高级 RAID 设备

在某些情况下，您可能想要在安装完成前创建的阵列上安装操作系统。通常，这意味着在复杂的 RAID 设备中设置 **/boot** 或 root 文件系统阵列。在这种情况下，您可能需要使用 Anaconda 安装程序不支持的数组选项。要临时解决这个问题，请执行以下步骤。

注意

安装程序的有限 Rescue Mode 不包括 man page。**mdadm** 和 **md** man page 都包含创建自定义 RAID 阵列的有用信息，在整个临时解决方案中可能需要。

流程

1. 插入安装磁盘。
2. 在初始启动过程中，选择 **Rescue Mode** 而不是 **Install** 或 **Upgrade**。当系统完全引导至 **Rescue mode** 时，您可以看到命令行终端。
3. 在这个终端中执行以下命令：
   a. 使用 **parted** 命令在目标硬盘上创建 RAID 分区。
   b. 使用这些分区中的 **mdadm** 命令，使用任何以及所有可用的设置和选项来手动创建 raid 阵列。
4. 可选：创建阵列后，在阵列上创建文件系统。
5. 重新启动计算机，再选择要安装的 **Install** 或 **Upgrade**。当 Anaconda 安装程序搜索系统中的磁盘时，它会找到预先存在的 RAID 设备。
6. 当问到如何使用系统中的磁盘时，请选择 **Custom Layout**，并点 **Next**。在设备列表中，会列出预先存在的 MD RAID 设备。
7. 选择一个 RAID 设备并点 **Edit**。
8. 如果之前没有创建挂载点，则配置其挂载点以及应使用的文件系统类型，然后点 **Done**。Anaconda 安装至此已存在的 RAID 设备，保留在 Rescue Mode 中创建时选择的自定义选项。

19.14. 设置用于监控 RAID 的电子邮件通知

您可以使用 **mdadm** 工具设置电子邮件警报来监控 RAID。当 **MAILADDR** 变量设置为所需的电子邮件地址后，监控系统会将警报发送到添加的电子邮件地址。

先决条件

- 已安装 **mdadm** 软件包。
- 设定邮件服务。

流程

1. 通过扫描 RAID 详情来创建 **/etc/mdadm.conf** 配置文件来监控阵列：

```
# mdadm --detail --scan >> /etc/mdadm.conf
```

   请注意，**ARRAY** 和 **MAILADDR** 是强制的变量。

2. 使用您选择的文本编辑器打开 **/etc/mdadm.conf** 配置文件，并使用通知的电子邮件地址添加 **MAILADDR** 变量。例如，添加新行：

```
MAILADDR example@example.com
```

在这里，*example@example.com* 是从阵列监控接收警报的电子邮件地址。

3. 保存 **/etc/mdadm.conf** 文件中的更改并关闭它。

其他资源

- **mdadm.conf(5)** man page

19.15. 替换 RAID 中失败的磁盘

您可以使用剩余的磁盘从失败磁盘重建数据。RAID 级别和磁盘总数决定了成功重建数据所需的最小剩余磁盘量。

在此过程中，*/dev/md0* RAID 包含四个磁盘。*/dev/sdd* 磁盘失败，您需要将它替换为 */dev/sdf* 磁盘。

先决条件

- 用于替换的备用磁盘。
- 已安装 **mdadm** 软件包。

流程

1. 检查失败的磁盘：

   a. 查看内核日志：

   ```
   # journalctl -k -f
   ```

   b. 搜索类似如下的消息：

   ```
   md/raid:md0: Disk failure on sdd, disabling device.

   md/raid:md0: Operation continuing on 3 devices.
   ```

   c. 按 **Ctrl**+**C** 键退出 **journalctl** 程序。

2. 将失败的磁盘标记为故障：

   ```
   # mdadm --manage /dev/md0 --fail /dev/sdd
   ```

3. 可选：检查失败的磁盘是否已正确标记：

   ```
   # mdadm --detail /dev/md0
   ```

   输出末尾是 */dev/md0* RAID 中的磁盘列表，其中磁盘 */dev/sdd* 具有 **faulty** 状态 ：

   ```
   Number   Major   Minor   RaidDevice State
      0       8       16        0      active sync   /dev/sdb
      1       8       32        1      active sync   /dev/sdc
      -       0        0        2      removed
      3       8       64        3      active sync   /dev/sde

      2       8       48        -      faulty   /dev/sdd
   ```

4. 从 RAID 中删除失败的磁盘：

```
# mdadm --manage /dev/md0 --remove /dev/sdd
```

> **警告**
>
> 如果您的 RAID 无法与另一个磁盘失败，在新磁盘有 **active sync** 状态前不会删除任何磁盘。您可以使用 **watch cat /proc/mdstat** 命令监控进度。

5. 在 RAID 中添加新磁盘：

```
# mdadm --manage /dev/md0 --add /dev/sdf
```

/dev/md0 RAID 现在包括新磁盘 */dev/sdf*，**mdadm** 服务将自动从其他磁盘将数据复制到其中。

验证

- 检查阵列的详情：

```
# mdadm --detail /dev/md0
```

如果这个命令显示 */dev/md0* RAID 中的磁盘列表，其中新磁盘在输出末尾具有 **spare rebuilding** 状态，则数据仍会从其他磁盘复制到其中：

```
Number   Major   Minor   RaidDevice State
   0       8       16        0      active sync   /dev/sdb
   1       8       32        1      active sync   /dev/sdc
   4       8       80        2      spare rebuilding   /dev/sdf
   3       8       64        3      active sync   /dev/sde
```

数据复制完成后，新磁盘会处于 **active sync** 状态。

其他资源

- 设置用于监控 RAID 的电子邮件通知

19.16. 修复 RAID 磁盘

这个步骤描述了如何修复 RAID 阵列中的磁盘。

先决条件

- 已安装 **mdadm** 软件包。

流程

1. 检查阵列失败的磁盘行为：

```
# echo check > /sys/block/md0/md/sync_action
```

这会检查数组和 **/sys/block/md0/md/sync_action** 文件显示 sync 操作。

2. 使用您选择的文本编辑器打开 **/sys/block/md0/md/sync_action** 文件，并查看是否有任何有关磁盘同步失败的消息。

3. 查看 **/sys/block/md0/md/mismatch_cnt** 文件。如果 **mismatch_cnt** 参数不是 **0**，这意味着 RAID 磁盘需要修复。

4. 修复阵列中的磁盘：

```
# echo repair > /sys/block/md0/md/sync_action
```

这会修复阵列中的磁盘，并将结果写入 **/sys/block/md0/md/sync_action** 文件。

5. 查看同步进度：

```
# cat /sys/block/md0/md/sync_action
repair

# cat /proc/mdstat
Personalities : [raid0] [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdg[1] dm-3[0]
      511040 blocks super 1.2 [2/2] [UU]
unused devices: <none>
```

第 20 章 使用 LUKS 加密块设备

通过使用磁盘加密，您可以通过对其进行加密来保护块设备上的数据。要访问设备的解密内容，请输入密码短语或密钥作为身份验证。这对移动计算机和可移动介质非常重要，因为它有助于保护设备的内容，即使它已从系统上物理移除。LUKS 格式是 Red Hat Enterprise Linux 中块设备加密的默认实现。

20.1. LUKS 磁盘加密

Linux Unified Key Setup-on-disk-format (LUKS)提供了一组简化管理加密设备的工具。使用 LUKS，您可以加密块设备，并启用多个用户密钥来解密主密钥。要批量加密分区，请使用这个主密钥。

Red Hat Enterprise Linux 使用 LUKS 执行块设备加密。默认情况下，在安装过程中不选中加密块设备的选项。如果您选择加密磁盘的选项，则系统会在每次引导计算机时提示您输入密码短语。这个密码短语解锁解密分区的批量加密密钥。如果要修改默认分区表，您可以选择您要加密的分区。这是在分区表设置中设定的。

加密系统

LUKS 使用的默认密码是 **aes-xts-plain64**。LUKS 的默认密钥大小为 512 字节。 **Anaconda** XTS 模式的 LUKS 的默认密钥大小为 512 位。以下是可用密码：

- 高级加密标准(AES)
- Twofish
- Serpent

LUKS 执行的操作

- LUKS 对整个块设备进行加密，因此非常适合保护移动设备的内容，如可移动存储介质或笔记本电脑磁盘驱动器。
- 加密块设备的底层内容是任意的，这有助于加密交换设备。对于将特殊格式化块设备用于数据存储的某些数据库，这也很有用。
- LUKS 使用现有的设备映射器内核子系统。
- LUKS 增强了密码短语，防止字典攻击。
- LUKS 设备包含多个密钥插槽，这意味着您可以添加备份密钥或密码短语。

重要

以下情况不建议使用 LUKS：

- LUKS 等磁盘加密解决方案仅在您的系统关闭时保护数据。在系统启动并且 LUKS 解密磁盘后，该磁盘上的文件可供有权访问它们的任何人使用。
- 需要多个用户对同一设备具有不同访问密钥的情况。LUKS1 格式提供八个密钥插槽，LUKS2 提供最多 32 个密钥插槽。
- 需要文件级加密的应用程序。

其他资源

- LUKS 项目主页

- LUKS On-Disk Format 规格
- FIPS 197: 高级加密标准(AES)

20.2. RHEL 中的 LUKS 版本

在 Red Hat Enterprise Linux 中，LUKS 加密的默认格式为 LUKS2。旧的 LUKS1 格式仍被完全支持，并作为与早期 Red Hat Enterprise Linux 版本兼容的格式提供。与 LUKS1 重新加密相比，LUKS2 重新加密被视为更健壮且更安全。

LUKS2 格式允许将来对各个部分的更新，而无需修改二进制结构。它在内部对元数据使用 JSON 文本格式，提供元数据的冗余，检测元数据损坏，并从元数据副本自动修复。

重要

不要在只支持 LUKS1 的系统中使用 LUKS2。

从 Red Hat Enterprise Linux 9.2 开始，您可以对两个 LUKS 版本使用 **cryptsetup reencrypt** 命令来加密磁盘。

在线重新加密

LUKS2 格式支持在设备正在使用时重新加密加密设备。例如：您不必卸载该设备中的文件系统来执行以下任务：

- 更改卷密钥
- 更改加密算法
  加密未加密的设备时，您仍然必须卸载文件系统。您可以在简短初始化加密后重新挂载文件系统。

  LUKS1 格式不支持在线重新加密。

转换

在某些情况下，您可以将 LUKS1 转换为 LUKS2。在以下情况下无法进行转换：

- LUKS1 设备被标记为在由基于策略的解密(PBD) Clevis 解决方案使用。当检测到某些 **luksmeta** 元数据时，**cryptsetup** 工具不会转换设备。
- 设备正在活跃。在可能任何转换前，设备必须处于不活跃状态。

20.3. LUKS2 重新加密过程中数据保护选项

LUKS2 提供了几个在重新加密过程中优先考虑性能或数据保护的选项。它为 **resilience** 选项提供以下模式，您可以使用 **cryptsetup reencrypt --resilience resilience-mode *dev/sdx*** 命令选择这些模式的任一个：

checksum

默认模式。它在数据保护和性能之间保持平衡。
这个模式将扇区的各个校验和存储在重新加密区域中，恢复过程可用来检测 LUKS2 重新加密的扇区。模式要求块设备扇区写入具有"原子"性。

journal

最安全的模式，但也是最慢的模式。由于此模式将重新加密区域记录在二进制区域中，因此 LUKS2 将数据写入两次。

none

none 模式优先考虑性能，不提供数据保护。它只保护数据免受安全进程终止的影响，如 **SIGTERM** 信号或用户按了 **Ctrl+C** 键。任何意外的系统故障或应用程序失败都可能会导致数据损坏。

如果 LUKS2 重新加密进程意外被强行终止，LUKU2 可通过以下方法执行恢复：

自动

在下一个 LUKS2 设备打开操作过程中，执行以下操作之一会触发自动恢复操作：

- 执行 **cryptsetup open** 命令。
- 使用 **systemd-cryptsetup** 命令附加设备。

手动

通过在 LUKS2 设备上使用 **cryptsetup repair** ***/dev/sdx*** 命令。

其他资源

- **cryptsetup-reencrypt (8)** 和 **cryptsetup-repair (8)** 手册页

20.4. 使用 LUKS2 加密块设备上的现有数据

您可以使用 LUKS2 格式加密尚未加密设备上的现有数据。新的 LUKS 标头保存在设备的标头中。

先决条件

- 块设备有一个文件系统。
- 已备份了数据。

警告

由于硬件、内核或人为故障，您可能会在加密过程中丢失数据。在开始加密数据之前，请确保您有可靠的备份。

流程

1. 卸载您要加密的设备上的所有文件系统，例如：

```
# umount /dev/mapper/vg00-lv00
```

2. 为存储 LUKS 标头腾出空间。使用以下适合您场景的选项之一：
   - 如果是加密逻辑卷，您可以扩展逻辑卷而无需调整文件系统的大小。例如：

```
# lvextend -L+32M /dev/mapper/vg00-lv00
```

- 使用分区管理工具（如 **parted** ）扩展分区。
- 缩小该设备的文件系统。您可以对 ext2、ext3 或 ext4 文件系统使用 **resize2fs** 工具。请注意，您无法缩小 XFS 文件系统。

3. 初始化加密：

```
# cryptsetup reencrypt --encrypt --init-only --reduce-device-size 32M /dev/mapper/vg00-lv00
lv00_encrypted

/dev/mapper/lv00_encrypted is now active and ready for online encryption.
```

4. 挂载该设备：

```
# mount /dev/mapper/lv00_encrypted /mnt/lv00_encrypted
```

5. 将持久映射的条目添加到 **/etc/crypttab** 文件中：

   a. 查找 **luksUUID** ：

   ```
   # cryptsetup luksUUID /dev/mapper/vg00-lv00

   a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
   ```

   b. 在您选择的文本编辑器中打开 **/etc/crypttab**，并在此文件中添加一个设备：

   ```
   $ vi /etc/crypttab

   lv00_encrypted UUID=a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325 none
   ```

   将 *a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325* 替换为您设备的 **luksUUID**。

   c. 使用 **dracut** 刷新 initramfs：

   ```
   $ dracut -f --regenerate-all
   ```

6. 向 **/etc/fstab** 文件中添加一个永久挂载条目：

   a. 查找活跃 LUKS 块设备的文件系统的 UUID：

   ```
   $ blkid -p /dev/mapper/lv00_encrypted

   /dev/mapper/lv00-encrypted: UUID="37bc2492-d8fa-4969-9d9b-bb64d3685aa9"
   BLOCK_SIZE="4096" TYPE="xfs" USAGE="filesystem"
   ```

   b. 在您选择的文本编辑器中打开 **/etc/fstab**，并在此文件中添加一个设备，例如：

   ```
   $ vi /etc/fstab

   UUID=37bc2492-d8fa-4969-9d9b-bb64d3685aa9 /home auto rw,user,auto 0
   ```

   将 *37bc2492-d8fa-4969-9d9b-bb64d3685aa9* 替换为您文件系统的 UUID。

7. 恢复在线加密：

```
# cryptsetup reencrypt --resume-only /dev/mapper/vg00-lv00

Enter passphrase for /dev/mapper/vg00-lv00:
Auto-detected active dm device 'lv00_encrypted' for data device /dev/mapper/vg00-lv00.
Finished, time 00:31.130, 10272 MiB written, speed 330.0 MiB/s
```

验证

1. 验证现有数据是否已加密：

```
# cryptsetup luksDump /dev/mapper/vg00-lv00

LUKS header information
Version: 2
Epoch: 4
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID: a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
Label: (no label)
Subsystem: (no subsystem)
Flags: (no flags)

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

2. 查看加密的空白块设备的状态：

```
# cryptsetup status lv00_encrypted

/dev/mapper/lv00_encrypted is active and is in use.
  type:    LUKS2
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device:  /dev/mapper/vg00-lv00
```

其他资源

- **cryptsetup(8)**, **cryptsetup-reencrypt(8)**, **lvextend(8)**, **resize2fs(8)**, 和 **parted(8)** 手册页

20.5. 使用带有分离标头的 LUKS2 在块设备上加密现有数据

您可以加密块设备上的现有数据，而无需为存储 LUKS 标头创建可用空间。标头存储在分离的位置，它也充当额外的安全层。该流程使用 LUKS2 加密格式。

先决条件

- 块设备有一个文件系统。

- 已备份了数据。

警告

由于硬件、内核或人为故障，您可能会在加密过程中丢失数据。在开始加密数据之前，请确保您有可靠的备份。

流程

1. 卸载设备上的所有文件系统，例如：

```
# umount /dev/nvme0n1p1
```

2. 初始化加密：

```
# cryptsetup reencrypt --encrypt --init-only --header /home/header /dev/nvme0n1p1
nvme_encrypted

WARNING!
========
Header file does not exist, do you want to create it?

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /home/header:
Verify passphrase:
/dev/mapper/nvme_encrypted is now active and ready for online encryption.
```

将 */home/header* 替换为带有分离的 LUKS 标头的文件的路径。分离的 LUKS 标头必须可以访问，以便稍后解锁加密的设备。

3. 挂载该设备：

```
# mount /dev/mapper/nvme_encrypted /mnt/nvme_encrypted
```

4. 恢复在线加密：

```
# cryptsetup reencrypt --resume-only --header /home/header /dev/nvme0n1p1

Enter passphrase for /dev/nvme0n1p1:
Auto-detected active dm device 'nvme_encrypted' for data device /dev/nvme0n1p1.
Finished, time 00m51s,   10 GiB written, speed 198.2 MiB/s
```

验证

1. 验证在使用带有分离标头的 LUKS2 块设备上的现有数据是否已加密：

```
# cryptsetup luksDump /home/header

LUKS header information
```

```
Version:        2
Epoch:          88
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           c4f5d274-f4c0-41e3-ac36-22a917ab0386
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
 offset: 0 [bytes]
 length: (whole device)
 cipher: aes-xts-plain64
 sector: 512 [bytes]
[...]
```

2. 查看加密的空白块设备的状态：

```
# cryptsetup status nvme_encrypted

/dev/mapper/nvme_encrypted is active and is in use.
  type:    LUKS2
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device:  /dev/nvme0n1p1
```

其他资源

- **cryptsetup(8)** 和 **cryptsetup-reencrypt(8)** 手册页

20.6. 使用 LUKS2 加密空白块设备

您可以加密空白块设备，您可以使用 LUKS2 格式将其用于加密存储。

先决条件

- 空白块设备。您可以使用 **lsblk** 等命令来查找该设备上是否没有实际的数据，例如，文件系统。

流程

1. 将分区设置为加密的 LUKS 分区：

```
# cryptsetup luksFormat /dev/nvme0n1p1

WARNING!
========
This will overwrite data on /dev/nvme0n1p1 irrevocably.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/nvme0n1p1:
Verify passphrase:
```

2. 打开加密的 LUKS 分区：

```
# cryptsetup open /dev/nvme0n1p1 nvme0n1p1_encrypted

Enter passphrase for /dev/nvme0n1p1:
```

这会解锁分区，并使用设备映射器将其映射到新设备。要不覆盖加密的数据，这个命令会警告内核，该设备是一个加密设备，并使用 **/dev/mapper/*device_mapped_name*** 路径通过 LUKS 来解决。

3. 创建一个文件系统来将加密的数据写入分区，该分区必须可通过设备映射名称访问：

```
# mkfs -t ext4 /dev/mapper/nvme0n1p1_encrypted
```

4. 挂载该设备：

```
# mount /dev/mapper/nvme0n1p1_encrypted mount-point
```

验证

1. 验证空白块设备是否已加密：

```
# cryptsetup luksDump /dev/nvme0n1p1

LUKS header information
Version:        2
Epoch:          3
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           34ce4870-ffdf-467c-9a9e-345a53ed8a25
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
 offset: 16777216 [bytes]
 length: (whole device)
 cipher: aes-xts-plain64
 sector: 512 [bytes]
[...]
```

2. 查看加密的空白块设备的状态：

```
# cryptsetup status nvme0n1p1_encrypted

/dev/mapper/nvme0n1p1_encrypted is active and is in use.
  type:    LUKS2
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device:  /dev/nvme0n1p1
  sector size:  512
  offset:  32768 sectors
  size:    20938752 sectors
  mode:    read/write
```

其他资源

- **cryptsetup (8)**, **cryptsetup-open (8)**, 和 **cryptsetup-lusFormat (8)** 手册页

20.7. 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

第 21 章 使用 NVDIMM 持久性内存存储

您可以在连接到您的系统的非线性内存模块 (NVDIMM) 设备上启用和管理各种存储类型。

有关在 NVDIMM 存储上安装 Red Hat Enterprise Linux 9，请参阅安装到 NVDIMM 设备。

21.1. NVDIMM 持久内存技术

非易失性双内存模块 (NVDIMM) 持久内存（也称为存储类内存或 **pmem**）是内存和存储的组合。

NVDIMM 将存储的持久性与低访问延迟和动态 RAM (DRAM) 的高带宽合并。以下是使用 NVDIMM 的其他优点：

- NVDIMM 存储是字节地址的，这意味着可以使用 CPU 负载和存储指令来访问。除了访问传统的基于块的存储所需的 read() 和 write() 系统调用外，NVDIMM 还支持直接加载和存储编程模型。
- NVDIMM 的性能特征与具有非常低访问延迟的 DRAM 类似，通常以十到百纳秒为单位。
- 当电源关闭时，保存在 NVDIMM 中的数据会被保留，类似于持久性内存。
- 通过直接访问 (DAX) 技术，无需通过系统页面缓存，即可直接到内存映射存储。这样便可为其他目的释放 DRAM。

在以下用例中 NVDIMM 很有用，比如：

数据库

NVDIMM 减少的存储访问延迟可提高数据库性能。

快速重启

快速重启也被称为热缓存效果。例如：文件服务器在启动后没有在内存中的文件内容。客户端连接和读取或者写入数据时，会在页面缓存中缓存这些数据。最后，缓存包括大多数热数据。重启后，该系统必须在传统存储上再次启动该进程。
使用 NVDIMM 时，如果应用程序被正确设计，应用程序可能会在重启后保留 warm 缓存。在本例中，不会涉及页面缓存：应用程序会直接在持久内存中缓存数据。

快速写缓存

在数据被存入持久介质前，文件服务器通常不会确认客户端的写入请求。使用 NVDIMM 作为快速写入缓存可让文件服务器快速确认写入请求，因为会较低延迟。

21.2. NVDIMM 交集和地区

非易失性双内存模块 (NVDIMM) 设备支持分组到交集区域。

NVDIMM 设备可以像常规动态 RAM (DRAM) 一样分组为交集。interleave 集与跨多个 DIMM 的 RAID 0 级别（条带）配置类似。Interleave 集也称为区域（region）。

它有以下优点：

- 当 NVDIMM 设备被配置为 interleave 集时，会提高性能。
- 交集可以将多个较小的 NVDIMM 设备组合成一个更大的逻辑设备。

NVDIMM 交集是在系统 BIOS 或 UEFI 固件中配置的。Red Hat Enterprise Linux 为每个交集创建一个区域设备。

21.3. NVDIMM 命名空间

根据标签区域的大小，非易失性双内存模块 (NVDIMM) 区域可以划分为一个或多个命名空间。使用命名空间，您可以根据命名空间的访问模式访问设备，如 **sector**、**fsdax**、**devdax** 和 **raw**。更多信息，NVDIMM 访问模式。

有些 NVDIMM 设备不支持区域中的多个命名空间：

- 如果您的 NVDIMM 设备支持标签，您可以将区域重新划分到命名空间。
- 如果您的 NVDIMM 设备不支持标签，则区域只能包含单个命名空间。在这种情况下，Red Hat Enterprise Linux 会创建一个覆盖整个区域的默认命名空间。

21.4. NVDIMM 访问模式

您可以配置 Non-Volatile Dual In-line Memory Modules (NVDIMM) 命名空间，以使用以下模式之一：

sector

将存储显示为一个快速块设备。这个模式可用于没有修改以使用 NVDIMM 存储的传统应用程序，或者用于使用完整 I/O 堆栈（包括设备映射器）的应用程序。
sector 设备可以与系统中的其它块设备相同。您可以在上面创建分区或文件系统，将其配置为软件 RAID 集的一部分，或者将其用作 **dm-cache** 的缓存设备。

此模式中的设备可作为 **/dev/pmem*N*s** 提供。创建命名空间后，查看列出的 **blockdev** 值。

devdax 或设备直接访问 (DAX)

使用 **devdax** 时，NVDIMM 设备支持直接访问编程，如存储网络行业关联 (SNIA) 非易失性内存 (NVM) 编程模型规格中所述。在这个模式中，I/O 绕过内核的存储堆栈。因此无法使用设备映射器驱动程序。
设备 DAX 通过使用 DAX 字符设备节点提供对 NVDIMM 存储的原始访问。可以使用 CPU 缓存清除和隔离指令，使 **devdax** 设备中的数据可用。某些数据库和虚拟机虚拟机监控程序可能会受益于此模式。无法在 **devdax** 设备上创建文件系统。

这个模式中的设备可作为 **/dev/dax*N.M*** 提供。创建命名空间后，请参阅列出的 **chardev** 值。

fsdax，或者文件系统直接访问 (DAX)

使用 **fsdax** 时，NVDIMM 设备支持直接访问编程，如存储网络行业关联 (SNIA) 非易失性内存(NVM) 编程模型规格中所述。在这个模式中，I/O 会绕过内核的存储堆栈，因此无法使用很多设备映射器驱动程序。
您可以在文件系统 DAX 设备中创建文件系统。

此模式中的设备可作为 **/dev/pmem*N*** 提供。创建命名空间后，查看列出的 **blockdev** 值。

重要

文件系统 DAX 技术仅作为技术预览提供，不受红帽支持。

raw

显示不支持 DAX 的内存磁盘。在这个模式中，命名空间有一些限制，不应使用。
此模式中的设备可作为 **/dev/pmem*N*** 提供。创建命名空间后，查看列出的 **blockdev** 值。

21.5. 安装 NDCTL

您可以安装 **ndctl** 工具来配置和监控 Non-Volatile Dual In-line Memory Modules (NVDIMM) 设备。

流程

- 安装 **ndctl** 工具：

```
# dnf install ndctl
```

21.6. 在 NVDIMM 上创建扇区命名空间以充当块设备

您可以在扇区模式（也称为传统模式）中配置非线性内存模块 (NVDIMM) 设备，以支持传统的基于块的存储。

您可以：

- 将现有命名空间重新配置为扇区模式，或者
- 如果有可用空间，创建一个新的 sector 命名空间。

先决条件

- 一个 NVDIMM 设备被附加到您的系统。

21.6.1. 将现有的 NVDIMM 命名空间重新配置为扇区模式

您可以将非易失性双内存模块 (NVDIMM) 命名空间重新配置为扇区模式，将其用作快速块设备。

警告

重新配置命名空间会删除之前在命名空间中存储的数据。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。

流程

1. 查看现有命名空间：

```
# ndctl list --namespaces --idle
[
  {
    "dev":"namespace1.0",
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":1
```

```
  },
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":0
  }
]
```

2. 将所选命名空间重新配置为扇区模式：

```
# ndctl create-namespace --force --reconfig=namespace-ID --mode=sector
```

例 21.1. 在扇区模式中重新配置 namespace1.0

```
# ndctl create-namespace --force --reconfig=namespace1.0 --mode=sector
{
  "dev":"namespace1.0",
  "mode":"sector",
  "size":"755.26 GiB (810.95 GB)",
  "uuid":"2509949d-1dc4-4ee0-925a-4542b28aa616",
  "sector_size":4096,
  "blockdev":"pmem1s"
}
```

重新配置的命名空间现在位于 **/dev** 目录下，存为 **/dev/pmem1s** 文件。

验证

- 验证系统中的现有命名空间是否已重新配置：

```
# ndctl list --namespace namespace1.0
[
  {
    "dev":"namespace1.0",
    "mode":"sector",
    "size":810954706944,
    "uuid":"2509949d-1dc4-4ee0-925a-4542b28aa616",
    "sector_size":4096,
    "blockdev":"pmem1s"
  }
]
```

其他资源

- **ndctl-create-namespace (1)** 手册页

21.6.2. 在扇区模式下创建新 NVDIMM 命名空间

您可以在扇区模式中创建一个非线性内存模块 (NVDIMM) 命名空间，以便在区域中存在可用空间时将其用作快速块设备。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。
- NVDIMM 设备支持标签在区域中创建多个命名空间。您可以使用以下命令检查它：

```
# ndctl read-labels nmem0 >/dev/null
  read 1 nmem
```

这表示它读取一个 NVDIMM 设备的标签。如果值为 **0**，这表示您的设备不支持标签。

流程

1. 列出系统上有可用空间的 **pmem** 区域。在以下示例中，在 *region1* 和 *region0* 区域中有空间：

```
# ndctl list --regions
[
  {
    "dev":"region1",
    "size":2156073582592,
    "align":16777216,
    "available_size":2117418876928,
    "max_available_extent":2117418876928,
    "type":"pmem",
    "iset_id":-9102197055295954944,
    "badblock_count":1,
    "persistence_domain":"memory_controller"
  },
  {
    "dev":"region0",
    "size":2156073582592,
    "align":16777216,
    "available_size":2143188680704,
    "max_available_extent":2143188680704,
    "type":"pmem",
    "iset_id":736272362787276936,
    "badblock_count":3,
    "persistence_domain":"memory_controller"
  }
]
```

2. 在任何可用区域上分配一个或多个命名空间：

```
# ndctl create-namespace --mode=sector --region=regionN --size=namespace-size
```

例 21.2. 在 region0 上创建 36-GiB 扇区命名空间

```
# ndctl create-namespace --mode=sector --region=region0 --size=36G
{
  "dev":"namespace0.1",
  "mode":"sector",
  "size":"35.96 GiB (38.62 GB)",
  "uuid":"ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
```

```
    "sector_size":4096,
    "blockdev":"pmem0.1s"
  }
```

新命名空间现在作为 **/dev/pmem0.1s** 提供。

验证

- 验证新命名空间是否在扇区模式中创建：

```
# ndctl list -RN -n namespace0.1
{
  "regions":[
    {
      "dev":"region0",
      "size":2156073582592,
      "align":16777216,
      "available_size":2104533975040,
      "max_available_extent":2104533975040,
      "type":"pmem",
      "iset_id":736272362787276936,
      "badblock_count":3,
      "persistence_domain":"memory_controller",
      "namespaces":[
        {
          "dev":"namespace0.1",
          "mode":"sector",
          "size":38615912448,
          "uuid":"ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
          "sector_size":4096,
          "blockdev":"pmem0.1s"
        }
      ]
    }
  ]
}
```

其他资源

- **ndctl-create-namespace (1)** 手册页

21.7. 在 NVDIMM 上创建设备 DAX 命名空间

在设备 DAX 模式下配置附加到您系统的 NVDIMM 设备，以支持具有直接访问功能的字符存储。

考虑以下选项：

- 将现有命名空间重新配置为设备 DAX 模式。
- 如果有可用空间，请创建新设备 DAX 命名空间。

21.7.1. 设备直接访问模式中的 NVDIMM

设备直接访问（设备 DAX、**devdax**）提供了应用程序直接访问存储的方法，而无需参与文件系统。设备 DAX 的优点是它提供有保证的容错粒度，可以使用 **ndctl** 工具的 **--align** 选项来进行配置。

对于 Intel 64 和 AMD64 构架，支持以下故障颗粒度：

- 4 KiB
- 2 MiB
- 1 GiB

设备 DAX 节点只支持以下系统调用：

- **open()**
- **close()**
- **mmap()**

您可以使用 **ndctl list --human --capabilities** 命令查看 NVDIMM 设备支持的协调。例如，若要为 *region0* 设备查看它，请使用 **ndctl list --human --capabilities -r region0** 命令。

注意

不支持 **read()** 和 **write()** 系统调用，因为设备 DAX 用例与 SNIA Non-Volatile Memory Programming Model 关联。

21.7.2. 将现有的 NVDIMM 命名空间重新配置为设备 DAX 模式

您可以将现有的 Non-Volatile Dual In-line Memory Modules (NVDIMM) 命名空间重新配置为设备 DAX 模式。

警告

重新配置命名空间会删除之前在命名空间中存储的数据。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。

流程

1. 列出系统中的所有命名空间：

```
# ndctl list --namespaces --idle

[
  {
    "dev":"namespace1.0",
    "mode":"raw",
    "size":34359738368,
```

```
    "uuid":"ac951312-b312-4e76-9f15-6e00c8f2e6f4"
    "state":"disabled",
    "numa_node":1
  },
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":38615912448,
    "uuid":"ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
    "state":"disabled",
    "numa_node":0
  }
]
```

2. 重新配置任何命名空间：

```
# ndctl create-namespace --force --mode=devdax --reconfig=namespace-ID
```

例 21.3. 将命名空间配置为设备 DAX

以下命令为支持 DAX 的数据存储重新配置 **namespace0.1**。它与 2-MiB 故障粒度一致，以确保操作系统一次在 2-MiB 页面中故障：

```
# ndctl create-namespace --force --mode=devdax  --align=2M --reconfig=namespace0.1
{
  "dev":"namespace0.1",
  "mode":"devdax",
  "map":"dev",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"426d6a52-df92-43d2-8cc7-046241d6d761",
  "daxregion":{
    "id":0,
    "size":"35.44 GiB (38.05 GB)",
    "align":2097152,
    "devices":[
      {
        "chardev":"dax0.1",
        "size":"35.44 GiB (38.05 GB)",
        "target_node":4,
        "mode":"devdax"
      }
    ]
  },
  "align":2097152
}
```

命名空间现在位于 **/dev/dax0.1** 路径。

验证

- 验证系统中的现有命名空间是否已重新配置：

```
# ndctl list --namespace namespace0.1
```

```
[
  {
    "dev":"namespace0.1",
    "mode":"devdax",
    "map":"dev",
    "size":38048628736,
    "uuid":"426d6a52-df92-43d2-8cc7-046241d6d761",
    "chardev":"dax0.1",
    "align":2097152
  }
]
```

其他资源

- **ndctl-create-namespace (1)** 手册页

21.7.3. 在设备 DAX 模式下创建新 NVDIMM 命名空间

如果区域中有可用空间，您可以在非线性内存模块 (NVDIMM) 设备上创建新设备 DAX 命名空间。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。
- NVDIMM 设备支持标签在区域中创建多个命名空间。您可以使用以下命令检查它：

```
# ndctl read-labels nmem0 >/dev/null
read 1 nmem
```

这表示它读取一个 NVDIMM 设备的标签。如果值为 **0**，这表示您的设备不支持标签。

流程

1. 列出系统上有可用空间的 **pmem** 区域。在以下示例中，在 *region1* 和 *region0* 区域中有空间：

```
# ndctl list --regions
[
  {
    "dev":"region1",
    "size":2156073582592,
    "align":16777216,
    "available_size":2117418876928,
    "max_available_extent":2117418876928,
    "type":"pmem",
    "iset_id":-9102197055295954944,
    "badblock_count":1,
    "persistence_domain":"memory_controller"
  },
  {
    "dev":"region0",
    "size":2156073582592,
    "align":16777216,
    "available_size":2143188680704,
    "max_available_extent":2143188680704,
```

```
    "type":"pmem",
    "iset_id":736272362787276936,
    "badblock_count":3,
    "persistence_domain":"memory_controller"
  }
]
```

2. 在任何可用区域上分配一个或多个命名空间：

```
# ndctl create-namespace --mode=devdax --region=region_N_ --size=namespace-size
```

例 21.4. 在区域上创建命名空间

以下命令在 region0 上创建 36-GiB 设备 DAX 命名空间。它与 2-MiB 故障粒度一致，以确保操作系统一次在 2-MiB 页面中故障：

```
# ndctl create-namespace --mode=devdax --region=region0 --align=2M --size=36G
{
  "dev":"namespace0.2",
  "mode":"devdax",
  "map":"dev",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"89d13f41-be6c-425b-9ec7-1e2a239b5303",
  "daxregion":{
    "id":0,
    "size":"35.44 GiB (38.05 GB)",
    "align":2097152,
    "devices":[
      {
        "chardev":"dax0.2",
        "size":"35.44 GiB (38.05 GB)",
        "target_node":4,
        "mode":"devdax"
      }
    ]
  },
  "align":2097152
}
```

命名空间现在作为 **/dev/dax0.2** 提供。

验证

- 验证新命名空间是否在扇区模式中创建：

```
# ndctl list -RN -n namespace0.2
{
  "regions":[
    {
      "dev":"region0",
      "size":2156073582592,
      "align":16777216,
      "available_size":2065879269376,
```

```
    "max_available_extent":2065879269376,
    "type":"pmem",
    "iset_id":736272362787276936,
    "badblock_count":3,
    "persistence_domain":"memory_controller",
    "namespaces":[
     {
      "dev":"namespace0.2",
      "mode":"devdax",
      "map":"dev",
      "size":38048628736,
      "uuid":"89d13f41-be6c-425b-9ec7-1e2a239b5303",
      "chardev":"dax0.2",
      "align":2097152
     }
    ]
   }
  ]
}
```


其他资源

- **ndctl-create-namespace (1)** 手册页

21.8. 在 NVDIMM 上创建文件系统 DAX 命名空间

在文件系统 DAX 模式下配置附加到您系统的 NVDIMM 设备，以支持具有直接访问功能的文件系统。

考虑以下选项：

- 将现有命名空间重新配置为文件系统 DAX 模式。
- 如果有可用空间，请创建新文件系统 DAX 命名空间。

重要

文件系统 DAX 技术仅作为技术预览提供，不受红帽支持。

21.8.1. 文件系统直接访问模式中的 NVDIMM

当在文件系统直接访问（文件系统 DAX、**fsdax**）模式下配置 NVDIMM 设备时，您可以在上面创建一个文件系统。对此文件系统上的文件执行 **mmap()** 操作的任何应用程序都可以直接访问其存储。这启用了 NVDIMM 的直接访问编程模型。

以下新的 **-o dax** 选项现在可用，如果需要，可以通过文件属性控制直接访问行为：

-o dax=inode

当您不要在挂载文件系统时指定任何 dax 选项时，这个选项是默认选项。使用这个选项，您可以在文件上设置属性标志，以控制是否激活 dax 模式。如果需要，您可以在单个文件中设置此标志。
您还可以在一个目录上设置此标志，并且该目录中的任何文件将使用相同的标志创建。您可以使用 **xfs_io -c 'chattr +x'** directory-name 命令设置此属性标志。

-o dax=never

使用此选项时，即使 dax 标记被设置为 **inode** 模式，也不会启用 dax 模式。这意味着，per-inode dax 属性标志会被忽略，使用这个标志设置的文件永远不会启用直接访问。

-o dax=always

这个选项等同于旧的 **-o dax** 行为。使用这个选项，您可以激活文件系统中任何文件的直接访问模式，而不考虑 dax 属性标志。

警告

在以后的发行版本中，**-o dax** 可能不被支持，如果需要，您可以使用 **-o dax=always**。在这个模式中，每个文件都可能处于直接访问模式。

按页元数据分配

这个模式需要在系统 DRAM 或 NVDIMM 设备本身中分配每个页面元数据。每个 4 KiB 页的这个数据结构的开销是 64 字节：

- 在小设备中，开销非常小，足以满足 DRAM 的要求。例如，16-GiB 命名空间只需要 256 MiB 用于页面结构。因为 NVDIMM 设备通常比较小且昂贵，所以最好将页面跟踪数据结构保存在 DRAM 中。
- 在大小为 TB级或更大的 NVDIMM 设备中，存储页面跟踪数据结构所需的内存量可能会超过系统中的 DRAM 数量。一个 TiB 的 NVDIMM 需要 16 GiB 用于页面结构。因此，在这种情况下首选将数据结构存储在 NVDIMM 本身中。
  您可以在配置命名空间时使用 **--map** 选项配置每个页面元数据存储的位置：
- 要在系统 RAM 中分配，请使用 **--map=mem**。
- 要在 NVDIMM 上分配，请使用 **--map=dev**。

21.8.2. 将现有的 NVDIMM 命名空间重新配置为文件系统 DAX 模式

您可以将现有的非易失性双内存模块 (NVDIMM) 命名空间重新配置为文件系统 DAX 模式。

警告

重新配置命名空间会删除之前在命名空间中存储的数据。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。

流程

1. 列出系统中的所有命名空间：

```
# ndctl list --namespaces --idle
[
  {
    "dev":"namespace1.0",
    "mode":"raw",
    "size":34359738368,
    "uuid":"ac951312-b312-4e76-9f15-6e00c8f2e6f4"
    "state":"disabled",
    "numa_node":1
  },
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":38615912448,
    "uuid":"ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
    "state":"disabled",
    "numa_node":0
  }
]
```

2. 重新配置任何命名空间：

```
# ndctl create-namespace --force --mode=fsdax --reconfig=namespace-ID
```

例 21.5. 将命名空间配置为文件系统 DAX

要将 **namespace0.0** 用于支持 DAX 的文件系统，请使用以下命令：

```
# ndctl create-namespace --force --mode=fsdax --reconfig=namespace0.0
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "map":"dev",
  "size":"11.81 GiB (12.68 GB)",
  "uuid":"f8153ee3-c52d-4c6e-bc1d-197f5be38483",
  "sector_size":512,
  "align":2097152,
  "blockdev":"pmem0"
}
```

命名空间现在位于 **/dev/pmem0** 路径中。

验证

- 验证系统中的现有命名空间是否已重新配置：

```
# ndctl list --namespace namespace0.0
[
  {
    "dev":"namespace0.0",
    "mode":"fsdax",
    "map":"dev",
    "size":12681478144,
```

```
    "uuid":"f8153ee3-c52d-4c6e-bc1d-197f5be38483",
    "sector_size":512,
    "align":2097152,
    "blockdev":"pmem0"
  }
]
```

其他资源

- **ndctl-create-namespace (1)** 手册页

21.8.3. 在文件系统 DAX 模式下创建新 NVDIMM 命名空间

如果区域中有可用空间，您可以在非线性内存模块 (NVDIMM) 设备上创建一个新文件系统 DAX 命名空间。

先决条件

- 已安装 **ndctl** 工具。如需更多信息，请参阅安装 ndctl。
- NVDIMM 设备支持标签在区域中创建多个命名空间。您可以使用以下命令检查它：

```
# ndctl read-labels nmem0 >/dev/null
read 1 nmem
```

这表示它读取一个 NVDIMM 设备的标签。如果值为 **0**，这表示您的设备不支持标签。

流程

1. 列出系统上有可用空间的 **pmem** 区域。在以下示例中，在 *region1* 和 *region0* 区域中有空间：

```
# ndctl list --regions
[
  {
    "dev":"region1",
    "size":2156073582592,
    "align":16777216,
    "available_size":2117418876928,
    "max_available_extent":2117418876928,
    "type":"pmem",
    "iset_id":-9102197055295954944,
    "badblock_count":1,
    "persistence_domain":"memory_controller"
  },
  {
    "dev":"region0",
    "size":2156073582592,
    "align":16777216,
    "available_size":2143188680704,
    "max_available_extent":2143188680704,
    "type":"pmem",
    "iset_id":736272362787276936,
    "badblock_count":3,
```

```
    "persistence_domain":"memory_controller"
  }
]
```

2. 在任何可用区域上分配一个或多个命名空间：

```
# ndctl create-namespace --mode=fsdax --region=regionN --size=namespace-size
```

例 21.6. 在区域上创建命名空间

以下命令在 *region0* 上创建 36-GiB 文件系统 DAX 命名空间：

```
# ndctl create-namespace --mode=fsdax --region=region0 --size=36G
{
  "dev":"namespace0.3",
  "mode":"fsdax",
  "map":"dev",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"99e77865-42eb-4b82-9db6-c6bc9b3959c2",
  "sector_size":512,
  "align":2097152,
  "blockdev":"pmem0.3"
}
```

命名空间现在作为 **/dev/pmem0.3** 提供。

验证

- 验证新命名空间是否在扇区模式中创建：

```
# ndctl list -RN -n namespace0.3
{
  "regions":[
    {
      "dev":"region0",
      "size":2156073582592,
      "align":16777216,
      "available_size":2027224563712,
      "max_available_extent":2027224563712,
      "type":"pmem",
      "iset_id":736272362787276936,
      "badblock_count":3,
      "persistence_domain":"memory_controller",
      "namespaces":[
        {
          "dev":"namespace0.3",
          "mode":"fsdax",
          "map":"dev",
          "size":38048628736,
          "uuid":"99e77865-42eb-4b82-9db6-c6bc9b3959c2",
          "sector_size":512,
          "align":2097152,
          "blockdev":"pmem0.3"
```

```
      }
     ]
    }
   ]
}
```

其他资源

- **ndctl-create-namespace (1)** 手册页

21.8.4. 在文件系统 DAX 设备中创建文件系统

您可以在文件系统 DAX 设备中创建文件系统并挂载文件系统。创建文件系统后，应用程序可以使用持久内存并在 *mount-point* 目录中创建文件，打开文件，并使用 **mmap** 操作来映射文件以进行直接访问。

在 Red Hat Enterprise Linux 9 中，作为技术预览可在 NVDIMM 上创建 XFS 和 ext4 文件系统。

流程

1. 可选：在文件系统 DAX 设备中创建一个分区。如需更多信息，请参阅使用 parted 创建分区。

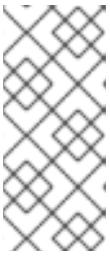
注意

当在 **fsdax** 设备中创建分区时，分区必须在页边界上保持一致。在 Intel 64 和 AMD64 构架中，启动和结束分区至少需要 4 KiB 校对。2 MiB 是首选的校对。

默认情况下，**parted** 工具在 1 MiB 边界上对齐分区。对于第一个分区，指定 2 MiB 作为分区的起点。如果分区的大小是 2 MiB 的倍数，则所有其他分区也都一致。

2. 在分区或者 NVDIMM 设备中创建 XFS 或者 ext4 文件系统：

```
# mkfs.xfs  -d su=2m,sw=1 fsdax-partition-or-device
```



注意

现在，支持 dax 的文件和 reflinked 文件现在可以在文件系统中共存。但是，对于单个文件，dax 和 reflink 是互斥的。

对于 XFS，禁用共享的 copy-on-write 数据扩展，因为它们与 dax 挂载选项不兼容。另外，为了增加大型页面映射的可能性，请设置条带单元和条带宽度。

3. 挂载文件系统：

```
# mount f_sdax-partition-or-device mount-point_
```

不需要使用 dax 选项挂载文件系统来启用直接访问模式。当您在挂载时没有指定 dax 选项时，文件系统处于 **dax=inode** 模式。在激活直接访问模式前，在文件上设置 dax 选项。

其他资源

- **mkfs.xfs (8)** 手册页
- 文件系统直接访问模式中的 NVDIMM

21.9. 使用 S.M.A.R.T 监控 NVDIMM 健康状况。

一些非线性内存模块 (NVDIMM) 设备支持自监控、分析和报告技术 (S.M.A.R.T.) 接口以检索健康信息。

重要

定期监控 NVDIMM 健康状况以防止数据丢失。如果 S.M.A.R.T. 报告 NVDIMM 设备健康状态的问题，请替换它，如弃用并替换有问题的 NVDIMM 设备中所述。

先决条件

- 可选：在一些系统中，上传 **acpi_ipmi** 驱动程序以检索健康信息：

```
# modprobe acpi_ipmi
```

流程

- 访问健康信息：

```
# ndctl list --dimms --health
[
  {
    "dev":"nmem1",
    "id":"8089-a2-1834-00001f13",
    "handle":17,
    "phys_id":32,
    "security":"disabled",
    "health":{
      "health_state":"ok",
      "temperature_celsius":36.0,
      "controller_temperature_celsius":37.0,
      "spares_percentage":100,
      "alarm_temperature":false,
      "alarm_controller_temperature":false,
      "alarm_spares":false,
      "alarm_enabled_media_temperature":true,
      "temperature_threshold":82.0,
      "alarm_enabled_ctrl_temperature":true,
      "controller_temperature_threshold":98.0,
      "alarm_enabled_spares":true,
      "spares_threshold":50,
      "shutdown_state":"clean",
      "shutdown_count":4
    }
  },
[...]
]
```

其他资源

- **ndctl-list (1)** 手册页

21.10. 检测和替换断开问题的 NVDIMM 设备

如果您发现与系统日志中报告的 Non-Volatile Dual In-line Memory Modules (NVDIMM) 相关的错误消息，或者 S.M.A.R.T.，这可能意味着 NVDIMM 设备失败。在这种情况下，需要：

1. 检测哪个 NVDIMM 设备失败
2. 备份保存的数据
3. 物理替换该设备

流程

1. 检测有问题的设备：

```
# ndctl list --dimms --regions --health
{
  "dimms":[
    {
      "dev":"nmem1",
      "id":"8089-a2-1834-00001f13",
      "handle":17,
      "phys_id":32,
      "security":"disabled",
      "health":{
        "health_state":"ok",
        "temperature_celsius":35.0,
        [...]
      }
[...]
}
```

2. 查找有问题的 NVDIMM 的 **phys_id** 属性：

```
# ndctl list --dimms --human
```

在上例中，您知道 **nmem0** 是有问题的 NVDIMM。因此，查找 **nmem0** 的 **phys_id** 属性。

例 21.7. NVDIMM 的 phys_id 属性

在以下示例中，**phys_id** 是 **0x10**:

```
# ndctl list --dimms --human

[
  {
    "dev":"nmem1",
    "id":"XXXX-XX-XXXX-XXXXXXXX",
    "handle":"0x120",
    "phys_id":"0x1c"
  },
  {
    "dev":"nmem0",
    "id":"XXXX-XX-XXXX-XXXXXXXX",
    "handle":"0x20",
    "phys_id":"0x10",
    "flag_failed_flush":true,
```

```
    "flag_smart_event":true
  }
]
```

3. 查找有问题的 NVDIMM 的内存插槽：

```
# dmidecode
```

在输出中，找到 **Handle** 标识符与有问题的 NVDIMM 的 **phys_id** 属性匹配的条目。**Locator** 字段列出了有问题的 NVDIMM 使用的内存插槽。

例 21.8. NVDIMM 内存插槽列表

在以下示例中，**nmem0** 设备与 **0x0010** 标识符匹配，并使用 **DIMM-XXX-YYYY** 内存插槽：

```
# dmidecode

...
Handle 0x0010, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0004
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 125 GB
        Form Factor: DIMM
        Set: 1
        Locator: DIMM-XXX-YYYY
        Bank Locator: Bank0
        Type: Other
        Type Detail: Non-Volatile Registered (Buffered)
...
```

4. 备份 NVDIMM 命名空间中的所有数据。如果您在替换 NVDIMM 前没有备份数据，当您从系统中删除 NVDIMM 时数据将会丢失。

警告

在某些情况下，比如 NVDIMM 完全无法正常工作，备份可能会失败。

要防止这种情况，请使用 S.M.A.R.T.T 定期监控 NVDIMM 设备，如 使用 S.M.A.R.T.T 监控 NVDIMM 健康状况 中所述，并在它们中断前替换失败的 NVDIMM。

5. 列出 NVDIMM 上的命名空间：

```
# ndctl list --namespaces --dimm=DIMM-ID-number
```

例 21.9. NVDIMM 命名空间列表

在以下示例中，**nmem0** 设备包含 **namespace0.0** 和 **namespace0.2** 命名空间，您需要备份：

```
# ndctl list --namespaces --dimm=0

[
  {
    "dev":"namespace0.2",
    "mode":"sector",
    "size":67042312192,
    "uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size":4096,
    "blockdev":"pmem0.2s",
    "numa_node":0
  },
  {
    "dev":"namespace0.0",
    "mode":"sector",
    "size":67042312192,
    "uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size":4096,
    "blockdev":"pmem0s",
    "numa_node":0
  }
]
```

6. 以物理方式替换失效的 NVDIMM。

其他资源

- **ndctl-list (1)** 和 **dmidecode (8)** man page

第 22 章 丢弃未使用块

您可以在支持它们的块设备中执行或调度丢弃操作。块丢弃操作与底层存储进行通信，其中文件系统块不再被挂载的文件系统使用。块丢弃操作允许 SSD 优化垃圾回收例程，它们可以通知精简置备存储重新使用未使用的物理块。

要求

- 基本文件系统的块设备必须支持物理的丢弃（discard）操作。
  如果 **/sys/block/*<device>*/queue/discard_max_bytes** 文件中的值不为零，则支持物理丢弃操作。

22.1. 块丢弃操作的类型

您可以使用不同方法运行 discard 操作：

批量丢弃

由用户明确触发，并丢弃所选文件系统中所有未使用的块。

在线丢弃

在挂载时指定，并在无需用户干预的情况下实时触发。在线丢弃操作只丢弃从 **used** 转换到**free** 状态的块。

定期丢弃

是 **systemd** 服务定期运行的批量操作。

XFS 和 ext4 文件系统支持所有的类型。

建议
红帽建议您使用批处理或周期性丢弃。

仅在以下情况下使用在线丢弃：

- 系统负载不允许使用批量丢弃，或者
- 为了保持性能，需要在线丢弃操作。

22.2. 执行批块丢弃

您可以执行批量块丢弃操作，以丢弃挂载的文件系统上未使用的块。

先决条件

- 挂载文件系统。
- 文件系统底层的块设备支持物理忽略操作。

流程

- 使用 **fstrim** 工具：
  - 要只在所选文件系统中执行丢弃，请使用：

```
# fstrim mount-point
```

- 要在所有挂载的文件系统中执行丢弃，请使用：

```
# fstrim --all
```

如果您在以下设备上执行 **fstrim** 命令：

- 不支持丢弃操作的设备，或者
- 由多个设备组成的逻辑设备（LVM 或者 MD），其中任意设备不支持丢弃操作：

下面的信息将显示：

```
# fstrim /mnt/non_discard

fstrim: /mnt/non_discard: the discard operation is not supported
```

其他资源

- **fstrim(8)** 手册页。

22.3. 启用在线块丢弃

您可以执行在线块丢弃操作，以自动丢弃所有支持的文件系统上未使用的块。

流程

- 在挂载时启用在线丢弃：
  - 手动挂载文件系统时，请添加 **-o discard** 挂载选项：

```
# mount -o discard device mount-point
```

  - 永久挂载文件系统时，请将 **discard** 选项添加到 **/etc/fstab** 文件的挂载条目中。

其他资源

- **mount(8)** 手册页。
- **fstab(5)** 手册页。

22.4. 启用定期块丢弃

您可以启用 **systemd** 计时器来定期丢弃所有支持的文件系统上未使用的块。

流程

- 启用并启动 **systemd** 计时器：

```
# systemctl enable --now fstrim.timer
Created symlink /etc/systemd/system/timers.target.wants/fstrim.timer →
/usr/lib/systemd/system/fstrim.timer.
```

验证

- 验证计时器的状态：

```
# systemctl status fstrim.timer
fstrim.timer - Discard unused blocks once a week
   Loaded: loaded (/usr/lib/systemd/system/fstrim.timer; enabled; vendor preset: disabled)
   Active: active (waiting) since Wed 2023-05-17 13:24:41 CEST; 3min 15s ago
  Trigger: Mon 2023-05-22 01:20:46 CEST; 4 days left
     Docs: man:fstrim

May 17 13:24:41 localhost.localdomain systemd[1]: Started Discard unused blocks once a
week.
```

第 23 章 删除存储设备

您可以从正在运行的系统中安全地删除存储设备，这有助于防止系统内存过载和数据丢失。

先决条件

- 在删除存储设备前，您必须确定您在 I/O 清除过程中因为系统内存负载增加而您有足够的可用内存。使用以下命令查看系统的当前内存负载和可用内存：

```
# vmstat 1 100
# free
```

- 红帽不推荐在以下系统中删除存储设备：
  - 空闲内存低于内存总量的 5%，每 100 个超过 10 个样本。
  - 交换是活跃的（在 **vmstat** 命令的输出中非零的 **si** 和 **so** 列）。

23.1. 安全删除存储设备

从正在运行的系统中安全地删除存储设备需要顶级的方法。从顶层（通常是应用程序或文件系统）开始，并在底层（即物理设备）上工作。

您可以通过多种方式使用存储设备，它们可以在物理设备之上有不同的虚拟配置。例如：您可以将设备的多个实例分组到多路径设备中，使其成为 RAID 的一部分，或者您可以将其成为 LVM 组的一部分。此外，设备可以通过文件系统访问，或者可以直接访问设备，如"原始"设备。

使用 top-to-bottom 方法时，您必须确保：

- 要删除的设备没有被使用
- 对该设备的所有待处理的 I/O 都会被清除
- 操作系统无法引用存储设备

23.2. 删除块设备和相关的元数据

要从正在运行的系统中安全地删除块设备，以防止系统内存过载和数据丢失，您需要首先从它们中删除元数据。从文件系统开始，处理堆栈中的每一层，然后继续处理磁盘。这些操作可防止将您的系统置于不一致的状态。

根据您要删除的设备类型，使用的特定命令可能有所不同：

- **lvremove**、**vgremove** 和 **pvremove** 特定于 LVM。
- 对于软件 RAID，请运行 **mdadm** 以删除阵列。如需更多信息，请参阅 管理 RAID。
- 对于使用 LUKS 加密的块设备，有特定的额外步骤。以下流程对于使用 LUKS 加密的块设备不适用。如需更多信息，请参阅 使用 LUKS 加密块设备。

警告

重新扫描 SCSI 总线或执行更改操作系统状态的其他操作，而无需遵循这个流程，因为 I/O 超时、设备被意外删除或数据丢失。

先决条件

- 您有一个现有的包含文件系统、逻辑卷和卷组的块设备堆栈。
- 您确保没有其他应用程序或服务正在使用您要删除的设备。
- 备份您要删除的设备中的数据。
- 可选：如果要删除多路径设备，且您无法访问其路径设备，请运行以下命令禁用多路径设备的队列：

```
# multipathd disablequeueing map multipath-device
```

这可让设备的 I/O 失败，允许使用该设备的应用程序关闭。

注意

一次一层地删除设备及其元数据可确保不会在磁盘上保留过时的签名。

流程

1. 卸载文件系统：

```
# umount /mnt/mount-point
```

2. 删除文件系统：

```
# wipefs -a /dev/vg0/myvol
```

注意

如果您已在 **/etc/fstab** 文件中添加了一个条目，以在文件系统和挂载点之间进行一个永久关联，您还应在此时编辑 **/etc/fstab** 以删除该条目。

根据您要删除的设备类型，继续执行以下步骤：

3. 删除包含文件系统的逻辑卷(LV)：

```
# lvremove vg0/myvol
```

4. 如果卷组中没有其他的逻辑卷(VG)，您可以安全地删除包含该设备的 VG：

```
# vgremove vg0
```

5. 从 PV 设备中删除物理卷(PV)元数据：

```
# pvremove /dev/sdc1
```

```
# wipefs -a /dev/sdc1
```

6. 删除包含 PV 的分区：

```
# parted /dev/sdc rm 1
```

注意

只有在您要完全擦除该设备时，才按照下面的步骤操作。

7. 删除分区表：

```
# wipefs -a /dev/sdc
```

注意

只有在您想要物理删除该设备时，才按照下面的步骤操作。

- 如果您要删除多路径设备，请执行以下命令：

  a. 查看该设备的所有路径：

  ```
  # multipath -l
  ```

  稍后需要这个命令的输出。

  i. 清除 I/O 并删除多路径设备：

  ```
  # multipath -f multipath-device
  ```

- 如果该设备没有配置为多路径设备，或者设备配置为多路径设备，并且您之前将 I/O 传递给单个路径，请将任何未完成的 I/O 刷新到所有使用的设备路径：

  ```
  # blockdev --flushbufs device
  ```

  对于直接访问的设备非常重要，**umount** 或 **vgreduce** 命令不会清除 I/O。

- 如果您要删除 SCSI 设备，请执行以下命令：

  a. 删除对基于路径的设备名称的任何引用，如 ***/dev/sd***、***/dev/disk/by-path*** 或 **major:minor** number（在系统上的应用程序、脚本或工具中）。这样可保证以后添加的不同设备不会为当前的设备错误。

  b. 从 SCSI 子系统中删除该设备的每个路径：

  ```
  # echo 1 > /sys/block/device-name/device/delete
  ```

此处，如果设备之前被用作多路径设备，则 ***device-name*** 可从 **multipath -l** 命令的输出中检索到。

8. 从正在运行的系统中删除物理设备。请注意，当您删除此设备时，I/O 到其它设备不会停止。

验证

- 验证您要删除的设备是否没有在 **lsblk** 命令的输出中显示。以下是一个输出示例：

```
# lsblk

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0    5G  0 disk
sr0     11:0    1 1024M  0 rom
vda    252:0    0   10G  0 disk
|-vda1 252:1    0    1M  0 part
|-vda2 252:2    0  100M  0 part /boot/efi
`-vda3 252:3    0  9.9G  0 part /
```

其他资源

- **multipath(8)**, **pvremove(8)**, **vgremove(8)**, **lvremove(8)**, **wipefs(8)**, **parted(8)**, **blockdev(8)** 和 **umount(8)** 手册页。

第 24 章 设置 STRATIS 文件系统

Stratis 作为服务运行，来管理物理存储设备池，简化本地存储管理，易于使用，同时帮助您设置和管理复杂的存储配置。

24.1. 什么是 STRATIS

Stratis 是 Linux 的本地存储管理解决方案。它着重说明简单性和易用性，并可让您访问高级存储功能。

Stratis 使以下操作更为容易：

- 存储的初始配置
- 稍后进行修改
- 使用高级存储功能

Stratis 是一个支持高级存储功能的本地存储管理系统。Stratis 的核心概念是一个存储 *池*。这个池是从一个或多个本地磁盘或分区创建的，文件系统是从池创建的。

池启用了许多有用的功能，例如：

- 文件系统快照
- 精简置备
- 等级
- 加密

其它资源

- Stratis 网站

24.2. STRATIS 卷的组件

了解组成 Stratis 卷的组件。

另外，Stratis 在命令行界面和 API 中显示以下卷组件：

blockdev

块设备，如磁盘或者磁盘分区。

pool

由一个或多个块设备组成。
池有固定大小，与块设备的大小相等。

池包含大多数 Stratis 层，如使用 **dm-cache** 目标的非易失性数据缓存。

Stratis 为每个池创建一个 **/dev/stratis/*my-pool*/** 目录。这个目录包含了到代表池里 Stratis 文件系统的设备的链接。

filesystem

每个池可以包含一个或多个文件系统来存储文件。

文件系统会被精简置备，且没有固定的总大小。文件系统的实际大小随着保存着文件系统中的数据而增长。如果数据的大小接近文件系统的虚拟大小，Stratis 将自动增大精简卷和文件系统。

文件系统使用 XFS 格式化。

重要

Stratis 跟踪关于使用 Stratis 创建的文件系统的信息，但 XFS 并不知道，并且使用 XFS 进行的更改不会在 Stratis 中自动创建更新。用户不得重新格式化或重新配置由 Stratis 管理的 XFS 文件系统。

Stratis 在 **/dev/stratis/*my-pool*/*my-fs*** 路径创建到文件系统的链接。

注意

Stratis 使用许多设备映射器设备，显示在 **dmsetup** 列表中和 **/proc/partitions** 文件中。类似地，**lsblk** 命令输出反映了 Stratis 的内部工作方式和层。

24.3. 可用于 STRATIS 的块设备

可与 Stratis 一起使用的存储设备。

支持的设备
Stratis 池已被测试以可用于这些块设备：

- LUKS
- LVM 逻辑卷
- MD RAID
- DM Multipath
- iSCSI
- HDD 和 SSD
- NVMe 设备

不支持的设备
因为 Stratis 包含精简置备层，因此红帽不推荐将 Stratis 池放在已经精简置备的块设备中。

24.4. 安装 STRATIS

安装 Stratis 所需的软件包。

流程

1. 安装提供 Stratis 服务和命令行工具的软件包：

```
# dnf install stratisd stratis-cli
```

2. 验证 **stratisd** 服务是否已启用：

```
# systemctl enable --now stratisd
```

24.5. 创建未加密的 STRATIS 池

您可以从一个或多个块设备创建未加密的 Stratis 池。

先决条件

- 已安装 Stratis。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 创建 Stratis 池的块设备没有被使用，且没有被挂载。
- 要在其上创建 Stratis 池的每个块设备至少为 1 GB。
- 在 IBM Z 构架中，必须对 **/dev/dasd*** 块设备进行分区。使用分区设备来创建 Stratis 池。

有关分区 DASD 设备的详情，请参考 在 IBM Z 上配置 Linux 实例 。

注意

您无法加密未加密的 Stratis 池。

流程

1. 删除您要在 Stratis 池中使用的每个块设备上存在的任何文件系统、分区表或 RAID 签名：

   ```
   # wipefs --all block-device
   ```

   其中 ***block-device*** 是块设备的路径；例如，**/dev/sdb**。

2. 在所选的块设备上创建新的未加密的 Stratis 池：

   ```
   # stratis pool create my-pool block-device
   ```

   其中 ***block-device*** 是到空或已擦除的块设备的路径。

   注意

   在一行中指定多个块设备：

   ```
   # stratis pool create my-pool block-device-1 block-device-2
   ```

3. 确认创建了新的 Stratis 池：

   ```
   # stratis pool list
   ```

24.6. 创建一个加密的 STRATIS 池

要保护您的数据，您可以从一个或多个块设备创建一个加密的 Stratis 池。

当您创建加密的 Stratis 池时，内核密钥环将用作主加密机制。后续系统重启此内核密钥环后，用来解锁加密的 Stratis 池。

当从一个或多个块设备创建加密的 Stratis 池时，请注意以下几点：

- 每个块设备都使用 **cryptsetup** 库进行加密，并实施 **LUKS2** 格式。
- 每个 Stratis 池都可以有一个唯一的密钥，或者与其他池共享相同的密钥。这些密钥保存在内核密钥环中。
- 组成 Stratis 池的块设备必须全部加密或者全部未加密。不可能同时在同一个 Stratis 池中加密和未加密块设备。
- 添加到加密 Stratis 池的数据层中的块设备会自动加密。

先决条件

- Stratis v2.1.0 或更高版本已安装。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 创建 Stratis 池的块设备没有被使用，且没有被挂载。
- 在其上创建 Stratis 池的每个块设备至少为 1GB。
- 在 IBM Z 构架中，必须对 **/dev/dasd*** 块设备进行分区。使用 Stratis 池中的分区。

有关分区 DASD 设备的详情，请参考 在 IBM Z 上配置 Linux 实例 。

流程

1. 删除您要在 Stratis 池中使用的每个块设备上存在的任何文件系统、分区表或 RAID 签名：

```
# wipefs --all block-device
```

其中 ***block-device*** 是块设备的路径；例如，**/dev/sdb**。

2. 如果您还没有创建密钥集，请运行以下命令，并按照提示创建用于加密的密钥集。

```
# stratis key set --capture-key key-description
```

其中 ***key-description*** 是对在内核密钥环中创建的密钥的引用。

3. 创建加密的 Stratis 池并指定用于加密的密钥描述。您还可以使用 **--keyfile-path** 选项指定密钥路径，而不是使用 ***key-description*** 选项。

```
# stratis pool create --key-desc key-description my-pool block-device
```

其中

key-description
引用您在上一步中创建的内核密钥环中存在的密钥。

my-pool
指定新的 Stratis 池的名称。

block-device

指定到空或者有线块设备的路径。

注意

在一行中指定多个块设备：

```
# stratis pool create --key-desc key-description my-pool block-device-1 block-device-2
```

4. 确认创建了新的 Stratis 池：

```
# stratis pool list
```

24.7. 在 STRATIS 文件系统中设置过度置备模式

存储堆栈可以到达过度置备的状态。如果文件系统大小比提供支持的池变大，则池会变得满。要防止这种情况，禁用 overprovisioning，这样可确保池提供的所有文件系统的大小不会超过池提供的可用物理存储。如果您将 Stratis 用于关键应用程序或 root 文件系统，则这个模式会阻止某些失败情况。

如果您启用过度置备，API 信号会在存储被完全分配时通知您。通知充当用户警告，通知他们当所有剩余的池空间填满时，Stratis 没有空间被扩展。

先决条件

- 已安装 Stratis。如需更多信息，请参阅 安装 Stratis。

流程

要正确设置池，您可以有两个可能：

1. 从一个或多个块设备创建池：

```
# stratis pool create --no-overprovision pool-name /dev/sdb
```

   - 通过使用 **--no-overprovision** 选项，池无法分配比实际可用的物理空间更多的逻辑空间。

2. 在现有池中设置过度置备模式：

```
# stratis pool overprovision pool-name <yes|no>
```

   - 如果设置为 "yes"，则启用过度置备到池。这意味着池支持的 Stratis 文件系统的逻辑大小总和可能会超过可用空间量。

验证

1. 运行以下命令来查看 Stratis 池的完整列表：

```
# stratis pool list
Name       Total Physical                Properties   UUID                                 Alerts
```

```
pool-name     1.42 TiB / 23.96 MiB / 1.42 TiB   ~Ca,~Cr,~Op    cb7cb4d8-9322-4ac4-a6fd-eb7ae9e1e540
```

2. 检查 **stratis pool list** 输出中是否有池 overprovisioning 模式标记。" ~ " 是 "NOT" 的数学符号，因此 **~Op** 表示不进行过度配置。

3. 可选： 运行以下内容来检查特定池的过度置备：

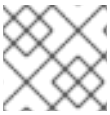
```
# stratis pool overprovision pool-name yes

# stratis pool list

Name          Total Physical                 Properties     UUID                                   Alerts
pool-name     1.42 TiB / 23.96 MiB / 1.42 TiB   ~Ca,~Cr,~Op    cb7cb4d8-9322-4ac4-a6fd-eb7ae9e1e540
```

其他资源

- *Stratis 存储* 网页。

24.8. 将 STRATIS 池绑定到 NBDE

将加密的 Stratis 池绑定到网络绑定磁盘加密(NBDE)需要 Tang 服务器。当包含 Stratis 池的系统重启时，它与 Tang 服务器进行连接，以自动解锁加密的池，而无需提供内核密钥环描述。



注意

将 Stratis 池绑定到补充的 Clevis 加密机制不会删除主内核密钥环加密。

先决条件

- Stratis v2.3.0 或更高版本已安装。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了加密的 Stratis 池，并且拥有用于加密的密钥的密钥描述。如需更多信息，请参阅创建加密的 Stratis 池。
- 您可以连接到 Tang 服务器。如需更多信息，请参阅 部署 SELinux 为 enforcing 模式的 Tang 服务器

流程

- 将加密的 Stratis 池绑定到 NBDE：

```
# stratis pool bind nbde --trust-url my-pool tang-server
```

其中

my-pool
: 指定加密的 Stratis 池的名称。

tang-server
: 指定 Tang 服务器的 IP 地址或 URL。

其它资源

- 使用基于策略的解密配置加密卷的自动解锁

24.9. 将 STRATIS 池绑定到 TPM

当您将加密的 Stratis 池绑定到受信任的平台模块(TPM) 2.0 时，包含池的系统会重启，并且池会自动解锁，而无需提供内核 keyring 描述。

先决条件

- Stratis v2.3.0 或更高版本已安装。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了一个加密的 Stratis 池。如需更多信息，请参阅创建加密的 Stratis 池。

流程

- 将加密的 Stratis 池绑定到 TPM:

```
# stratis pool bind tpm my-pool key-description
```

其中

my-pool
指定加密的 Stratis 池的名称。

key-description
引用内核密钥环中存在的密钥，该密钥是在您创建加密的 Stratis 池时生成的。

24.10. 使用内核密钥环解加密的 STRATIS 池

系统重启后，您的加密 Stratis 池或组成它的块设备可能不可见。您可以使用用来加密池的内核密钥环来解锁池。

先决条件

- Stratis v2.1.0 已安装。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了一个加密的 Stratis 池。如需更多信息，请参阅创建加密的 Stratis 池。

流程

1. 使用之前使用的相同密钥描述重新创建密钥集：

```
# stratis key set --capture-key key-description
```

其中 *key-description* 引用内核密钥环中存在的密钥，该密钥是您在创建加密的 Stratis 池时生成的。

2. 验证 Stratis 池是可见的：

```
# stratis pool list
```

24.11. 解除 STRATIS 池与补充加密的绑定

当您解除加密的 Stratis 池与支持的附加加密机制的绑定时，主内核密钥环加密将保持不变。对于从一开始就使用 Clevis 加密创建的池，情况并非如此。

先决条件

- Stratis v2.3.0 或更高版本已安装在您的系统上。如需更多信息，请参阅 安装 Stratis。
- 您已创建了一个加密的 Stratis 池。如需更多信息，请参阅创建加密的 Stratis 池。
- 加密的 Stratis 池绑定到受支持的补充加密机制。

流程

- 解除加密的 Stratis 池与补充加密机制的绑定：

  ```
  # stratis pool unbind clevis my-pool
  ```

  其中

  my-pool 指定您要解绑的 Stratis 池的名称。

其它资源

- 将加密的 Stratis 池绑定到 NBDE
- 将加密的 Stratis 池绑定到 TPM

24.12. 启动和停止 STRATIS 池

您可以启动和停止 Stratis 池。这可让您选择破坏或关闭用于构建池的所有对象，如文件系统、缓存设备、精简池和加密设备。请注意，如果池主动使用任何设备或文件系统，它可能发出警告且无法停止。

停止的状态记录在池的元数据中。这些池不会在以下引导上启动，直到池收到 start 命令。

先决条件

- 已安装 Stratis。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了未加密的或者加密的 Stratis 池。请参阅 创建未加密的 Stratis 池

或 创建加密的 Stratis 池。

流程

- 使用以下命令启动 Stratis 池。**--unlock-method** 选项指定池被加密的解锁方法：

  ```
  # stratis pool start pool-uuid --unlock-method <keyring|clevis>
  ```

- 另外，使用以下命令停止 Stratis 池。这会关闭存储堆栈，但保留所有元数据不变：

```
# stratis pool stop pool-name
```

验证步骤

- 使用以下命令列出系统中的所有池：

```
# stratis pool list
```

- 使用以下命令列出所有不是之前启动的池。如果指定了 UUID，该命令会打印与 UUID 对应的池的详细信息：

```
# stratis pool list --stopped --uuid UUID
```

24.13. 创建 STRATIS 文件系统

在现有 Stratis 池上创建 Stratis 文件系统。

先决条件

- 已安装 Stratis。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 池。请参阅 创建未加密的 Stratis 池

或 创建加密的 Stratis 池。

流程

1. 要在池中创建 Stratis 文件系统，请使用：

```
# stratis filesystem create --size number-and-unit my-pool my-fs
```

其中

number-and-unit
指定文件系统的大小。规格格式必须遵循标准大小规格格式进行输入，即 B、KiB、MiB、GiB、TiB 或 PiB。

my-pool
指定 Stratis 池的名称。

my-fs
为文件系统指定一个任意名称。
例如：

例 24.1. 创建 Stratis 文件系统

```
# stratis filesystem create --size 10GiB pool1 filesystem1
```

验证步骤

- 列出池中的文件系统，以检查是否创建了 Stratis 文件系统：

```
# stratis fs list my-pool
```

其它资源

- 挂载 Stratis 文件系统.

24.14. 挂载 STRATIS 文件系统

挂载现有的 Stratis 文件系统以访问其内容。

先决条件

- 已安装 Stratis。如需更多信息，请参阅 安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 文件系统。如需更多信息，请参阅 创建 Stratis 文件系统。

流程

- 要挂载文件系统，请使用 Stratis 在 **/dev/stratis/** 目录中维护的条目：

```
# mount /dev/stratis/my-pool/my-fs mount-point
```

现在该文件系统被挂载到 *mount-point* 目录中并可使用。

其它资源

- 创建 Stratis 文件系统 。

24.15. 永久挂载 STRATIS 文件系统

这个过程永久挂载 Stratis 文件系统，以便在引导系统后自动可用。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 文件系统。请参阅 创建 Stratis 文件系统。

流程

1. 确定文件系统的 UUID 属性：

```
$ lsblk --output=UUID /dev/stratis/my-pool/my-fs
```

例如：

例 24.2. 查看 Stratis 文件系统的 UUID

```
$ lsblk --output=UUID /dev/stratis/my-pool/fs1

UUID
a1f0b64a-4ebb-4d4e-9543-b1d79f600283
```

2. 如果挂载点目录不存在，请创建它：

```
# mkdir --parents mount-point
```

3. 以 root 用户身份，编辑 **/etc/fstab** 文件，并为文件系统添加一行，由 UUID 标识。使用 **xfs** 作为文件系统类型，并添加 **x-systemd.requires=stratisd.service** 选项。
例如：

例 24.3. /etc/fstab 中的 /fs1 挂载点

```
UUID=a1f0b64a-4ebb-4d4e-9543-b1d79f600283 /fs1 xfs defaults,x-
systemd.requires=stratisd.service 0 0
```

4. 重新生成挂载单元以便您的系统注册新配置：

```
# systemctl daemon-reload
```

5. 尝试挂载文件系统来验证配置是否正常工作：

```
# mount mount-point
```

其它资源

- 永久挂载文件系统

24.16. 使用 SYSTEMD 服务在 /ETC/FSTAB 中设置非 ROOT STRATIS 文件系统

您可以使用 systemd 服务管理 /etc/fstab 中的非 root 文件系统。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 文件系统。请参阅 创建 Stratis 文件系统。

流程

- 对于所有非 root Stratis 文件系统，请使用：

```
# /dev/stratis/[STRATIS_SYMLINK] [MOUNT_POINT] xfs defaults, x-
systemd.requires=stratis-fstab-setup@[POOL_UUID].service,x-systemd.after=stratis-stab-
setup@[POOL_UUID].service <dump_value> <fsck_value>
```

其它资源

- 永久挂载文件系统。

第 25 章 使用额外块设备扩展 STRATIS 卷

您可以在 Stratis 池中添加附加块设备以便为 Stratis 文件系统提供更多存储容量。

25.1. STRATIS 卷的组件

了解组成 Stratis 卷的组件。

另外，Stratis 在命令行界面和 API 中显示以下卷组件：

blockdev

块设备，如磁盘或者磁盘分区。

pool

由一个或多个块设备组成。
池有固定大小，与块设备的大小相等。

池包含大多数 Stratis 层，如使用 **dm-cache** 目标的非易失性数据缓存。

Stratis 为每个池创建一个 **/dev/stratis/*my-pool*/** 目录。这个目录包含了到代表池里 Stratis 文件系统的设备的链接。

filesystem

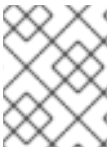
每个池可以包含一个或多个文件系统来存储文件。
文件系统会被精简置备，且没有固定的总大小。文件系统的实际大小随着保存着文件系统中的数据而增长。如果数据的大小接近文件系统的虚拟大小，Sratis 将自动增大精简卷和文件系统。

文件系统使用 XFS 格式化。

重要

Stratis 跟踪关于使用 Stratis 创建的文件系统的信息，但 XFS 并不知道，并且使用 XFS 进行的更改不会在 Stratis 中自动创建更新。用户不得重新格式化或重新配置由 Stratis 管理的 XFS 文件系统。

Stratis 在 **/dev/stratis/*my-pool*/*my-fs*** 路径创建到文件系统的链接。



注意

Stratis 使用许多设备映射器设备，显示在 **dmsetup** 列表中和 **/proc/partitions** 文件中。类似地，**lsblk** 命令输出反映了 Stratis 的内部工作方式和层。

25.2. 在 STRATIS 池中添加块设备

此流程在 Stratis 池中添加一个或多个块设备，供 Stratis 文件系统使用。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。

- 要添加到 Stratis 池中的块设备不会被使用且没有挂载。
- 要添加到 Stratis 池中的块设备的大小至少为 1 GiB。

流程

- 要在池中添加一个或多个块设备，请使用：

```
# stratis pool add-data my-pool device-1 device-2 device-n
```

其它资源

- **Stratis(8)** 手册页

25.3. 其它资源

- *Stratis 存储* 网站

第 26 章 监控 STRATIS 文件系统

作为 Stratis 用户，您可以查看系统中 Stratis 卷的信息，以监控其状态和剩余空间。

26.1. 不同工具报告的 STRATIS 大小

本节解释了标准工具（如 **df**）和 **stratis** 工具所报告的 Stratis 大小之间的区别。

标准 Linux 工具（如 **df**）报告 Stratis 上 的 XFS 文件系统层的大小，其为 1 TiB。这不是有用的信息，因为由于精简资源调配， Stratis 的实际存储使用率较少，而且在 XFS 层接近满了的时候，Stratis 会自动增加文件系统。

重要

定期监控写入 Stratis 文件系统的数据量，将其报告为 *总物理使用* 值。请确定没有超过 *总计物理大小* 值。

其它资源

- **Stratis(8)** 手册页。

26.2. 显示关于 STRATIS 卷的信息

此流程列出了您的 Stratis 卷的统计信息，如总数、使用量、可用大小、文件系统以及属于池中的块设备。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。

流程

- 要显示系统中用于 Stratis 的所有块设备的信息：

```
# stratis blockdev

Pool Name  Device Node    Physical Size   State  Tier
my-pool    /dev/sdb           9.10 TiB  In-use  Data
```

- 显示系统中所有 Stratis **池**的信息：

```
# stratis pool

Name    Total Physical Size  Total Physical Used
my-pool          9.10 TiB             598 MiB
```

- 显示系统中所有 Stratis **文件系统**的信息：

```
# stratis filesystem

Pool Name  Name  Used     Created           Device
```

```
my-pool    my-fs 546 MiB  Nov 08 2018 08:03  /dev/stratis/my-pool/my-fs
```

其它资源

- **Stratis(8)** 手册页。

26.3. 其他资源

- *Stratis 存储* 网站

第 27 章 在 STRATIS 文件系统中使用快照

您可以使用 Stratis 文件系统的快照任意时间捕获文件系统状态，并在以后恢复它。

27.1. STRATIS 快照的特性

在 Stratis 中，快照是作为另一个 Stratis 文件系统的副本创建的常规 Stratis 文件系统。快照最初包含与原始文件系统相同的文件内容，但可以随快照的更改而改变。您对快照所做的任何修改都不会反映在原始文件系统中。

Stratis 中的当前快照实现的特征如下：

- 文件系统快照是另一个文件系统。
- 快照及其原始卷在生命周期中不会被链接。快照的文件系统可以比它从中创建的文件系统更长。
- 文件系统不一定被挂载来生成快照。
- 每个快照使用大约一半的实际后备存储，这是 XFS 日志所需要的。

27.2. 创建 STRATIS 快照

这个过程会创建一个 Stratis 文件系统作为现有 Stratis 文件系统的快照。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 文件系统。请参阅 创建 Stratis 文件系统。

流程

- 要创建 Stratis 快照，请使用：

```
# stratis fs snapshot my-pool my-fs my-fs-snapshot
```

其它资源

- **Stratis(8)** 手册页。

27.3. 访问 STRATIS 快照的内容

这个过程挂载 Stratis 文件系统的快照，使其可在读写操作中访问。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 快照。请参阅 创建 Stratis 文件系统。

流程

- 要访问快照，请将其作为常规文件系统挂载到 **/dev/stratis/*my-pool*/** 目录：

```
# mount /dev/stratis/my-pool/my-fs-snapshot mount-point
```

其它资源

- 挂载 Stratis 文件系统.
- **mount(8)** 手册页。

27.4. 将 STRATIS 文件系统恢复到以前的快照

这个过程将 Stratis 文件系统的内容恢复到 Stratis 快照中捕获的状态。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 快照。请参阅创建 Stratis 快照。

流程

1. 另外，备份文件系统的当前状态，以便以后可以访问它：

```
# stratis filesystem snapshot my-pool my-fs my-fs-backup
```

2. 卸载并删除原始文件系统：

```
# umount /dev/stratis/my-pool/my-fs
# stratis filesystem destroy my-pool my-fs
```

3. 在原始文件系统名称下创建快照副本：

```
# stratis filesystem snapshot my-pool my-fs-snapshot my-fs
```

4. 挂载快照，它现在可以和原始文件系统的名称相同：

```
# mount /dev/stratis/my-pool/my-fs mount-point
```

名为 *my-fs* 的文件系统的内容与快照 *my-fs-snapshot* 一致。

其它资源

- **Stratis(8)** 手册页。

27.5. 删除 STRATIS 快照

这个过程从池中删除 Stratis 快照。快照中的数据会丢失。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 快照。请参阅创建 Stratis 快照。

流程

1. 卸载快照：

```
# umount /dev/stratis/my-pool/my-fs-snapshot
```

2. 销毁快照：

```
# stratis filesystem destroy my-pool my-fs-snapshot
```

其它资源

- **Stratis(8)** 手册页。

27.6. 其他资源

- *Stratis 存储* 网站

第 28 章 删除 STRATIS 文件系统

您可以通过销毁其上的数据来删除现有的 Stratis 文件系统或 Stratis 池。

28.1. STRATIS 卷的组件

了解组成 Stratis 卷的组件。

另外，Stratis 在命令行界面和 API 中显示以下卷组件：

blockdev

块设备，如磁盘或者磁盘分区。

pool

由一个或多个块设备组成。
池有固定大小，与块设备的大小相等。

池包含大多数 Stratis 层，如使用 **dm-cache** 目标的非易失性数据缓存。

Stratis 为每个池创建一个 **/dev/stratis/*my-pool*/** 目录。这个目录包含了到代表池里 Stratis 文件系统的设备的链接。

filesystem

每个池可以包含一个或多个文件系统来存储文件。
文件系统会被精简置备，且没有固定的总大小。文件系统的实际大小随着保存着文件系统中的数据而增长。如果数据的大小接近文件系统的虚拟大小，Sratis 将自动增大精简卷和文件系统。

文件系统使用 XFS 格式化。

重要

Stratis 跟踪关于使用 Stratis 创建的文件系统的信息，但 XFS 并不知道，并且使用 XFS 进行的更改不会在 Stratis 中自动创建更新。用户不得重新格式化或重新配置由 Stratis 管理的 XFS 文件系统。

Stratis 在 **/dev/stratis/*my-pool*/*my-fs*** 路径创建到文件系统的链接。

注意

Stratis 使用许多设备映射器设备，显示在 **dmsetup** 列表中和 **/proc/partitions** 文件中。类似地，**lsblk** 命令输出反映了 Stratis 的内部工作方式和层。

28.2. 删除 STRATIS 文件系统

这个过程删除现有的 Stratis 文件系统。保存的数据会丢失。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。

- 您已创建了 Stratis 文件系统。请参阅 创建 Stratis 文件系统。

流程

1. 卸载文件系统：

```
# umount /dev/stratis/my-pool/my-fs
```

2. 销毁文件系统：

```
# stratis filesystem destroy my-pool my-fs
```

3. 验证文件系统不再存在：

```
# stratis filesystem list my-pool
```

其它资源

- **Stratis(8)** 手册页。

28.3. 删除 STRATIS 池

此流程删除现有的 Stratis 池。保存的数据会丢失。

先决条件

- 已安装 Stratis。请参阅安装 Stratis。
- **stratisd** 服务在运行。
- 您已创建了 Stratis 池：
  - 要创建未加密的池，请参阅 创建未加密的 Stratis 池
  - 要创建加密的池，请参阅创建加密的 Stratis 池。

流程

1. 列出池中的文件系统：

```
# stratis filesystem list my-pool
```

2. 卸载池中的所有文件系统：

```
# umount /dev/stratis/my-pool/my-fs-1 \
        /dev/stratis/my-pool/my-fs-2 \
        /dev/stratis/my-pool/my-fs-n
```

3. 销毁文件系统：

```
# stratis filesystem destroy my-pool my-fs-1 my-fs-2
```

4. 销毁池：

```
# stratis pool destroy my-pool
```

5. 验证池不再存在：

```
# stratis pool list
```

其它资源

- **Stratis(8)** 手册页。

28.4. 其他资源

- *Stratis 存储* 网站