Red Hat Enterprise Linux 8

Managing storage devices

Deploying and configuring single-node storage in Red Hat Enterprise Linux 8

Last Updated: 2019-05-21

Red Hat Enterprise Linux 8 Managing storage devices

Deploying and configuring single-node storage in Red Hat Enterprise Linux 8

Legal Notice

Abstract

This documentation collection provides instructions on how to effectively manage storage devices in Red Hat Enterprise Linux 8.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. GETTING STARTED WITH PARTITIONS

As a system administrator, you can use the following procedures to create, delete, and modify various types of disk partitions.

For an overview of the advantages and disadvantages to using partitions on block devices, see the following KBase article: https://access.redhat.com/solutions/163853.

1.1. VIEWING THE PARTITION TABLE

As a system administrator, you can display the partition table of a block device to see the partition layout and details about individual partitions.

1.1.1. Viewing the partition table with parted

This procedure describes how to view the partition table on a block device using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device you want to examine: for example, **/dev/sda**.

2. View the partition table:

```
(parted) print
```

3. Optionally, use the following command to switch to another device you want to examine next:

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

Additional resources

- The **parted(8)** man page.

1.1.2. Example output of parted print

This section provides an example output of the **print** command in the **parted** shell and describes fields in the output.

Example 1.1. Output of the print command

```
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
```

Following is a description of the fields:

Model: ATA SAMSUNG MZNLN256 (scsi)

The disk type, manufacturer, model number, and interface.

Disk /dev/sda: 256GB

The file path to the block device and the storage capacity.

Partition Table: msdos

The disk label type.

Number

The partition number. For example, the partition with minor number 1 corresponds to **/dev/sda1**.

Start and **End**

The location on the device where the partition starts and ends.

Type

Valid types are metadata, free, primary, extended, or logical.

File system

The file system type. If the **File system** field of a device shows no value, this means that its file system type is unknown. The **parted** utility cannot recognize the file system on encrypted devices.

Flags

Lists the flags set for the partition. Available flags are **boot**, **root**, **swap**, **hidden**, **raid**, **lvm**, or **lba**.

1.2. CREATING A PARTITION TABLE ON A DISK

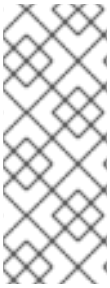
As a system administrator, you can format a block device with different types of partition tables to enable using partitions on the device.

WARNING

Formatting a block device with a partition table deletes all data stored on the device.

1.2.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.



NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.2.2. Comparison of partition table types

This section compares the properties of different types of partition tables that you can create on a block device.

Table 1.1. Partition table types

Partition table	Maximum number of partitions	Maximum partition size
Master Boot Record (MBR)	4 primary, or 3 primary and 12 logical inside an extended partition	2TiB
GUID Partition Table (GPT)	128	8ZiB

1.2.3. Creating a partition table on a disk with parted

This procedure describes how to format a block device with a partition table using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition table: for example, /**dev**/**sda**.

2. Determine if there already is a partition table on the device:

```
(parted) print
```

   If the device already contains partitions, they will be deleted in the next steps.

3. Create the new partition table:

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

   - Replace *table-type* with with the intended partition table type:
     - **msdos** for MBR
     - **gpt** for GPT

   Example 1.2. Creating a GPT table

   For example, to create a GPT table on the disk, use:

```
(parted) mklabel gpt
```

   The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the partition table exists:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

Additional resources

- The **parted(8)** man page.

Next steps

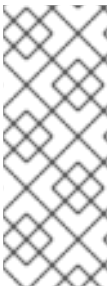
- Create partitions on the device. See Section 1.3, "Creating a partition" for details.

1.3. CREATING A PARTITION

As a system administrator, you can create new partitions on a disk.

1.3.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.



NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.3.2. Partition types

This section describes different attributes that specify the type of a partition.

Partition types or flags
The partition type, or flag, is used by a running system only rarely. However, the partition type matters to on-the-fly generators, such as **systemd-gpt-auto-generator**, which use the partition type to, for example, automatically identify and mount devices.

- The **parted** utility provides some control of partition types by mapping the partition type to *flags*. The parted utility can handle only certain partition types: for example LVM, swap, or RAID.
- The **fdisk** utility supports the full range of partition types by specifying hexadecimal codes.

Partition file system type
The **parted** utility optionally accepts a file system type argument when creating a partition. The value is used to:

- Set the partition flags on MBR, or
- Set the partition UUID type on GPT. For example, the **swap**, **fat**, or **hfs** file system types set different GUIDs. The default value is the Linux Data GUID.

The argument does not modify the file system on the partition in any way. It only differentiates between the supported flags or GUIDs.

The following file system types are supported:

- **xfs**
- **ext2**
- **ext3**
- **ext4**
- **fat16**
- **fat32**
- **hfs**
- **hfs+**

- **linux-swap**
- **ntfs**
- **reiserfs**

1.3.3. Creating a partition with parted

This procedure describes how to create a new partition on a block device using the **parted** utility.

Prerequisites

- There is a partition table on the disk. For details on how to format the disk, see Section 1.2, "Creating a partition table on a disk".
- If the partition you want to create is larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT).

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition: for example, /**dev**/**sda**.

2. View the current partition table to determine if there is enough free space:

```
(parted) print
```

   - If there is not enough free space, you can resize an existing partition. For more information, see Section 1.5, "Resizing a partition" .
   - From the partition table, determine:
     - The start and end points of the new partition
     - On MBR, what partition type it should be.

3. Create the new partition:

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

   - Replace *part-type* with with **primary**, **logical**, or **extended** based on what you decided from the partition table. This applies only to the MBR partition table.
   - Replace *name* with an arbitrary partition name. This is required for GPT partition tables.
   - Replace *fs-type* with any one of **xfs**, **ext2**, **ext3**, **ext4**, **fat16**, **fat32**, **hfs**, **hfs+**, **linux-swap**, **ntfs**, or **reiserfs**. The *fs-type* parameter is optional. Note that **parted** does not create the file system on the partition.
   - Replace *start* and *end* with the sizes that determine the starting and ending points of the partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

Example 1.3. Creating a small primary partition

For example, to create a primary partition from 1024MiB until 2048MiB on an MBR table, use:

```
(parted) mkpart primary 1024MiB 2048MiB
```

The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the created partition is in the partition table with the correct partition type, file system type, and size:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Use the following command to wait for the system to register the new device node:

```
# udevadm settle
```

7. Verify that the kernel recognizes the new partition:

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

Additional resources

- The **parted(8)** man page.

1.3.4. Setting a partition type with fdisk

This procedure describes how to set a partition type, or flag, using the **fdisk** utility.

Prerequisites

- There is a partition on the disk.

Procedure

1. Start the interactive **fdisk** shell:

```
# fdisk block-device
```

   - Replace *block-device* with the path to the device where you want to set a partition type: for example, /**dev**/**sda**.

2. View the current partition table to determine the minor partition number:

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

You can see the current partition type in the **Type** column and its corresponding type ID in the **Id** column.

3. Enter the partition type command and select a partition using its minor number:

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

4. Optionally, list the available hexadecimal codes:

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

5. Set the partition type:

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

6. Write your changes and exit the **fdisk** shell:

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

7. Verify your changes:

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

1.4. REMOVING A PARTITION

As a system administrator, you can remove a disk partition that is no longer used to free up disk space.

WARNING

Removing a partition deletes all data stored on the partition.

1.4.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions

The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.4.2. Removing a partition with parted

This procedure describes how to remove a disk partition using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to remove a partition: for example, /**dev**/**sda**.

2. View the current partition table to determine the minor number of the partition to remove:

```
(parted) print
```

3. Remove the partition:

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

   - Replace *minor-number* with the minor number of the partition you want to remove: for example, **3**.

   The changes start taking place as soon as you enter this command, so review it before executing it.

4. Confirm that the partition is removed from the partition table:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Verify that the kernel knows the partition is removed:

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

7. Remove the partition from the **/etc/fstab** file if it is present. Find the line that declares the removed partition, and remove it from the file.

8. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

```
# systemctl daemon-reload
```

9. If you have deleted a swap partition or removed pieces of LVM, remove all references to the partition from the kernel command line in the **/etc/default/grub** file and regenerate GRUB configuration:

   - On a BIOS-based system:

```
# grub2-mkconfig --output=/etc/grub2.cfg
```

   - On a UEFI-based system:

```
# grub2-mkconfig --output=/etc/grub2-efi.cfg
```

10. To register the changes in the early boot system, rebuild the **initramfs** file system:

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

Additional resources

- The **parted(8)** man page

1.5. RESIZING A PARTITION

As a system administrator, you can extend a partition to utilize unused disk space, or shrink a partition to use its capacity for different purposes.

1.5.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.5.2. Resizing a partition with parted

This procedure resizes a disk partition using the **parted** utility.

Prerequisites

- If you want to shrink a partition, back up the data that are stored on it.

WARNING

Shrinking a partition might result in data loss on the partition.

- If you want to resize a partition to be larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT). For details on how to format the disk, see Section 1.2, "Creating a partition table on a disk".

Procedure

1. If you want to shrink the partition, shrink the file system on it first so that it is not larger than the resized partition. Note that XFS does not support shrinking.
2. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to resize a partition: for example, /**dev**/**sda**.

3. View the current partition table:

```
(parted) print
```

   From the partition table, determine:

   - The minor number of the partition
   - The location of the existing partition and its new ending point after resizing

4. Resize the partition:

```
(parted) resizepart minor-number new-end
```

   - Replace *minor-number* with the minor number of the partition that you are resizing: for example, **3**.
   - Replace *new-end* with the size that determines the new ending point of the resized partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

Example 1.4. Extending a partition

For example, to extend a partition located at the beginning of the disk to be 2GiB in size, use:

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

The changes start taking place as soon as you enter this command, so review it before executing it.

5. View the partition table to confirm that the resized partition is in the partition table with the correct size:

```
(parted) print
```

6. Exit the **parted** shell:

```
(parted) quit
```

7. Verify that the kernel recognizes the new partition:

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

8. If you extended the partition, extend the file system on it as well. See (reference) for details.

Additional resources

- The **parted(8)** man page.

CHAPTER 2. OVERVIEW OF PERSISTENT NAMING ATTRIBUTES

As a system administrator, you need to refer to storage volumes using persistent naming attributes to build storage setups that are reliable over multiple system boots.

2.1. DISADVANTAGES OF NON-PERSISTENT NAMING ATTRIBUTES

Red Hat Enterprise Linux provides a number of ways to identify storage devices. It is important to use the correct option to identify each device when used in order to avoid inadvertently accessing the wrong device, particularly when installing to or reformatting drives.

Traditionally, non-persistent names in the form of /**dev**/**sd***(major number)(minor number)* are used on Linux to refer to storage devices. The major and minor number range and associated **sd** names are allocated for each device when it is detected. This means that the association between the major and minor number range and associated **sd** names can change if the order of device detection changes.

Such a change in the ordering might occur in the following situations:

- The parallelization of the system boot process detects storage devices in a different order with each system boot.
- A disk fails to power up or respond to the SCSI controller. This results in it not being detected by the normal device probe. The disk is not accessible to the system and subsequent devices will have their major and minor number range, including the associated **sd** names shifted down. For example, if a disk normally referred to as **sdb** is not detected, a disk that is normally referred to as **sdc** would instead appear as **sdb**.
- A SCSI controller (host bus adapter, or HBA) fails to initialize, causing all disks connected to that HBA to not be detected. Any disks connected to subsequently probed HBAs are assigned different major and minor number ranges, and different associated **sd** names.
- The order of driver initialization changes if different types of HBAs are present in the system. This causes the disks connected to those HBAs to be detected in a different order. This might also occur if HBAs are moved to different PCI slots on the system.
- Disks connected to the system with Fibre Channel, iSCSI, or FCoE adapters might be inaccessible at the time the storage devices are probed, due to a storage array or intervening switch being powered off, for example. This might occur when a system reboots after a power failure, if the storage array takes longer to come online than the system take to boot. Although some Fibre Channel drivers support a mechanism to specify a persistent SCSI target ID to WWPN mapping, this does not cause the major and minor number ranges, and the associated **sd** names to be reserved; it only provides consistent SCSI target ID numbers.

These reasons make it undesirable to use the major and minor number range or the associated **sd** names when referring to devices, such as in the **/etc/fstab** file. There is the possibility that the wrong device will be mounted and data corruption might result.

Occasionally, however, it is still necessary to refer to the **sd** names even when another mechanism is used, such as when errors are reported by a device. This is because the Linux kernel uses **sd** names (and also SCSI host/channel/target/LUN tuples) in kernel messages regarding the device.

2.2. FILE SYSTEM AND DEVICE IDENTIFIERS

This sections explains the difference between persistent attributes identifying file systems and block devices.

File system identifiers

File system identifiers are tied to a particular file system created on a block device. The identifier is also stored as part of the file system. If you copy the file system to a different device, it still carries the same file system identifier. On the other hand, if you rewrite the device, such as by formatting it with the **mkfs** utility, the device loses the attribute.

File system identifiers include:

- Unique identifier (UUID)
- Label

Device identifiers

Device identifiers are tied to a block device: for example, a disk or a partition. If you rewrite the device, such as by formatting it with the **mkfs** utility, the device keeps the attribute, because it is not stored in the file system.

Device identifiers include:

- World Wide Identifier (WWID)
- Partition UUID
- Serial number

Recommendations

- Some file systems, such as logical volumes, span multiple devices. Red Hat recommends accessing these file systems using file system identifiers rather than device identifiers.

2.3. DEVICE NAMES MANAGED BY THE UDEV MECHANISM IN /DEV/DISK/

This section lists different kinds of persistent naming attributes that the **udev** service provides in the /**dev**/**disk**/ directory.

The **udev** mechanism is used for all types of devices in Linux, not just for storage devices. In the case of storage devices, Red Hat Enterprise Linux contains **udev** rules that create symbolic links in the /**dev**/**disk**/ directory. This enables you to refer to storage devices by:

- Their content
- A unique identifier
- Their serial number.

Although **udev** naming attributes are persistent, in that they do not change on their own across system reboots, some are also configurable.

2.3.1. File system identifiers

The UUID attribute in /dev/disk/by-uuid/

Entries in this directory provide a symbolic name that refers to the storage device by a **unique identifier** (UUID) in the content (that is, the data) stored on the device. For example:

```
/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can use the UUID to refer to the device in the **/etc/fstab** file using the following syntax:

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can configure the UUID attribute when creating a file system, and you can also change it later on.

The Label attribute in /dev/disk/by-label/

Entries in this directory provide a symbolic name that refers to the storage device by a **label** in the content (that is, the data) stored on the device.

For example:

```
/dev/disk/by-label/Boot
```

You can use the label to refer to the device in the **/etc/fstab** file using the following syntax:

```
LABEL=Boot
```

You can configure the Label attribute when creating a file system, and you can also change it later on.

2.3.2. Device identifiers

The WWID attribute in /dev/disk/by-id/

The World Wide Identifier (WWID) is a persistent, **system-independent identifier** that the SCSI Standard requires from all SCSI devices. The WWID identifier is guaranteed to be unique for every storage device, and independent of the path that is used to access the device. The identifier is a property of the device but is not stored in the content (that is, the data) on the devices.

This identifier can be obtained by issuing a SCSI Inquiry to retrieve the Device Identification Vital Product Data (page **0x83**) or Unit Serial Number (page **0x80**).

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current **/dev/sd** name on that system. Applications can use the **/dev/disk/by-id/** name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems.

Example 2.1. WWID mappings

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	**/dev/sda**	A device with a page **0x83** identifier
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	**/dev/sdb**	A device with a page **0x80** identifier

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3	**/dev/sdc3**	A disk partition

In addition to these persistent names provided by the system, you can also use **udev** rules to implement persistent names of your own, mapped to the WWID of the storage.

The Partition UUID attribute in /dev/disk/by-partuuid

The Partition UUID (PARTUUID) attribute identifies partitions as defined by GPT partition table.

Example 2.2. Partition UUID mappings

PARTUUID symlink	Non-persistent device
/dev/disk/by-partuuid/4cd1448a-01	**/dev/sda1**
/dev/disk/by-partuuid/4cd1448a-02	**/dev/sda2**
/dev/disk/by-partuuid/4cd1448a-03	**/dev/sda3**

The Path attribute in /dev/disk/by-path/

This attribute provides a symbolic name that refers to the storage device by the **hardware path** used to access the device.

WARNING

The Path attribute is unreliable, and Red Hat does not recommend using it.

2.4. THE WORLD WIDE IDENTIFIER WITH DM MULTIPATH

This section describes the mapping between the World Wide Identifier (WWID) and non-persistent device names in a Device Mapper Multipath configuration.

If there are multiple paths from a system to a device, DM Multipath uses the WWID to detect this. DM Multipath then presents a single "pseudo-device" in the **/dev/mapper/wwid** directory, such as **/dev/mapper/3600508b400105df70000e00000ac0000**.

The command **multipath -l** shows the mapping to the non-persistent identifiers:

- ***Host*:*Channel*:*Target*:*LUN***
- /**dev**/**sd** name
- ***major*:*minor*** number

Example 2.3. WWID mappings in a multipath configuration

An example output of the **multipath -l** command:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

DM Multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding /**dev**/**sd** name on the system. These names are persistent across path changes, and they are consistent when accessing the device from different systems.

When the **user_friendly_names** feature of DM Multipath is used, the WWID is mapped to a name of the form /**dev**/**mapper**/**mpath*N***. By default, this mapping is maintained in the file /**etc**/**multipath**/**bindings**. These **mpath*N*** names are persistent as long as that file is maintained.

IMPORTANT

If you use **user_friendly_names**, then additional steps are required to obtain consistent names in a cluster.

2.5. LIMITATIONS OF THE UDEV DEVICE NAMING CONVENTION

The following are some limitations of the **udev** naming convention:

- It is possible that the device might not be accessible at the time the query is performed because the **udev** mechanism might rely on the ability to query the storage device when the **udev** rules are processed for a **udev** event. This is more likely to occur with Fibre Channel, iSCSI or FCoE storage devices when the device is not located in the server chassis.
- The kernel might send **udev** events at any time, causing the rules to be processed and possibly causing the /**dev**/**disk**/**by-***/ links to be removed if the device is not accessible.
- There might be a delay between when the **udev** event is generated and when it is processed, such as when a large number of devices are detected and the user-space **udevd** service takes some amount of time to process the rules for each one. This might cause a delay between when the kernel detects the device and when the /**dev**/**disk**/**by-***/ names are available.
- External programs such as **blkid** invoked by the rules might open the device for a brief period of time, making the device inaccessible for other uses.

2.6. LISTING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to find out the persistent naming attributes of non-persistent storage devices.

Procedure

- To list the UUID and Label attributes, use the **lsblk** utility:

```
$ lsblk --fs storage-device
```

For example:

Example 2.4. Viewing the UUID and Label of a file system

```
$ lsblk --fs /dev/sda1

NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs    Boot  afa5d5e3-9050-48c3-acc1-bb30095f3dc4 /boot
```

- To list the PARTUUID attribute, use the **lsblk** utility with the **--output +PARTUUID** option:

```
$ lsblk --output +PARTUUID
```

For example:

Example 2.5. Viewing the PARTUUID attribute of a partition

```
$ lsblk --output +PARTUUID /dev/sda1

NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT PARTUUID
sda1   8:1    0  512M  0 part /boot      4cd1448a-01
```

- To list the WWID attribute, examine the targets of symbolic links in the **/dev/disk/by-id/** directory. For example:

Example 2.6. Viewing the WWID of all storage devices on the system

```
$ file /dev/disk/by-id/*

/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001:
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1:
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2:
symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root:                                symbolic link to
../../dm-0
/dev/disk/by-id/dm-name-rhel_rhel8-swap:                                 symbolic link
to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g:
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
```

```
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrlRLhzfMyOKd:
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-wu4QQm:
symbolic link to ../../sda2
```

2.7. MODIFYING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to change the UUID or Label persistent naming attribute of a file system.

NOTE

Changing **udev** attributes happens in the background and might take a long time. The **udevadm settle** command waits until the change is fully registered, which ensures that your next command will be able to utilize the new attribute correctly.

In the following commands:

- Replace *new-uuid* with the UUID you want to set; for example, **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**. You can generate a UUID using the **uuidgen** command.
- Replace *new-label* with a label; for example, **backup_data**.

Prerequisites

- If you are modifying the attributes of an XFS file system, unmount it first.

Procedure

- To change the UUID or Label attributes of an **XFS** file system, use the **xfs_admin** utility:

```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of an **ext4**, **ext3**, or **ext2** file system, use the **tune2fs** utility:

```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of a swap volume, use the **swaplabel** utility:

```
# swaplabel --uuid new-uuid --label new-label swap-device
# udevadm settle
```

CHAPTER 3. USING NVDIMM PERSISTENT MEMORY STORAGE

As a system administrator, you can enable and manage various types of storage on Non-Volatile Dual In-line Memory Modules (NVDIMM) devices connected to your system.

For installing Red Hat Enterprise Linux 8 on NVDIMM storage, see Installing to an NVDIMM device instead.

3.1. THE NVDIMM PERSISTENT MEMORY TECHNOLOGY

NVDIMM persistent memory, also called storage class memory or **pmem**, is a combination of memory and storage.

NVDIMM combines the durability of storage with the low access latency and the high bandwidth of dynamic RAM (DRAM):

- NVDIMM storage is byte-addressable, so it can be accessed by using the CPU load and store instructions. In addition to the **read()** and **write()** system calls, which are required for accessing traditional block-based storage, NVDIMM also supports direct load and store programming model.
- The performance characteristics of NVDIMM are similar to DRAM with very low access latency, typically in the tens to hundreds of nanoseconds.
- Data stored on NVDIMM are preserved when the power is off, like with storage.
- The direct access (DAX) technology enables applications to memory map storage directly, without going through the system page cache. This frees up DRAM for other purposes.

NVDIMM is beneficial in use cases such as:

Databases

The reduced storage access latency on NVDIMM can dramatically improve database performance.

Rapid restart

Rapid restart is also called the warm cache effect. For example, a file server has none of the file contents in memory after starting. As clients connect and read or write data, that data is cached in the page cache. Eventually, the cache contains mostly hot data. After a reboot, the system must start the process again on traditional storage.
NVDIMM enables an application to keep the warm cache across reboots if the application is designed properly. In this example, there would be no page cache involved: the application would cache data directly in the persistent memory.

Fast write-cache

File servers often do not acknowledge a client's write request until the data is on durable media. Using NVDIMM as a fast write cache enables a file server to acknowledge the write request quickly thanks to the low latency.

3.2. NVDIMM INTERLEAVING AND REGIONS

NVDIMM devices support grouping into interleaved regions.

NVDIMM devices can be grouped into interleave sets in the same way as regular DRAM. An interleave set is similar to a RAID 0 level (stripe) configuration across multiple DIMMs. An Interleave set is also called a *region*.

Interleaving has the following advantages:

- NVDIMM devices benefit from increased performance when they are configured into interleave sets.
- Interleaving can combine multiple smaller NVDIMM devices into a larger logical device.

NVDIMM interleave sets are configured in the system BIOS or UEFI firmware.

Red Hat Enterprise Linux creates one region device for each interleave set.

3.3. NVDIMM NAMESPACES

NVDIMM regions are divided into one or more namespaces. Namespaces enable you to access the device using different methods, based on the type of the namespace.

Some NVDIMM devices do not support multiple namespaces on a region:

- If your NVDIMM device supports labels, you can subdivide the region into namespaces.
- If your NVDIMM device does not support labels, the region can only contain a single namespace. In that case, Red Hat Enterprise Linux creates a default namespace that covers the entire region.

3.4. NVDIMM ACCESS MODES

You can configure NVDIMM namespaces to use either of the following modes:

sector

Presents the storage as a fast block device. This mode is useful for legacy applications that have not been modified to use NVDIMM storage, or for applications that make use of the full I/O stack, including Device Mapper.
A **sector** device can be used in the same way as any other block device on the system. You can create partitions or file systems on it, configure it as part of a software RAID set, or use it as the cache device for **dm-cache**.

Devices in this mode are available at **/dev/pmem*N*s**. See the **blockdev** value listed after creating the namespace.

devdax, or device direct access (DAX)

Enables NVDIMM devices to support direct access programming as described in the Storage Networking Industry Association (SNIA) Non-Volatile Memory (NVM) Programming Model specification. In this mode, I/O bypasses the storage stack of the kernel. Therefore, no Device Mapper drivers can be used.
Device DAX provides raw access to NVDIMM storage by using a DAX character device node. Data on a **devdax** device can be made durable using CPU cache flushing and fencing instructions. Certain databases and virtual machine hypervisors might benefit from this mode. File systems cannot be created on **devdax** devices.

Devices in this mode are available at **/dev/dax*N.M*.** See the **chardev** value listed after creating the namespace.

fsdax, or file system direct access (DAX)

Enables NVDIMM devices to support direct access programming as described in the Storage Networking Industry Association (SNIA) Non-Volatile Memory (NVM) Programming Model specification. In this mode, I/O bypasses the storage stack of the kernel, and many Device Mapper drivers therefore cannot be used.
You can create file systems on file system DAX devices.

Devices in this mode are available at **/dev/pmem*N***. See the **blockdev** value listed after creating the namespace.

IMPORTANT

The file system DAX technology is provided only as a Technology Preview, and is not supported by Red Hat.

raw

Presents a memory disk that does not support DAX. In this mode, namespaces have several limitations and should not be used.
Devices in this mode are available at **/dev/pmem*N***. See the **blockdev** value listed after creating the namespace.

3.5. CREATING A SECTOR NAMESPACE ON AN NVDIMM TO ACT AS A BLOCK DEVICE

You can configure an NVDIMM device in sector mode, which is also called *legacy mode*, to support traditional, block-based storage.

You can either:

- reconfigure an existing namespace to sector mode, or
- create a new sector namespace if there is available space.

3.5.1. Prerequisites

- An NVDIMM device is attached to your system.

3.5.2. Installing ndctl

This procedure installs the **ndctl** utility, which is used to configure and monitor NVDIMM devices.

Procedure

- To install the **ndctl** utility, use the following command:

```
# yum install ndctl
```

3.5.3. Reconfiguring an existing NVDIMM namespace to sector mode

This procedure reconfigures an NVDIMM namespace to sector mode for use as a fast block device.

WARNING

Reconfiguring a namespace deletes all data previously stored on the namespace.

Prerequisites

- The **ndctl** utility is installed. See Section 3.5.2, “Installing ndctl”.

Procedure

1. Reconfigure the selected namespace to sector mode:

```
# ndctl create-namespace \
      --force \
      --reconfig=namespace-ID \
      --mode=sector
```

Example 3.1. Reconfiguring namespace1.0 in sector mode

To reconfigure the **namespace1.0** namespace to use **sector** mode:

```
# ndctl create-namespace \
      --force \
      --reconfig=namespace1.0 \
      --mode=sector

{
  "dev":"namespace1.0",
  "mode":"sector",
  "size":"11.99 GiB (12.87 GB)",
  "uuid":"5805480e-90e6-407e-96a4-23e1cde2ed78",
  "raw_uuid":"879d9e9f-fd43-4ed5-b64f-3bcd0781391a",
  "sector_size":4096,
  "blockdev":"pmem1s",
  "numa_node":1
}
```

2. The reconfigured namespace is now available under the **/dev** directory as **/dev/pmem*N*s**.

Additional resources

- The **ndctl-create-namespace(1)** man page

3.5.4. Creating a new NVDIMM namespace in sector mode

This procedure creates a new sector namespace on an NVDIMM device, enabling you to use it as a traditional block device.

Prerequisites

- The **ndctl** utility is installed. See Section 3.5.2, "Installing ndctl".
- The NVDIMM device supports labels.

Procedure

1. List the **pmem** regions on your system that have available space. In the following example, space is available in the **region5** and **region4** regions:

```
# ndctl list --regions

[
  {
    "dev":"region5",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-7337419320239190016
  },
  {
    "dev":"region4",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-137289417188962304
  }
]
```

2. On any of the available regions, allocate one or more namespaces:

```
# ndctl create-namespace \
      --mode=sector \
      --region=regionN \
      --size=namespace-size
```

Example 3.2. Creating a namespace on a region

The following command creates a 36-GiB sector namespace on **region4**:

```
# ndctl create-namespace \
      --mode=sector \
      --region=region4 \
      --size=36G
```

3. The new namespace is now available under the **/dev** directory as **/dev/pmem*N*s**.

Additional resources

- The **ndctl-create-namespace(1)** man page

3.6. CREATING A DEVICE DAX NAMESPACE ON AN NVDIMM

You can configure an NVDIMM device in device DAX mode to support character storage with direct access capabilities.

You can either:

- reconfigure an existing namespace to device DAX mode, or
- create a new device DAX namespace if there is available space.

3.6.1. Prerequisites

- An NVDIMM device is attached to your system.

3.6.2. NVDIMM in device direct access mode

Device direct access (device DAX, **devdax**) provides a means for applications to directly access storage, without the involvement of a file system. The benefit of device DAX is that it provides a guaranteed fault granularity, which can be configured using the **--align** option of the **ndctl** utility

For the Intel 64 and AMD64 architecture, the following fault granularities are supported:

- 4 KiB
- 2 MiB
- 1 GiB

Device DAX nodes support only the following system calls:

- **open()**
- **close()**
- **mmap()**

The **read()** and **write()** variants are not supported because the device DAX use case is tied to persistent memory programming.

3.6.3. Installing ndctl

This procedure installs the **ndctl** utility, which is used to configure and monitor NVDIMM devices.

Procedure

- To install the **ndctl** utility, use the following command:

```
# yum install ndctl
```

3.6.4. Reconfiguring an existing NVDIMM namespace to device DAX mode

This procedure reconfigures a namespace on an NVDIMM device to device DAX mode, and enables you to store data on the namespace.

WARNING

Reconfiguring a namespace deletes all data previously stored on the namespace.

Prerequisites

- The **ndctl** utility is installed. See Section 3.6.3, “Installing ndctl”.

Procedure

1. List all namespaces on your system:

```
# 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. Reconfigure any namespace:

```
# ndctl create-namespace \
      --force \
      --mode=devdax \
      --reconfig=namespace-ID
```

Example 3.3. Reconfiguring a namespace as device DAX

The following command reconfigures **namespace0.0** for data storage that supports DAX. It is aligned to a 2-MiB fault granularity to ensure that the operating system faults in 2-MiB pages at a time:

```
# ndctl create-namespace \
      --force \
      --mode=devdax \
      --align=2M \
      --reconfig=namespace0.0
```

3. The namespace is now available at the /**dev**/**dax*N.M*** path.

Additional resources

- The **ndctl-create-namespace(1)** man page

3.6.5. Creating a new NVDIMM namespace in device DAX mode

This procedure creates a new device DAX namespace on an NVDIMM device, enabling you to store data on the namespace.

Prerequisites

- The **ndctl** utility is installed. See Section 3.6.3, “Installing ndctl”.
- The NVDIMM device supports labels.

Procedure

1. List the **pmem** regions on your system that have available space. In the following example, space is available in the **region5** and **region4** regions:

```
# ndctl list --regions

[
  {
    "dev":"region5",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-7337419320239190016
  },
  {
    "dev":"region4",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-137289417188962304
  }
]
```

2. On any of the available regions, allocate one or more namespaces:

```
# ndctl create-namespace \
      --mode=devdax \
      --region=regionN \
      --size=namespace-size
```

Example 3.4. Creating a namespace on a region

The following command creates a 36-GiB device DAX namespace on **region4**. It is aligned to a 2-MiB fault granularity to ensure that the operating system faults in 2-MiB pages at a time:

```
# ndctl create-namespace \
      --mode=devdax \
      --region=region4 \
```

```
        --align=2M \
        --size=36G

{
  "dev":"namespace1.2",
  "mode":"devdax",
  "map":"dev",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"5ae01b9c-1ebf-4fb6-bc0c-6085f73d31ee",
  "raw_uuid":"4c8be2b0-0842-4bcb-8a26-4bbd3b44add2",
  "daxregion":{
    "id":1,
    "size":"35.44 GiB (38.05 GB)",
    "align":2097152,
    "devices":[
      {
        "chardev":"dax1.2",
        "size":"35.44 GiB (38.05 GB)"
      }
    ]
  },
  "numa_node":1
}
```

3. The namespace is now available at the /**dev**/**dax*N.M*** path.

Additional resources

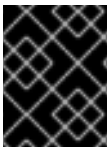
- The **ndctl-create-namespace(1)** man page

3.7. CREATING A FILE SYSTEM DAX NAMESPACE ON AN NVDIMM

You can configure an NVDIMM device in file system DAX mode to support a file system with direct access capabilities.

You can either:

- reconfigure an existing namespace to file system DAX mode, or
- create a new file system DAX namespace if there is available space.



IMPORTANT

The file system DAX technology is provided only as a Technology Preview, and is not supported by Red Hat.

3.7.1. Prerequisites

- An NVDIMM device is attached to your system.

3.7.2. NVDIMM in file system direct access mode

When an NVDIMM device is configured in file system direct access (file system DAX, **fsdax**) mode, a file system can be created on top of it.

Any application that performs an **mmap()** operation on a file on this file system gets direct access to its storage. This enables the direct access programming model on NVDIMM. The file system must be mounted with the **-o dax** option in order for direct mapping to happen.

Per-page metadata allocation
This mode requires allocating per-page metadata in the system DRAM or on the NVDIMM device itself. The overhead of this data structure is 64 bytes per each 4-KiB page:

- On small devices, the amount of overhead is small enough to fit in DRAM with no problems. For example, a 16-GiB namespace only requires 256 MiB for page structures. Because NVDIMM devices are usually small and expensive, storing the page tracking data structures in DRAM is preferable.
- On NVDIMM devices that are be terabytes in size or larger, the amount of memory required to store the page tracking data structures might exceed the amount of DRAM in the system. One TiB of NVDIMM requires 16 GiB just for page structures. As a result, storing the data structures on the NVDIMM itself is preferable in such cases.

You can configure where per-page metadata are stored using the **--map** option when configuring a namespace:

- To allocate in the system RAM, use **--map=mem**.
- To allocate on the NVDIMM, use **--map=dev**.

Partitions and file systems on fsdax
When creating partitions on an **fsdax** device, partitions must be aligned on page boundaries. On the Intel 64 and AMD64 architecture, at least 4 KiB alignment is required for the start and end of the partition. 2 MiB is the preferred alignment.

On Red Hat Enterprise Linux 8, both the XFS and ext4 file system can be created on NVDIMM as a Technology Preview.

3.7.3. Installing ndctl

This procedure installs the **ndctl** utility, which is used to configure and monitor NVDIMM devices.

Procedure

- To install the **ndctl** utility, use the following command:

```
# yum install ndctl
```

3.7.4. Reconfiguring an existing NVDIMM namespace to file system DAX mode

This procedure reconfigures a namespace on an NVDIMM device to file system DAX mode, and enables you to store files on the namespace.

WARNING

Reconfiguring a namespace deletes all data previously stored on the namespace.

Prerequisites

- The **ndctl** utility is installed. See Section 3.7.3, “Installing ndctl”.

Procedure

1. List all namespaces on your system:

```
# 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. Reconfigure any namespace:

```
# ndctl create-namespace \
        --force \
        --mode=fsdax \
        --reconfig=namespace-ID
```

Example 3.5. Reconfiguring a namespace as file system DAX

To use **namespace0.0** for a file system that supports DAX, use the following command:

```
# ndctl create-namespace \
        --force \
        --mode=fsdax \
        --reconfig=namespace0.0

{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "size":"32.00 GiB (34.36 GB)",
```

```
  "uuid":"ab91cc8f-4c3e-482e-a86f-78d177ac655d",
  "blockdev":"pmem0",
  "numa_node":0
}
```

3. The namespace is now available at the **/dev/pmem*N*** path.

Additional resources

- The **ndctl-create-namespace(1)** man page

3.7.5. Creating a new NVDIMM namespace in file system DAX mode

This procedure creates a new file system DAX namespace on an NVDIMM device, enabling you to store files on the namespace.

Prerequisites

- The **ndctl** utility is installed. See Section 3.7.3, “Installing ndctl”.
- The NVDIMM device supports labels.

Procedure

1. List the **pmem** regions on your system that have available space. In the following example, space is available in the **region5** and **region4** regions:

```
# ndctl list --regions

[
  {
    "dev":"region5",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-7337419320239190016
  },
  {
    "dev":"region4",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-137289417188962304
  }
]
```

2. On any of the available regions, allocate one or more namespaces:

```
# ndctl create-namespace \
      --mode=fsdax \
      --region=regionN \
      --size=namespace-size
```

Example 3.6. Creating a namespace on a region

The following command creates a 36-GiB file system DAX namespace on **region4**:

```
# ndctl create-namespace \
        --mode=fsdax \
        --region=region4 \
        --size=36G

{
  "dev":"namespace4.0",
  "mode":"fsdax",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"9c5330b5-dc90-4f7a-bccd-5b558fa881fe",
  "blockdev":"pmem4",
  "numa_node":0
}
```

3. The namespace is now available at the **/dev/pmem*N*** path.

Additional resources

- The **ndctl-create-namespace(1)** man page

3.7.6. Creating a file system on a file system DAX device

This procedure creates a file system on a file system DAX device and mounts the file system.

Procedure

1. Optionally, create a partition on the file system DAX device. See Section 1.3, “Creating a partition”.
   By default, the **parted** tool aligns partitions on 1 MiB boundaries. For the first partition, specify 2 MiB as the start of the partition. If the size of the partition is a multiple of 2 MiB, all other partitions are also aligned.

2. Create an XFS or ext4 file system on the partition or the NVDIMM device.
   For XFS, disable shared copy-on-write data extents when creating the file system:

   ```
   # mkfs.xfs -m reflink=0 fsdax-partition-or-device
   ```

3. Mount the file system with the **-o fsdax** mount option:

   ```
   # mount -o fsdax fsdax-partition-or-device mount-point
   ```

4. Applications can now use persistent memory and create files in the *mount-point* directory, open the files, and use the **mmap** operation to map the files for direct access.

Additional resources

- The **mkfs.xfs(8)** man page

3.8. TROUBLESHOOTING NVDIMM PERSISTENT MEMORY

You can detect and fix different kinds of errors on NVDIMM devices.

3.8.1. Prerequisites

- An NVDIMM device is connected to your system and configured.

3.8.2. Installing ndctl

This procedure installs the **ndctl** utility, which is used to configure and monitor NVDIMM devices.

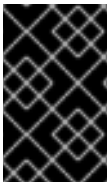
Procedure

- To install the **ndctl** utility, use the following command:

```
# yum install ndctl
```

3.8.3. Monitoring NVDIMM health using S.M.A.R.T.

Some NVDIMM devices support Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) interfaces for retrieving health information.



IMPORTANT

Monitor NVDIMM health regularly to prevent data loss. If S.M.A.R.T. reports problems with the health status of an NVDIMM device, replace it as described in Section 3.8.4, “Detecting and replacing a broken NVDIMM device”.

Prerequisites

- On some systems, the **acpi_ipmi** driver must be loaded to retrieve health information using the following command:

```
# modprobe acpi_ipmi
```

Procedure

- To access the health information, use the following command:

```
# ndctl list --dimms --health

...
   {
    "dev":"nmem0",
    "id":"802c-01-1513-b3009166",
    "handle":1,
    "phys_id":22,
    "health":
    {
     "health_state":"ok",
     "temperature_celsius":25.000000,
     "spares_percentage":99,
     "alarm_temperature":false,
     "alarm_spares":false,
     "temperature_threshold":50.000000,
     "spares_threshold":20,
     "life_used_percentage":1,
```

```
        "shutdown_state":"clean"
      }
     }
...
```

Additional resources

- The **ndctl-list(1)** man page

3.8.4. Detecting and replacing a broken NVDIMM device

If you find error messages related to NVDIMM reported in your system log or by S.M.A.R.T., it might mean an NVDIMM device is failing. In that case, it is necessary to:

1. Detect which NVDIMM device is failing
2. Back up data stored on it
3. Physically replace the device

Procedure

1. To detect the broken device, use the following command:

```
# ndctl list --dimms --regions --health --media-errors --human
```

The **badblocks** field shows which NVDIMM is broken. Note its name in the **dev** field.

Example 3.7. Health status of NVDIMM devices

In the following example, the NVDIMM named **nmem0** is broken:

```
# ndctl list --dimms --regions --health --media-errors --human

...
  "regions":[
    {
      "dev":"region0",
      "size":"250.00 GiB (268.44 GB)",
      "available_size":0,
      "type":"pmem",
      "numa_node":0,
      "iset_id":"0xXXXXXXXXXXXXXXXX",
      "mappings":[
        {
          "dimm":"nmem1",
          "offset":"0x10000000",
          "length":"0x1f40000000",
          "position":1
        },
        {
          "dimm":"nmem0",
          "offset":"0x10000000",
          "length":"0x1f40000000",
          "position":0
        }
```

```
      ],
      "badblock_count":1,
      "badblocks":[
        {
          "offset":65536,
          "length":1,
          "dimms":[
            "nmem0"
          ]
        }
      ],
      "persistence_domain":"memory_controller"
    }
  ]
}
```

2. Use the following command to find the **phys_id** attribute of the broken NVDIMM:

```
# ndctl list --dimms --human
```

From the previous example, you know that **nmem0** is the broken NVDIMM. Therefore, find the **phys_id** attribute of **nmem0**.

Example 3.8. The phys_id attributes of NVDIMMs

In the following example, the **phys_id** is **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. Use the following command to find the memory slot of the broken NVDIMM:

```
# dmidecode
```

In the output, find the entry where the **Handle** identifier matches the **phys_id** attribute of the broken NVDIMM. The **Locator** field lists the memory slot used by the broken NVDIMM.

Example 3.9. NVDIMM Memory Slot Listing

In the following example, the **nmem0** device matches the **0x0010** identifier and uses the **DIMM-XXX-YYYY** memory slot:

```
# 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. Back up all data in the namespaces on the NVDIMM. If you do not back up the data before replacing the NVDIMM, the data will be lost when you remove the NVDIMM from your system.

WARNING

In some cases, such as when the NVDIMM is completely broken, the backup might fail.

To prevent this, regularly monitor your NVDIMM devices using S.M.A.R.T. as described in Section 3.8.3, "Monitoring NVDIMM health using S.M.A.R.T." and replace failing NVDIMMs before they break.

Use the following command to list the namespaces on the NVDIMM:

```
# ndctl list --namespaces --dimm=DIMM-ID-number
```

Example 3.10. NVDIMM namespaces listing

In the following example, the **nmem0** device contains the **namespace0.0** and **namespace0.2** namespaces, which you need to back up:

```
# 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
  }
]
```


5. Replace the broken NVDIMM physically.

Additional resources

- The **ndctl-list(1)** man page
- The **dmidecode(8)** man page

CHAPTER 4. DISCARDING UNUSED BLOCKS

You can perform or schedule discard operations on block devices that support them.

4.1. BLOCK DISCARD OPERATIONS

Block discard operations discard blocks that are no longer in use by a mounted file system. They are useful on:

- Solid-state drives (SSDs)
- Thinly-provisioned storage

Requirements

The block device underlying the file system must support physical discard operations.

Physical discard operations are supported if the value in the **/sys/block/*device*/queue/discard_max_bytes** file is not zero.

4.2. TYPES OF BLOCK DISCARD OPERATIONS

You can run discard operations using different methods:

Batch discard

Are run explicitly by the user. They discard all unused blocks in the selected file systems.

Online discard

Are specified at mount time. They run in real time without user intervention. Online discard operations discard only the blocks that are transitioning from used to free.

Periodic discard

Are batch operations that are run regularly by a **systemd** service.

All types are supported by the XFS and ext4 file systems and by VDO.

Recommendations

Red Hat recommends that you use batch or periodic discard.

Use online discard only if:

- the system's workload is such that batch discard is not feasible, or
- online discard operations are necessary to maintain performance.

4.3. PERFORMING BATCH BLOCK DISCARD

This procedure performs a batch block discard operation to discard unused blocks on a mounted file system.

Prerequisites

- The file system is mounted.
- The block device underlying the file system supports physical discard operations.

Procedure

- Use the **fstrim** utility:
  - To perform discard only on a selected file system, use:

```
# fstrim mount-point
```

  - To perform discard on all mounted file systems, use:

```
# fstrim --all
```

If you execute the **fstrim** command on:

- a device that does not support discard operations, or
- a logical device (LVM or MD) composed of multiple devices, where any one of the device does not support discard operations,

the following message displays:

```
# fstrim /mnt/non_discard

fstrim: /mnt/non_discard: the discard operation is not supported
```

Additional resources

- The **fstrim(8)** man page

4.4. ENABLING ONLINE BLOCK DISCARD

This procedure enables online block discard operations that automatically discard unused blocks on all supported file systems.

Procedure

- Enable online discard at mount time:
  - When mounting a file system manually, add the **-o discard** mount option:

```
# mount -o discard device mount-point
```

  - When mounting a file system persistently, add the **discard** option to the mount entry in the **/etc/fstab** file.

Additional resources

- The **mount(8)** man page
- The **fstab(5)** man page

4.5. ENABLING PERIODIC BLOCK DISCARD

This procedure enables a **systemd** timer that regularly discards unused blocks on all supported file systems.

Procedure

- Enable and start the **systemd** timer:

```
# systemctl enable --now fstrim.timer
```

CHAPTER 5. GETTING STARTED WITH ISCSI

Red Hat Enterprise Linux 8 uses the **targetcli** shell as a command-line interface to perform the following operations:

- Add, remove, view, and monitor iSCSI storage interconnects to utilize iSCSI hardware.
- Export local storage resources that are backed by either files, volumes, local SCSI devices, or by RAM disks to remote systems.

The **targetcli** tool has a tree-based layout including built-in tab completion, auto-complete support, and inline documentation.

5.1. ADDING AN ISCSI TARGET

As a system administrator, you can add an iSCSI targets using the **targetcli** tool.

5.1.1. Installing targetcli

Install the **targetcli** tool to add, monitor, and remove iSCSI storage interconnects.

Procedure

1. Install **targetcli**:

```
# yum install targetcli
```

2. Start the target service:

```
# systemctl start target
```

3. Configure target to start at boot time:

```
# systemctl enable target
```

4. Open port **3260** in the firewall and reload the firewall configuration:

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

# firewall-cmd --reload
Success
```

5. View the **targetcli** layout:

```
# 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]
```

Additional resources

- The **targetcli** man page.

5.1.2. Creating an iSCSI target

Creating an iSCSI target enables the iSCSI initiator of the client to access the storage devices on the server. Both targets and initiators have unique identifying names.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, “Installing targetcli”.

Procedure

1. Navigate to the iSCSI directory:

```
/> iscsi/
```

NOTE

The **cd** command is used to change directories as well as to list the path to move into.

2. Use one of the following options to create an iSCSI target:

   a. Creating an iSCSI target using a default target name:

```
/iscsi> create

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

   b. Creating an iSCSI target using a specific name:

```
/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
```

   Replace *iqn.2006-04.com.example:444* with the specific target name.

3. Verify the newly created target:

```
/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]
```

Additional resources

- The **targetcli** man page.

5.1.3. iSCSI Backstore

An iSCSI backstore enables support for different methods of storing an exported LUN's data on the local machine. Creating a storage object defines the resources that the backstore uses. An administrator can choose any of the following backstore devices that Linux-IO (LIO) supports:

- **fileio** backstore: Create a **fileio** storage object if you are using regular files on the local file system as disk images. For creating a **fileio** backstore, see Section 5.1.4, "Creating a fileio storage object".
- **block** backstore: Create a **block** storage object if you are using any local block device and logical device. For creating a **block** backstore, see Section 5.1.5, "Creating a block storage object".
- **pscsi** backstore: Create a **pscsi** storage object if your storage object supports direct pass-through of SCSI commands. For creating a **pscsi** backstore, see Section 5.1.6, "Creating a pscsi storage object"
- **ramdisk** backstore: Create a **ramdisk** storage object if you want to create a temporary RAM backed device. For creating a **ramdisk** backstore, see Section 5.1.7, "Creating a Memory Copy RAM disk storage object".

Additional resources

- The **targetcli** man page.

5.1.4. Creating a fileio storage object

fileio storage objects can support either the **write_back** or **write_thru** operations. The **write_back** operation enables the local file system cache. This improves performance but increases the risk of data loss. It is recommended to use **write_back=false** to disable the **write_back** operation in favor of the **write_thru** operation.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, "Installing targetcli".

Procedure

1. Navigate to the backstores directory:

```
/> backstores/
```

2. Create a **fileio** storage object:

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

Created fileio file1 with size 209715200
```

3. Verify the created **fileio** storage object:

```
/backstores> ls
```

Additional resources

- The **targetcli** man page.

5.1.5. Creating a block storage object

The block driver allows the use of any block device that appears in the **/sys/block/** directory to be used with Linux-IO (LIO). This includes physical devices (for example, HDDs, SSDs, CDs, DVDs) and logical devices (for example, software or hardware RAID volumes, or LVM volumes).

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, “Installing targetcli”.

Procedure

1. Navigate to the backstores directory:

```
/> backstores/
```

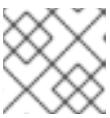
2. Create a **block** backstore:

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

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

3. Verify the created **block** storage object:

```
/backstores> ls
```



NOTE

You can also create a block backstore on a logical volume.

Additional resources

- The **targetcli** man page.

5.1.6. Creating a pscsi storage object

You can configure, as a backstore, any storage object that supports direct pass-through of SCSI commands without SCSI emulation, and with an underlying SCSI device that appears with **lsscsi** in the **/proc/scsi/scsi** (such as a SAS hard drive) . SCSI-3 and higher is supported with this subsystem.

WARNING

pscsi should only be used by advanced users. Advanced SCSI commands such as for Asymmetric Logical Unit Assignment (ALUAs) or Persistent Reservations (for example, those used by VMware ESX, and vSphere) are usually not implemented in the device firmware and can cause malfunctions or crashes. When in doubt, use **block** backstore for production setups instead.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, "Installing targetcli".

Procedure

1. Navigate to the backstores directory:

```
/> backstores/
```

2. Create a **pscsi** backstore for a physical SCSI device, a TYPE_ROM device using **/dev/sr0** in this example:

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

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

3. Verify the created **pscsi** storage object:

```
/backstores> ls
```

Additional resources

- The **targetcli** man page.

5.1.7. Creating a Memory Copy RAM disk storage object

Memory Copy RAM disks (**ramdisk**) provide RAM disks with full SCSI emulation and separate memory mappings using memory copy for initiators. This provides capability for multi-sessions and is particularly useful for fast and volatile mass storage for production purposes.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, "Installing targetcli".

Procedure

1. Navigate to the backstores directory:

```
/> backstores/
```

2. Create a 1GB RAM disk backstore:

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

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

3. Verify the created **ramdisk** storage object:

```
/backstores> ls
```

Additional resources

- The **targetcli** man page.

5.1.8. Creating an iSCSI portal

Creating an iSCSI portal adds an IP address and a port to the target that keeps the target enabled.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, “Installing targetcli”.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Section 5.1.2, “Creating an iSCSI target” .

Procedure

1. Navigate to the TPG directory:

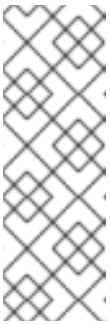
```
/iscsi> iqn.2006-04.example:444/tpg1/
```

2. Use one of the following options to create an iSCSI portal:

   a. Creating a default portal uses the default iSCSI port **3260** and allows the target to listen to all IP addresses on that port:

```
/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
```



NOTE

When an iSCSI target is created, a default portal is also created. This portal is set to listen to all IP addresses with the default port number that is: **0.0.0.0:3260**.

To remove the default portal:

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

   b. Creating a portal using a specific IP address:

```
/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
```

3. Verify the newly created portal:

```
/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]
```

Additional resources

- The **targetcli** man page.

5.1.9. Creating an iSCSI LUN

Logical unit number (LUN) is a physical device that is backed by the iSCSI backstore. Each LUN has a unique number.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, “Installing targetcli”.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Section 5.1.2, “Creating an iSCSI target” .
- Created storage objects. For more information, see Section 5.1.3, “iSCSI Backstore”.

Procedure

1. Create LUNs of already created storage objects:

```
/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.
```

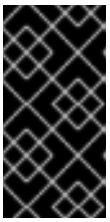
2. Verify the created LUNs:

```
/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]
```

Default LUN name starts at **0**.



IMPORTANT

By default, LUNs are created with read-write permissions. If a new LUN is added after ACLs are created, LUN automatically maps to all available ACLs and can cause a security risk. To create a LUN with read-only permissions, see Section 5.1.10, "Creating a read-only iSCSI LUN" .

Additional resources

- The **targetcli** man page.

5.1.10. Creating a read-only iSCSI LUN

By default, LUNs are created with read-write permissions. This procedure describes how to create a read-only LUN.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, "Installing targetcli".
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Section 5.1.2, "Creating an iSCSI target" .
- Created storage objects. For more information, see Section 5.1.3, "iSCSI Backstore".

Procedure

1. Set read-only permissions:

```
/> set global auto_add_mapped_luns=false

Parameter auto_add_mapped_luns is now 'false'.
```

This prevents the auto mapping of LUNs to existing ACLs allowing the manual mapping of LUNs.

2. Create the 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
```

Example:

```
/> iscsi/iqn.2006-04.example:444/tpg1/acls/2006-04.com.example.foo:888/ create
mapped_lun=1 tpg_lun_or_backstore=/backstores/block/block2 write_protect=1

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

3. Verify the created LUN:

```
/> ls

o- / ..... [...]
  o- backstores ..... [...]
  <snip>
  o- iscsi ..... [Targets: 1]
  | o- iqn.2006-04.example:444 ..... [TPGs: 1]
  |   o- tpg1 ..... [no-gen-acls, no-auth]
  |     o- acls ..... [ACLs: 2]
  |     | o- 2006-04.com.example.foo:888 .. [Mapped LUNs: 2]
  |     | | o- mapped_lun0 ..... [lun0 block/disk1 (rw)]
  |     | | o- mapped_lun1 ..... [lun1 block/disk2 (ro)]
  |     o- luns ..... [LUNs: 2]
  |     | o- lun0 ..... [block/disk1 (/dev/vdb)]
  |     | o- lun1 ..... [block/disk2 (/dev/vdc)]
  <snip>
```

The mapped_lun1 line now has (**ro**) at the end (unlike mapped_lun0's (**rw**)) stating that it is read-only.

Additional resources

- The **targetcli** man page.

5.1.11. Creating an iSCSI ACL

In **targetcli**, Access Control Lists (ACLs) are used to define access rules and each initiator has exclusive access to a LUN. Both targets and initiators have unique identifying names. You must know the unique name of the initiator to configure ACLs. The iSCSI initiators can be found in the **/etc/iscsi/initiatorname.iscsi** file.

Prerequisites

- Installed and running **targetcli**. For more information, see Section 5.1.1, "Installing targetcli".
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Section 5.1.2, "Creating an iSCSI target" .

Procedure

1. Navigate to the acls directory

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

2. Use one of the following options to create an ACL :

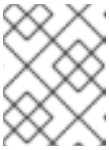
   a. Using the initiator name from **/etc/iscsi/initiatorname.iscsi** file on the initiator.

   b. Using a name that is easier to remember, see section Section 5.1.12, "Creating an iSCSI initiator" to ensure ACL matches the initiator.

```
/iscsi/iqn.20...444/tpg1/acls> create iqn.2006-04.com.example.foo:888

Created Node ACL for iqn.2006-04.com.example.foo:888
```

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



NOTE

The global setting **auto_add_mapped_luns** used in the preceding example, automatically maps LUNs to any created ACL.

You can set user-created ACLs within the TPG node on the target server:

```
/iscsi/iqn.20...scsi:444/tpg1> set attribute generate_node_acls=1
```

3. Verify the created ACL:

```
/iscsi/iqn.20...444/tpg1/acls> ls

o- acls .....[1 ACL]
  o- iqn.2006-04.com.example.foo:888 ....[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)]
```

Additional resources

- The **targetcli** man page.

5.1.12. Creating an iSCSI initiator

An iSCSI initiator forms a session to connect to the iSCSI target. For more information on iSCSI target, see Section 5.1.2, “Creating an iSCSI target” . By default, an iSCSI service is **lazily** started and the service starts after running the **iscsiadm** command. If root is not on an iSCSI device or there are no nodes marked with **node.startup = automatic** then the iSCSI service will not start until an **iscsiadm** command is executed that requires **iscsid** or the **iscsi** kernel modules to be started.

To force the **iscsid** daemon to run and iSCSI kernel modules to load:

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

Prerequisites

- Installed and running **targetcli** on a server machine. For more information, see Section 5.1.1, “Installing targetcli”.
- An iSCSI target associated with a Target Portal Groups (TPG) on a server machine. For more information, see Section 5.1.2, “Creating an iSCSI target” .
- Created iSCSI ACL. For more information, see Section 5.1.11, “Creating an iSCSI ACL” .

Procedure

1. Install **iscsi-initiator-utils** on client machine:

```
# yum install iscsi-initiator-utils
```

2. Check the initiator name:

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

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

3. If the ACL was given a custom name in Section 5.1.11, “Creating an iSCSI ACL” , modify the **/etc/iscsi/initiatorname.iscsi** file accordingly.

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

4. Discover the target and log in to the target with the displayed target IQN:

```
# 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.
```

Replace *10.64.24.179* with the target-ip-address.

You can use this procedure for any number of initiators connected to the same target if their respective initiator names are added to the ACL as described in the Section 5.1.11, “Creating an iSCSI ACL”.

5. Find the iSCSI disk name and create a file system on this iSCSI disk:

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

# mkfs.ext4 /dev/disk_name
```

Replace *disk_name* with the iSCSI disk name displayed in the **/var/log/messages** file.

6. Mount the file system:

```
# mkdir /mount/point

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

Replace */mount/point* with the mount point of the partition.

7. Edit the **/etc/fstab** file to mount the file system automatically when the system boots:

```
# vi /etc/fstab

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

Replace *disk_name* with the iSCSI disk name and */mount/point* with the mount point of the partition.

Additional resources

- The **targetcli** man page.
- The **iscsiadm** man page.

5.1.13. Setting up the Challenge-Handshake Authentication Protocol for the target

The **Challenge-Handshake Authentication Protocol (CHAP)** allows the user to protect the target with a password. The initiator must be aware of this password to be able to connect to the target.

Prerequisites

- Created iSCSI ACL. For more information, see Section 5.1.11, "Creating an iSCSI ACL" .

Procedure

1. Set attribute authentication:

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

Parameter authentication is now '1'.
```

2. Set **userid** and **password**:

```
/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'.
```

Additional resources

- The **targetcli** man page.

5.1.14. Setting up the Challenge-Handshake Authentication Protocol for the initiator

The **Challenge-Handshake Authentication Protocol (CHAP)** allows the user to protect the target with a password. The initiator must be aware of this password to be able to connect to the target.

Prerequisites

- Created iSCSI initiator. For more information, see Section 5.1.12, "Creating an iSCSI initiator" .
- Set the **CHAP** for the target. For more information, see Section 5.1.13, "Setting up the Challenge-Handshake Authentication Protocol for the target".

Procedure

1. Enable CHAP authentication in the **iscsid.conf** file:

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

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

   By default, the **node.session.auth.authmethod** is set to **None**

2. Add target **username** and **password** in the **iscsid.conf** file:

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

3. Start the **iscsid** daemon:

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

Additional resources

- The **iscsiadm** man page

5.2. MONITORING AN ISCSI SESSION

As a system administrator, you can monitor the iSCSI session using the **iscsiadm** utility.

5.2.1. Monitoring an iSCSI session using the iscsiadm utility

This procedure describes how to monitor the iscsi session using the **iscsiadm** utility.

By default, an iSCSI service is **lazily** started and the service starts after running the **iscsiadm** command. If root is not on an iSCSI device or there are no nodes marked with **node.startup = automatic** then the iSCSI service will not start until an **iscsiadm** command is executed that requires **iscsid** or the **iscsi** kernel modules to be started.

To force the **iscsid** daemon to run and iSCSI kernel modules to load:

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

Prerequisites

- Installed iscsi-initiator-utils on client machine:

```
yum install iscsi-initiator-utils
```

Procedure

1. Find information about the running sessions:

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

This command displays the session or device state, session ID (sid), some negotiated parameters, and the SCSI devices accessible through the session.

- For shorter output, for example, to display only the **sid-to-node** mapping, run:

```
# 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
```

These commands print the list of running sessions in the following format: **driver [sid] target_ip:port,target_portal_group_tag proper_target_name**.

Additional resources

- **/usr/share/doc/iscsi-initiator-utils-version/README** file.
- The **iscsiadm** man page.

5.3. REMOVING AN ISCSI TARGET

As a system administrator, you can remove the iSCSI target.

5.3.1. Removing an iSCSI object using targetcli tool

This procedure describes how to remove the iSCSI objects using the **targetcli** tool.

Procedure

1. Log off from the target:

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

   For more information on how to log in to the target, see Section 5.1.12, “Creating an iSCSI initiator”.

2. Remove the entire target, including all ACLs, LUNs, and portals:

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

   Replace *iqn.2006-04.com.example:444* with the target_iqn_name.

   - To remove an iSCSI backstore:

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

     - Replace *backstore-type* with either **fileio**, **block**, **pscsi**, or **ramdisk**.
     - Replace *block_backend* with the *backstore-name* you want to delete.

   - To remove parts of an iSCSI target, such as an ACL:

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

3. View the changes:

```
/> iscsi/ ls
```

Additional resources

- The **targetcli** man page.

CHAPTER 6. USING FIBRE CHANNEL DEVICES

Red Hat Enterprise Linux 8 ships with the following native Fibre Channel drivers:

- lpfc
- qla2xxx
- zfcp

6.1. FIBRE CHANNEL API

Following is the list of **/sys/class/** directories that contain files used to provide the userspace API. In each item, host numbers are designated by **H**, bus numbers are **B**, targets are **T**, logical unit numbers (LUNs) are **L**, and remote port numbers are **R**.

IMPORTANT

If your system is using multipath software, Red Hat recommends that you consult your hardware vendor before changing any of the values described in this section.

- Transport: **/sys/class/fc_transport/targetH:B:T/**
  - **port_id**: 24-bit port ID/address
  - **node_name**: 64-bit node name
  - **port_name**: 64-bit port name
- Remote Port: **/sys/class/fc_remote_ports/rport-H:B-R/**
  - **port_id**
  - **node_name**
  - **port_name**
  - **dev_loss_tmo**: controls when the scsi device gets removed from the system. After **dev_loss_tmo** triggers, the scsi device is removed. In the **multipath.conf** file , you can set **dev_loss_tmo** to **infinity**.
    In Red Hat Enterprise Linux 8, if you do not set the **fast_io_fail_tmo** option, **dev_loss_tmo** is capped to **600** seconds. By default, **fast_io_fail_tmo** is set to **5** seconds in Red Hat Enterprise Linux 8 if the **multipathd** service is running; otherwise, it is set to **off**.
  - **fast_io_fail_tmo**: specifies the number of seconds to wait before it marks a link as "bad". Once a link is marked bad, existing running I/O or any new I/O on its corresponding path fails.
    If I/O is in a blocked queue, it will not be failed until **dev_loss_tmo** expires and the queue is unblocked.

    If **fast_io_fail_tmo** is set to any value except off, **dev_loss_tmo** is uncapped. If **fast_io_fail_tmo** is set to off, no I/O fails until the device is removed from the system. If **fast_io_fail_tmo** is set to a number, I/O fails immediately when the **fast_io_fail_tmo** timeout triggers.
- Host: **/sys/class/fc_host/hostH/**

- **port_id**
- **node_name**
- **port_name**
- **issue_lip**: instructs the driver to rediscover remote ports.

6.2. RESIZING FIBRE CHANNEL LOGICAL UNITS

As a system administrator, you can resize Fibre Channel logical units.

Procedure

1. To determine which devices are paths for a **multipath** logical unit:

```
multipath -ll
```

2. To re-scan Fibre Channel logical units on a system that uses multipathing:

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

Additional resources

- The **multipath** man page.

6.3. DETERMINING THE LINK LOSS BEHAVIOR OF DEVICE USING FIBRE CHANNEL

If a driver implements the Transport **dev_loss_tmo** callback, access attempts to a device through a link will be blocked when a transport problem is detected.

Procedure

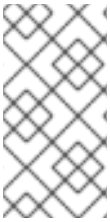
1. To determine the state of a remote port:

```
$ cat /sys/class/fc_remote_port/rport-H:B:R/port_state
```

This command will return any one of the following output:

- **Blocked** when the remote port along with devices accessed through it are blocked.
- **Online** if the remote port is operating normally
  If the problem is not resolved within **dev_loss_tmo** seconds, the **rport** and devices will be unblocked. All I/O running on that device along with any new I/O sent to that device will fail.

When a link loss exceeds **dev_loss_tmo**, the **scsi_device** and **sdN** devices are removed. Typically, the Fibre Channel class will leave the device as is; i.e. /**dev**/***sdx*** will remain /**dev**/***sdx***. This is because the target binding is saved by the Fibre Channel driver and when the target port returns, the SCSI addresses are recreated faithfully. However, this cannot be guaranteed; the *sdx* will be restored only if no additional change on in-storage box configuration of LUNs is made.



NOTE

Using **multipath**, you can modify the link loss behavior of device. For more information, see How to set dev_loss_tmo and fast_io_fail_tmo persistently, using a udev rule and Recommended tuning at scsi,multipath and at application layer while configuring Oracle RAC cluster Knowledgebase article.

Additional resources

- The **multipath.conf** man page

CHAPTER 7. CONFIGURING A FIBRE CHANNEL OVER AN ETHERNET INTERFACE

Red Hat Enterprise Linux 8 ships with the following native FCoE drivers:

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

7.1. CONFIGURING AN ETHERNET INTERFACE TO USE FCOE

As a system administrator, you can configure FCoE for **bnx2fc** driver.

Prerequisites

- Setting up and deploying a FCoE interface requires the **fcoe-utils** package:

```
# yum install fcoe-utils
```

Procedure

1. To configure a new virtual LAN (VLAN), create a copy of an existing network script:

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

   Replace ***/etc/fcoe/cfg-ethx*** with a network script and ***/etc/fcoe/cfg-ethX*** with an Ethernet device that supports FCoE.

   Modify the contents of the **cfg-*ethX*** file as required.

2. If you want the device to automatically load during boot time, set the following parameter in the *ifcfg-ethX* file.

```
# vi /etc/sysconfig/network-scripts/ifcfg-ethX

ONBOOT=yes
```

   For example, if the FCoE device is *eth2*, edit the **/etc/sysconfig/network-scripts/ifcfg-*eth2*** file accordingly.

3. To load the FCoE device:

```
# ip link set dev ethX up
```

4. To start the FCoE:

```
# systemctl start fcoe
```

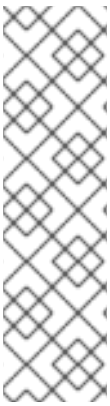
   The FCoE device displays if all other settings on the fabric are correct.

5. To view configured FCoE devices:

```
# fcoeadm -i
```

6. After correctly configuring the Ethernet interface to use FCoE, Red Hat recommends that you set FCoE service to run at startup.

```
# systemctl enable fcoe
```



NOTE

To stop the daemon:

```
# systemctl stop fcoe
```

Stopping the daemon does not reset the configuration of FCoE interfaces. To reset the configuration:

```
# systemctl -s SIGHUP kill fcoe
```

Additional resources

- The **fcoe** man page
- The **fcoeadm** man page
- FCoE software removal

7.2. CONFIGURING AN FCOE INTERFACE TO AUTOMATICALLY MOUNT AT BOOT

You can mount newly discovered disks via **udev** rules, **autofs**, and other similar methods. If a service requires the FCoE disk be mounted at boot-time, the FCoE disk should be mounted as soon as the fcoe service runs and before the initiation of any service that requires the FCoE disk. The FCoE mounting codes may differ depending on the system configuration, for example, a simple formatted FCoE disk, LVM, or a multipathed device node.

Procedure

1. To configure an FCoE disk to automatically mount at boot, add appropriate FCoE mounting code to the startup script for the **fcoe** service. The fcoe startup script is in the **/lib/systemd/system/fcoe.service** file.
   The following is a sample FCoE mounting code for mounting file systems specified via wild cards in the **/etc/fstab** file:

```
mount_fcoe_disks_from_fstab()
{
  local timeout=20
 local done=1
 local fcoe_disks=($(egrep 'by-path\/fc-.*_netdev' /etc/fstab | cut -d ' ' -f1))

 test -z $fcoe_disks && return 0
```

```
echo -n "Waiting for fcoe disks . "
 while [ $timeout -gt 0 ]; do
 for disk in ${fcoe_disks[*]}; do
   if ! test -b $disk; then
   done=0
  break
   fi
done

test $done -eq 1 && break;
sleep 1
echo -n ". "
done=1
let timeout--
done

if test $timeout -eq 0; then
 echo "timeout!"
else
 echo "done!"
fi

# mount any newly discovered disk
mount -a 2>/dev/null
}
```

2. To start the FCoE:

```
# systemctl start fcoe
```

The **mount_fcoe_disks_from_fstab** function should be invoked after the **fcoe** service script starts the **fcoemond** daemon. This will mount FCoE disks specified by the following paths in the **/etc/fstab** file:

```
/dev/disk/by-path/fc-0xXX:0xXX /mnt/fcoe-disk1 ext4  defaults,_netdev    0 0
```

```
/dev/disk/by-path/fc-0xYY:0xYY /mnt/fcoe-disk2 ext3  defaults,_netdev    0 0
```

Entries with **fc-** and **_netdev** sub-strings enable the **mount_fcoe_disks_from_fstab** function to identify FCoE disk mount entries.

NOTE

The **fcoe** service does not implement a timeout for FCoE disk discovery. The FCoE mounting code should implement its own timeout period.

Additional resources

- The **fcoe** man page
- The **fstab** man page.
- The **/usr/share/doc/fcoe-utils-version/README** file.

CHAPTER 8. MANAGING LAYERED LOCAL STORAGE WITH STRATIS

You can easily set up and manage complex storage configurations integrated by the Stratis high-level system.

IMPORTANT

Stratis is available as a Technology Preview. For information on Red Hat scope of support for Technology Preview features, see the Technology Preview Features Support Scope document.

Customers deploying Stratis are encouraged to provide feedback to Red Hat.

8.1. SETTING UP STRATIS FILE SYSTEMS

As a system administrator, you can enable and set up the Stratis volume-managing file system on your system to easily manage layered storage.

8.1.1. The purpose and features of Stratis

Stratis is a local storage-management solution for Linux. It is focused on simplicity and ease of use, and gives you access to advanced storage features.

Stratis makes the following activities easier:

- Initial configuration of storage
- Making changes later
- Using advanced storage features

Stratis is a hybrid user-and-kernel local storage management system that supports advanced storage features. The central concept of Stratis is a storage *pool*. This pool is created from one or more local disks or partitions, and volumes are created from the pool.

The pool enables many useful features, such as:

- File system snapshots
- Thin provisioning
- Tiering

8.1.2. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.

A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

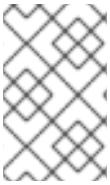
Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.



NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

8.1.3. Block devices usable with Stratis

This section lists storage devices that you can use for Stratis.

Supported devices
Stratis pools have been tested to work on these types of block devices:

- LUKS
- LVM logical volumes
- MDRAID
- DM Multipath
- iSCSI
- HDDs and SSDs
- NVMe devices

WARNING

In the current version, Stratis does not handle failures in hard drives or other hardware. If you create a Stratis pool over multiple hardware devices, you increase the risk of data loss because multiple devices must be operational to access the data.

Unsupported devices
Because Stratis contains a thin-provisioning layer, Red Hat does not recommend placing a Stratis pool on block devices that are already thinly-provisioned.

Additional resources

- For iSCSI and other block devices requiring network, see the **systemd.mount(5)** man page for information on the **_netdev** mount option.

8.1.4. Installing Stratis

This procedure installs all packages necessary to use Stratis.

Procedure

1. Install packages that provide the Stratis service and command-line utilities:

```
# yum install stratisd stratis-cli
```

2. Make sure that the **stratisd** service is enabled:

```
# systemctl enable --now stratisd
```

8.1.5. Creating a Stratis pool

This procedure creates a Stratis pool from one or more block devices.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- The block devices on which you are creating a Stratis pool are not in use and not mounted.
- The block devices on which you are creating a Stratis pool are at least 1 GiB in size each.
- On the IBM Z architecture, the **/dev/dasd*** block devices must to be partitioned. Use the partition in the Stratis pool.
  For information on partitioning DASD devices, see Configuring a Linux instance on IBM Z .

Procedure

1. If the selected block device contains file system, partition table, or RAID signatures, erase them:

```
# wipefs --all block-device
```

Replace *block-device* with the path to a block device, such as **/dev/sdb**.

2. To create a Stratis pool on the block device, use:

```
# stratis pool create my-pool block-device
```

- Replace *my-pool* with an arbitrary name for the pool.
- Replace *block-device* with the path to the empty or wiped block device, such as **/dev/sdb**.

To create a pool from more than one block device, list them all on the command line:

```
# stratis pool create my-pool device-1 device-2 device-n
```

3. To verify, list all pools on your system:

```
# stratis pool list
```

Additional resources

- The **stratis(8)** man page

Next steps

- Create a Stratis file system on the pool. See Section 8.1.6, "Creating a Stratis file system" .

8.1.6. Creating a Stratis file system

This procedure creates a Stratis file system on an existing Stratis pool.

Prerequisites

- Stratis is installed. See Section 8.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis pool. See Section 8.1.5, "Creating a Stratis pool" .

Procedure

1. To create a Stratis file system on a pool, use:

```
# stratis fs create my-pool my-fs
```

- Replace *my-pool* with the name of your existing Stratis pool.
- Replace *my-fs* with an arbitrary name for the file system.

2. To verify, list file systems within the pool:

```
# stratis fs list my-pool
```

Additional resources

- The **stratis(8)** man page

Next steps

- Mount the Stratis file system. See Section 8.1.7, “Mounting a Stratis file system” .

8.1.7. Mounting a Stratis file system

This procedure mounts an existing Stratis file system to access the content.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 8.1.6, “Creating a Stratis file system” .

Procedure

- To mount the file system, use the entries that Stratis maintains in the **/stratis/** directory:

```
# mount /stratis/my-pool/my-fs mount-point
```

The file system is now mounted on the *mount-point* directory and ready to use.

Additional resources

- The **mount(8)** man page

8.1.8. Persistently mounting a Stratis file system

This procedure persistently mounts a Stratis file system so that it is available automatically after booting the system.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 8.1.6, “Creating a Stratis file system” .

Procedure

1. Determine the UUID attribute of the file system:

```
$ lsblk --output=UUID /stratis/my-pool/my-fs
```

For example:

Example 8.1. Viewing the UUID of Stratis file system

```
$ lsblk --output=UUID /stratis/my-pool/fs1

UUID
```

```
a1f0b64a-4ebb-4d4e-9543-b1d79f600283
```

2. If the mount point directory does not exist, create it:

```
# mkdir --parents mount-point
```

3. As root, edit the **/etc/fstab** file and add a line for the file system, identified by the UUID. Use **xfs** as the file system type and add the **x-systemd.requires=stratisd.service** option.
For example:

Example 8.2. The /fs1 mount point in /etc/fstab

```
UUID=a1f0b64a-4ebb-4d4e-9543-b1d79f600283 /fs1 xfs defaults,x-
systemd.requires=stratisd.service 0 0
```

4. Regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

5. Try mounting the file system to verify that the configuration works:

```
# mount mount-point
```

Additional resources

- Persistently mounting file systems

8.1.9. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

8.2. EXTENDING A STRATIS VOLUME WITH ADDITIONAL BLOCK DEVICES

You can attach additional block devices to a Stratis pool to provide more storage capacity for Stratis file systems.

8.2.1. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.
A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

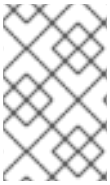
Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.



NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

8.2.2. Adding block devices to a Stratis pool

This procedure adds one or more block devices to a Stratis pool to be usable by Stratis file systems.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- The block devices that you are adding to the Stratis pool are not in use and not mounted.
- The block devices that you are adding to the Stratis pool are at least 1 GiB in size each.

Procedure

- To add one or more block devices to the pool, use:

```
# stratis pool add-data my-pool device-1 device-2 device-n
```

Additional resources

- The **stratis(8)** man page

8.2.3. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

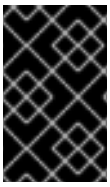
8.3. MONITORING STRATIS FILE SYSTEMS

As a Stratis user, you can view information about Stratis volumes on your system to monitor their state and free space.

8.3.1. Stratis sizes reported by different utilities

This section explains the difference between Stratis sizes reported by standard utilities such as **df** and the **stratis** utility.

Standard Linux utilities such as **df** report the size of the XFS file system layer on Stratis, which is 1 TiB. This is not useful information, because the actual storage usage of Stratis is less due to thin provisioning, and also because Stratis automatically grows the file system when the XFS layer is close to full.



IMPORTANT

Regularly monitor the amount of data written to your Stratis file systems, which is reported as the *Total Physical Used* value. Make sure it does not exceed the *Total Physical Size* value.

Additional resources

- The **stratis(8)** man page

8.3.2. Displaying information about Stratis volumes

This procedure lists statistics about your Stratis volumes, such as the total, used, and free size or file systems and block devices belonging to a pool.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.

Procedure

- To display information about all **block devices** used for Stratis on your system:

```
# stratis blockdev

Pool Name  Device Node    Physical Size   State  Tier
my-pool    /dev/sdb           9.10 TiB  In-use  Data
```

- To display information about all Stratis **pools** on your system:

```
# stratis pool

Name    Total Physical Size  Total Physical Used
my-pool          9.10 TiB             598 MiB
```

- To display information about all Stratis **file systems** on your system:

```
# stratis filesystem

Pool Name  Name  Used     Created            Device
my-pool    my-fs 546 MiB  Nov 08 2018 08:03  /stratis/my-pool/my-fs
```

Additional resources

- The **stratis(8)** man page

8.3.3. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

8.4. USING SNAPSHOTS ON STRATIS FILE SYSTEMS

You can use snapshots on Stratis file systems to capture file system state at arbitrary times and restore it in the future.

8.4.1. Characteristics of Stratis snapshots

This section describes the properties and limitations of file system snapshots on Stratis.

In Stratis, a snapshot is a regular Stratis file system created as a copy of another Stratis file system. The snapshot initially contains the same file content as the original file system, but can change as the snapshot is modified. Whatever changes you make to the snapshot will not be reflected in the original file system.

The current snapshot implementation in Stratis is characterized by the following:

- A snapshot of a file system is another file system.
- A snapshot and its origin are not linked in lifetime. A snapshotted file system can live longer than the file system it was created from.
- A file system does not have to be mounted to create a snapshot from it.
- Each snapshot uses around half a gigabyte of actual backing storage, which is needed for the XFS log.

8.4.2. Creating a Stratis snapshot

This procedure creates a Stratis file system as a snapshot of an existing Stratis file system.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 8.1.6, “Creating a Stratis file system” .

Procedure

- To create a Stratis snapshot, use:

```
# stratis fs snapshot my-pool my-fs my-fs-snapshot
```

Additional resources

- The **stratis(8)** man page

8.4.3. Accessing the content of a Stratis snapshot

This procedure mounts a snapshot of a Stratis file system to make it accessible for read and write operations.

Prerequisites

- Stratis is installed. See Section 8.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 8.4.2, "Creating a Stratis snapshot" .

Procedure

- To access the snapshot, mount it as a regular file system from the /**stratis**/***my-pool***/ directory:

```
# mount /stratis/my-pool/my-fs-snapshot mount-point
```

Additional resources

- Section 8.1.7, "Mounting a Stratis file system"
- The **mount(8)** man page

8.4.4. Reverting a Stratis file system to a previous snapshot

This procedure reverts the content of a Stratis file system to the state captured in a Stratis snapshot.

Prerequisites

- Stratis is installed. See Section 8.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 8.4.2, "Creating a Stratis snapshot" .

Procedure

1. Optionally, back up the current state of the file system to be able to access it later:

```
# stratis filesystem snapshot my-pool my-fs my-fs-backup
```

2. Unmount and remove the original file system:

```
# umount /stratis/my-pool/my-fs
# stratis filesystem destroy my-pool my-fs
```

3. Create a copy of the snapshot under the name of the original file system:

```
# stratis filesystem snapshot my-pool my-fs-snapshot my-fs
```

4. Mount the snapshot, which is now accessible with the same name as the original file system:

```
# mount /stratis/my-pool/my-fs mount-point
```

The content of the file system named *my-fs* is now identical to the snapshot *my-fs-snapshot*.

Additional resources

- The **stratis(8)** man page

8.4.5. Removing a Stratis snapshot

This procedure removes a Stratis snapshot from a pool. Data on the snapshot are lost.

Prerequisites

- Stratis is installed. See Section 8.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 8.4.2, "Creating a Stratis snapshot" .

Procedure

1. Unmount the snapshot:

```
# umount /stratis/my-pool/my-fs-snapshot
```

2. Destroy the snapshot:

```
# stratis filesystem destroy my-pool my-fs-snapshot
```

Additional resources

- The **stratis(8)** man page

8.4.6. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

8.5. REMOVING STRATIS FILE SYSTEMS

You can remove an existing Stratis file system or a Stratis pool, destroying data on them.

8.5.1. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.
A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

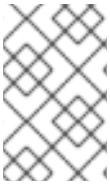
Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.



NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

8.5.2. Removing a Stratis file system

This procedure removes an existing Stratis file system. Data stored on it are lost.

Prerequisites

- Stratis is installed. See Section 8.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 8.1.6, “Creating a Stratis file system” .

Procedure

1. Unmount the file system:

```
# umount /stratis/my-pool/my-fs
```

2. Destroy the file system:

```
# stratis filesystem destroy my-pool my-fs
```

3. Verify that the file system no longer exists:

```
# stratis filesystem list my-pool
```

Additional resources

- The **stratis(8)** man page

8.5.3. Removing a Stratis pool

This procedure removes an existing Stratis pool. Data stored on it are lost.

Prerequisites

- Stratis is installed. See Section 8.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis pool. See Section 8.1.5, "Creating a Stratis pool" .

Procedure

1. List file systems on the pool:

```
# stratis filesystem list my-pool
```

2. Unmount all file systems on the pool:

```
# umount /stratis/my-pool/my-fs-1 \
        /stratis/my-pool/my-fs-2 \
        /stratis/my-pool/my-fs-n
```

3. Destroy the file systems:

```
# stratis filesystem destroy my-pool my-fs-1 my-fs-2
```

4. Destroy the pool:

```
# stratis pool destroy my-pool
```

5. Verify that the pool no longer exists:

```
# stratis pool list
```

Additional resources

- The **stratis(8)** man page

8.5.4. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/