Red Hat Enterprise Linux 8

Managing storage devices

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

Last Updated: 2022-02-02

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

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

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:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. 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**.

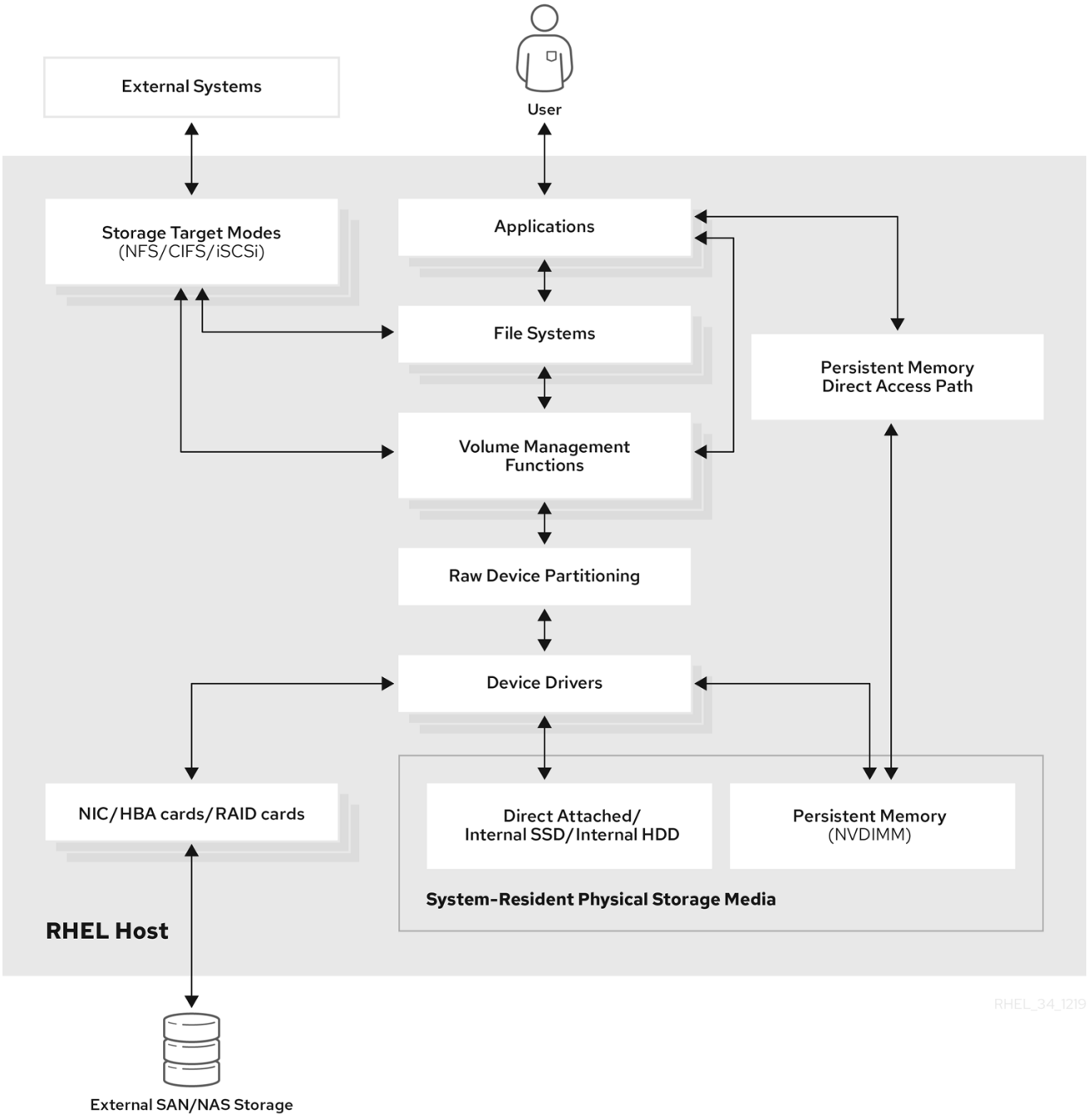
CHAPTER 1. AVAILABLE STORAGE OPTIONS OVERVIEW

There are several local, remote, and cluster-based storage options available on Red Hat Enterprise Linux 8.

Local storage implies that the storage devices are either installed on the system or directly attached to the system.

With remote storage, devices are accessed over LAN, the internet, or using a Fibre channel network. High Level Red Hat Enterprise Linux Storage Diagram describes the different storage options.

Figure 1.1. High Level Red Hat Enterprise Linux Storage Diagram



1.1. LOCAL STORAGE OVERVIEW

Red Hat Enterprise Linux 8 offers several local storage options.

Basic disk administration

Using **parted** and **fdisk**, you can create, modify, delete, and view disk partitions. The following are the partitioning layout standards:

Master Boot Record (MBR)

It is used with BIOS-based computers. You can create primary, extended, and logical partitions.

GUID Partition Table (GPT)

It uses Globally Unique identifier (GUID) and provides unique disk and partition GUID.

To encrypt the partition, you can use Linux Unified Key Setup-on-disk-format (LUKS). To encrypt the partition, select the option during the installation and the prompt displays to enter the passphrase. This passphrase unlocks the encryption key.

Storage consumption options

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

It is a combination of memory and storage. You can enable and manage various types of storage on NVDIMM devices connected to your system.

Block Storage Management

Data is stored in the form of blocks where each block has a unique identifier.

File Storage

Data is stored at file level on the local system. These data can be accessed locally using XFS (default) or ext4, and over a network by using NFS and SMB.

Logical volumes

Logical Volume Manager (LVM)

It creates logical devices from physical devices. Logical volume (LV) is a combination of the physical volumes (PV) and volume groups (VG). Configuring LVM include:

- Creating PV from the hard drives.
- Creating VG from the PV.
- Creating LV from the VG assigning mount points to the LV.

Virtual Data Optimizer (VDO)

It is used for data reduction by using deduplication, compression, and thin provisioning. Using LV below VDO helps in:

- Extending of VDO volume
- Spanning VDO volume over multiple devices

Local file systems

XFS

The default RHEL file system.

Ext4

A legacy file system.

Stratis

It is available as a Technology Preview. Stratis is a hybrid user-and-kernel local storage management system that supports advanced storage features.

1.2. REMOTE STORAGE OVERVIEW

Following are the remote storage options available in Red Hat Enterprise Linux 8:

Storage connectivity options

iSCSI

RHEL 8 uses the targetcli tool to add, remove, view, and monitor iSCSI storage interconnects.

Fibre Channel (FC)

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

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

Non-volatile Memory Express (NVMe)

An interface which allows host software utility to communicate with solid state drives. Use the following types of fabric transport to configure NVMe over fabrics:

- NVMe over fabrics using Remote Direct Memory Access (RDMA).
- NVMe over fabrics using Fibre Channel (FC)

Device Mapper multipathing (DM Multipath)

Allows you to configure multiple I/O paths between server nodes and storage arrays into a single device. These I/O paths are physical SAN connections that can include separate cables, switches, and controllers.

Network file system

- NFS
- SMB

1.3. GFS2 FILE SYSTEM OVERVIEW

The Red Hat Global File System 2 (GFS2) file system is a 64-bit symmetric cluster file system which provides a shared name space and manages coherency between multiple nodes sharing a common block device. A GFS2 file system is intended to provide a feature set which is as close as possible to a local file system, while at the same time enforcing full cluster coherency between nodes. To achieve this, the nodes employ a cluster-wide locking scheme for file system resources. This locking scheme uses communication protocols such as TCP/IP to exchange locking information.

In a few cases, the Linux file system API does not allow the clustered nature of GFS2 to be totally transparent; for example, programs using POSIX locks in GFS2 should avoid using the **GETLK** function since, in a clustered environment, the process ID may be for a different node in the cluster. In most cases however, the functionality of a GFS2 file system is identical to that of a local file system.

The Red Hat Enterprise Linux (RHEL) Resilient Storage Add-On provides GFS2, and it depends on the RHEL High Availability Add-On to provide the cluster management required by GFS2.

The **gfs2.ko** kernel module implements the GFS2 file system and is loaded on GFS2 cluster nodes.

To get the best performance from GFS2, it is important to take into account the performance considerations which stem from the underlying design. Just like a local file system, GFS2 relies on the page cache in order to improve performance by local caching of frequently used data. In order to maintain coherency across the nodes in the cluster, cache control is provided by the *glock* state machine.

Additional resources

- Configuring GFS2 file systems

1.4. GLUSTER STORAGE OVERVIEW

The Red Hat Gluster Storage (RHGS) is a software-defined storage platform that can be deployed in clusters. It aggregates disk storage resources from multiple servers into a single global namespace. GlusterFS is an open source distributed file system that is suitable for cloud and hybrid solutions.

Volumes form the base for GlusterFS and provide different requirements. Each volume is a collection of bricks, which are basic units of storage that are represented by an export directory on a server in the trusted storage pool.

The following types of GlusterFS volumes are available:

- **Distributed GlusterFS volume** is the default volume where each file is stored in one brick and the file cannot be shared between different bricks.
- **Replicated GlusterFS volume** type replicates user data, so that if one brick fails, the data is still accessible.
- **Distributed replicated GlusterFS volume** is a hybrid volume that distributes replicas over a large number of systems. It is suitable for environments where storage scalability and high-reliability are critical.

Additional resources

- Red Hat gluster storage administration guide

1.5. CEPH STORAGE OVERVIEW

Red Hat Ceph Storage (RHCS) is a scalable, open, software-defined storage platform that combines the most stable version of the Ceph storage system with a Ceph management platform, deployment utilities, and support services.

Red Hat Ceph Storage is designed for cloud infrastructure and web-scale object storage. Red Hat Ceph Storage clusters consist of the following types of nodes:

Red Hat Ceph Storage Ansible administration node

This type of node acts as the traditional Ceph Administration node did for previous versions of Red Hat Ceph Storage. This type of node provides the following functions:

- Centralized storage cluster management

- The Ceph configuration files and keys
- Optionally, local repositories for installing Ceph on nodes that cannot access the Internet for security reasons

Monitor nodes

Each monitor node runs the monitor daemon (**ceph-mon**), which maintains a copy of the cluster map. The cluster map includes the cluster topology. A client connecting to the Ceph cluster retrieves the current copy of the cluster map from the monitor which enables the client to read from and write data to the cluster.

IMPORTANT

Ceph can run with one monitor; however, to ensure high availability in a production cluster, Red Hat will only support deployments with at least three monitor nodes. Red Hat recommends deploying a total of 5 Ceph Monitors for storage clusters exceeding 750 OSDs.

OSD nodes

Each Object Storage Device (OSD) node runs the Ceph OSD daemon (**ceph-osd**), which interacts with logical disks attached to the node. Ceph stores data on these OSD nodes.
Ceph can run with very few OSD nodes, which the default is three, but production clusters realize better performance beginning at modest scales, for example 50 OSDs in a storage cluster. Ideally, a Ceph cluster has multiple OSD nodes, allowing isolated failure domains by creating the CRUSH map.

MDS nodes

Each Metadata Server (MDS) node runs the MDS daemon (**ceph-mds**), which manages metadata related to files stored on the Ceph File System (CephFS). The MDS daemon also coordinates access to the shared cluster.

Object Gateway node

Ceph Object Gateway node runs the Ceph RADOS Gateway daemon (**ceph-radosgw**), and is an object storage interface built on top of **librados** to provide applications with a RESTful gateway to Ceph Storage Clusters. The Ceph Object Gateway supports two interfaces:

S3

Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.

Swift

Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.

Additional resources

- Red Hat Ceph Storage

CHAPTER 2. MANAGING LOCAL STORAGE USING RHEL SYSTEM ROLES

To manage LVM and local file systems (FS) using Ansible, you can use the **storage** role, which is one of the RHEL System Roles available in RHEL 8.

Using the **storage** role enables you to automate administration of file systems on disks and logical volumes on multiple machines and across all versions of RHEL starting with RHEL 7.7.

For more information about RHEL System Roles and how to apply them, see Introduction to RHEL System Roles.

2.1. INTRODUCTION TO THE STORAGE ROLE

The **storage** role can manage:

- File systems on disks which have not been partitioned
- Complete LVM volume groups including their logical volumes and file systems

With the **storage** role you can perform the following tasks:

- Create a file system
- Remove a file system
- Mount a file system
- Unmount a file system
- Create LVM volume groups
- Remove LVM volume groups
- Create logical volumes
- Remove logical volumes
- Create RAID volumes
- Remove RAID volumes
- Create LVM pools with RAID
- Remove LVM pools with RAID

2.2. PARAMETERS THAT IDENTIFY A STORAGE DEVICE IN THE STORAGE SYSTEM ROLE

Your **storage** role configuration affects only the file systems, volumes, and pools that you list in the following variables.

storage_volumes

List of file systems on all unpartitioned disks to be managed.
Partitions are currently unsupported.

storage_pools

List of pools to be managed.
Currently the only supported pool type is LVM. With LVM, pools represent volume groups (VGs). Under each pool there is a list of volumes to be managed by the role. With LVM, each volume corresponds to a logical volume (LV) with a file system.

2.3. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AN XFS FILE SYSTEM ON A BLOCK DEVICE

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an XFS file system on a block device using the default parameters.

WARNING

The **storage** role can create a file system only on an unpartitioned, whole disk or a logical volume (LV). It cannot create the file system on a partition.

Example 2.1. A playbook that creates XFS on /dev/sdb

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
  roles:
    - rhel-system-roles.storage
```

- The volume name (***barefs*** in the example) is currently arbitrary. The **storage** role identifies the volume by the disk device listed under the **disks:** attribute.
- You can omit the **fs_type: xfs** line because XFS is the default file system in RHEL 8.
- To create the file system on an LV, provide the LVM setup under the **disks:** attribute, including the enclosing volume group. For details, see Example Ansible playbook to manage logical volumes.
  Do not provide the path to the LV device.

Additional resources

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

2.4. EXAMPLE ANSIBLE PLAYBOOK TO PERSISTENTLY MOUNT A FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to immediately and persistently mount an XFS file system.

Example 2.2. A playbook that mounts a file system on /dev/sdb to /mnt/data

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

- This playbook adds the file system to the **/etc/fstab** file, and mounts the file system immediately.
- If the file system on the **/dev/sdb** device or the mount point directory do not exist, the playbook creates them.

Additional resources

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

2.5. EXAMPLE ANSIBLE PLAYBOOK TO MANAGE LOGICAL VOLUMES

This section provides an example Ansible playbook. This playbook applies the **storage** role to create an LVM logical volume in a volume group.

Example 2.3. A playbook that creates a mylv logical volume in the myvg volume group

```
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt
  roles:
    - rhel-system-roles.storage
```

- The **myvg** volume group consists of the following disks:
  - **/dev/sda**
  - **/dev/sdb**
  - **/dev/sdc**
- If the **myvg** volume group already exists, the playbook adds the logical volume to the volume group.
- If the **myvg** volume group does not exist, the playbook creates it.
- The playbook creates an Ext4 file system on the **mylv** logical volume, and persistently mounts the file system at **/mnt**.

Additional resources

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

2.6. EXAMPLE ANSIBLE PLAYBOOK TO ENABLE ONLINE BLOCK DISCARD

This section provides an example Ansible playbook. This playbook applies the **storage** role to mount an XFS file system with online block discard enabled.

Example 2.4. A playbook that enables online block discard on /mnt/data/

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

Additional resources

- Example Ansible playbook to persistently mount a file system
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

2.7. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT4 FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext4 file system.

Example 2.5. A playbook that creates Ext4 on /dev/sdb and mounts it at /mnt/data

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

- The playbook creates the file system on the **/dev/sdb** disk.
- The playbook persistently mounts the file system at the ***/mnt/data*** directory.
- The label of the file system is ***label-name***.

Additional resources

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

2.8. EXAMPLE ANSIBLE PLAYBOOK TO CREATE AND MOUNT AN EXT3 FILE SYSTEM

This section provides an example Ansible playbook. This playbook applies the **storage** role to create and mount an Ext3 file system.

Example 2.6. A playbook that creates Ext3 on /dev/sdb and mounts it at /mnt/data

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

- The playbook creates the file system on the **/dev/sdb** disk.

- The playbook persistently mounts the file system at the ***/mnt/data*** directory.
- The label of the file system is ***label-name***.

Additional resources

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

2.9. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING EXT4 OR EXT3 FILE SYSTEM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** role to resize an existing Ext4 or Ext3 file system on a block device.

Example 2.7. A playbook that set up a single volume on a disk

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
  size: 12 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- If the volume in the previous example already exists, to resize the volume, you need to run the same playbook, just with a different value for the parameter **size**. For example:

Example 2.8. A playbook that resizes ext4 on /dev/sdb

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
  size: 10 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- The volume name (barefs in the example) is currently arbitrary. The storage role identifies the volume by the disk device listed under the disks: attribute.

NOTE

Using the **Resizing** action in other file systems can destroy the data on the device you are working on.

Additional resources

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

2.10. EXAMPLE ANSIBLE PLAYBOOK TO RESIZE AN EXISTING FILE SYSTEM ON LVM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage RHEL System Role to resize an LVM logical volume with a file system.

WARNING

Using the **Resizing** action in other file systems can destroy the data on the device you are working on.

Example 2.9. A playbook that resizes existing mylv1 and myvl2 logical volumes in the myvg volume group

```
---

- hosts: all
  vars:
   storage_pools:
     - name: myvg
       disks:
         - /dev/sda
         - /dev/sdb
         - /dev/sdc
       volumes:
          - name: mylv1
            size: 10 GiB
            fs_type: ext4
            mount_point: /opt/mount1
          - name: mylv2
            size: 50 GiB
            fs_type: ext4
            mount_point: /opt/mount2
```

```
- name: Create LVM pool over three disks
  incude_role:
    name: rhel-system-roles.storage
```

- This playbook resizes the following existing file systems:
  - The Ext4 file system on the **mylv1** volume, which is mounted at **/opt/mount1**, resizes to 10 GiB.
  - The Ext4 file system on the **mylv2** volume, which is mounted at **/opt/mount2**, resizes to 50 GiB.

Additional resources

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

2.11. EXAMPLE ANSIBLE PLAYBOOK TO CREATE A SWAP PARTITION USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the **storage** role to create a swap partition, if it does not exist, or to modify the swap partition, if it already exist, on a block device using the default parameters.

Example 2.10. A playbook that creates or modify an existing XFS on /dev/sdb

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

- The volume name (***swap_fs*** in the example) is currently arbitrary. The **storage** role identifies the volume by the disk device listed under the **disks:** attribute.

Additional resources

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

2.12. CONFIGURING A RAID VOLUME USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure a RAID volume on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure a RAID volume to suit your requirements.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **storage** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to deploy a RAID volume using the **storage** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_volumes:
      - name: data
        type: raid
        disks: [sdd, sde, sdf, sdg]
        raid_level: raid0
        raid_chunk_size: 32 KiB
        mount_point: /mnt/data
        state: present
  roles:
    - name: rhel-system-roles.storage
```

WARNING

Device names can change in certain circumstances; for example, when you add a new disk to a system. Therefore, to prevent data loss, we do not recommend using specific disk names in the playbook.

2. Optional. Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

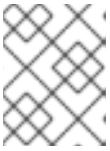
- Managing RAID.
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

2.13. CONFIGURING AN LVM POOL WITH RAID USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure an LVM pool with RAID on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **storage** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to configure an LVM pool with RAID using the **storage** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_pool
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
  roles:
    - name: rhel-system-roles.storage
```

NOTE

To create an LVM pool with RAID, you must specify the RAID type using the **raid_level** parameter.

2. Optional. Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Managing RAID.
- The **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file.

2.14. EXAMPLE ANSIBLE PLAYBOOK TO COMPRESS AND DEDUPLICATE A VDO VOLUME ON LVM USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage RHEL System Role to enable compression and deduplication to a Logical Manager Volumes (LVM) using the Virtual Data Optimizer (VDO) volume.

Example 2.11. A playbook that creates a mylv1 LVM VDO volume in the myvg volume group

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

In this example, the **compression** and **deduplication** pools are set to true, which specifies that the VDO is used. The following describes the usage of these parameters:

- The **deduplication** is used to deduplicate the duplicated data stored on the storage volume.

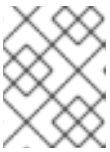
- The compression is used to compress the data stored on the storage volume, which results in more storage capacity.
- The vdo_pool_size specifies the actual size the volume takes on the device. The virtual size of VDO volume is set by the **size** parameter. NOTE: Because of the storage role use of LVM VDO, only one volume per pool can use the compression and deduplication.

2.15. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE ROLE

You can use the **storage** role to create and configure a volume encrypted with LUKS by running an Ansible playbook.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to create the volume.

- You have the **rhel-system-roles** package installed on the Ansible controller.
- You have an inventory file detailing the systems on which you want to deploy a LUKS encrypted volume using the storage System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

2. Optional: Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Encrypting block devices using LUKS
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file

2.16. EXAMPLE ANSIBLE PLAYBOOK TO EXPRESS POOL VOLUME SIZES AS PERCENTAGE USING THE STORAGE RHEL SYSTEM ROLE

This section provides an example Ansible playbook. This playbook applies the storage RHEL System Role to enable you to express Logical Manager Volumes (LVM) volume sizes as a percentage of the pool's total size.

Example 2.12. A playbook that express volume sizes as a percentage of the pool's total size

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

This example specifies the size of LVM volumes as a percentage of the pool size, for example: "60%". Additionally, you can also specify the size of LVM volumes as a percentage of the pool size in a human-readable size of the file system, for example, "10g" or "50 GiB".

2.17. ADDITIONAL RESOURCES

- **/usr/share/doc/rhel-system-roles/storage/**
- **/usr/share/ansible/roles/rhel-system-roles.storage/**

CHAPTER 3. 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.

3.1. 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. 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.

3.1.1. 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

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

3.1.2. 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 3.1. 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**.

3.2. 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.

3.2.1. 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.

NOTE

Red Hat recommends that, unless you have a reason for doing otherwise, you should *at least* create the following partitions: **swap**, /**boot**/, and / (root).

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.

3.2.2. Comparison of partition table types

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

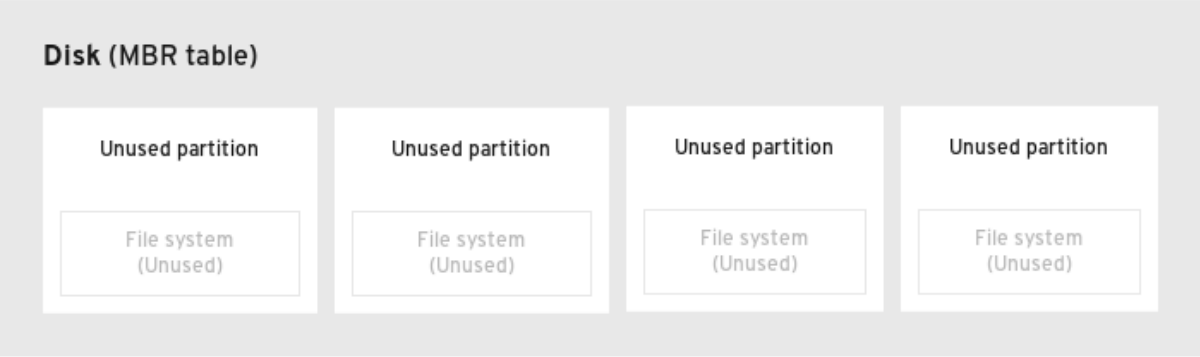
Table 3.1. Partition table types

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

3.2.3. MBR disk partitions

The diagrams in this chapter show the partition table as being separate from the actual disk. However, this is not entirely accurate. In reality, the partition table is stored at the very start of the disk, before any file system or user data, but for clarity, they are separate in the following diagrams.

Figure 3.1. Disk with MBR partition table



As the previous diagram shows, the partition table is divided into four sections of four primary partitions. A primary partition is a partition on a hard drive that can contain only one logical drive (or section). Each section can hold the information necessary to define a single partition, meaning that the partition table can define no more than four partitions.

Each partition table entry contains several important characteristics of the partition:

- The points on the disk where the partition starts and ends.
- Whether the partition is **active**. Only one partition can be flagged as **active**.
- The partition's type.

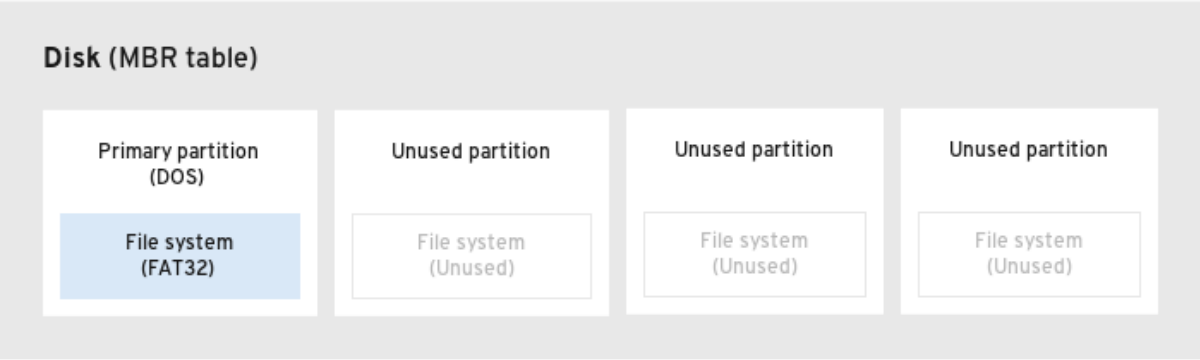
The starting and ending points define the partition's size and location on the disk. The "active" flag is used by some operating systems boot loaders. In other words, the operating system in the partition that is marked "active" is booted, in this case.

The type is a number that identifies the partition's anticipated usage. Some operating systems use the

partition type to denote a specific file system type, to flag the partition as being associated with a particular operating system, to indicate that the partition contains a bootable operating system, or some combination of the three.

The following diagram shows an example of a drive with single partition:

Figure 3.2. Disk with a single partition



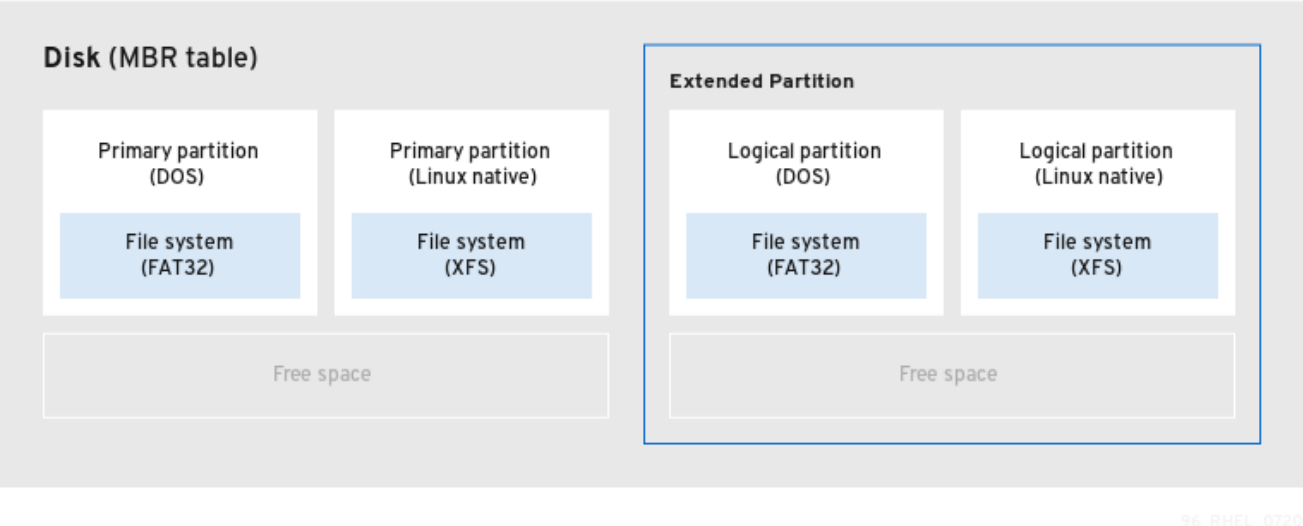
The single partition in this example is labeled as **DOS**. This label shows the partition type, with **DOS** being one of the most common ones.

3.2.4. Extended MBR partitions

In case four partitions are insufficient for your needs, you can use extended partitions to create up additional partitions. You can do this by setting the type of partition to "Extended".

An extended partition is like a disk drive in its own right – it has its own partition table, which points to one or more partitions (now called logical partitions, as opposed to the four primary partitions), contained entirely within the extended partition itself. The following diagram shows a disk drive with two primary partitions and one extended partition containing two logical partitions (along with some unpartitioned free space):

Figure 3.3. Disk with both a primary and an extended MBR partition



As this figure implies, there is a difference between primary and logical partitions – there can be only up

to four primary and extended partitions, but there is no fixed limit to the number of logical partitions that can exist. However, due to the way in which partitions are accessed in Linux, no more than 15 logical partitions can be defined on a single disk drive.

3.2.5. MBR partition types

The table below shows a list of some of the commonly used MBR partition types and hexadecimal numbers used to represent them.

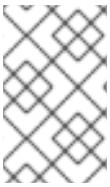
Table 3.2. MBR partition types

MBR partition type	Value	MBR partition type	Value
Empty	00	Novell Netware 386	65
DOS 12-bit FAT	01	PIC/IX	75
XENIX root	O2	Old MINIX	80
XENIX usr	O3	Linux/MINUX	81
DOS 16-bit ⇐32M	04	Linux swap	82
Extended	05	Linux native	83
DOS 16-bit >=32	06	Linux extended	85
OS/2 HPFS	07	Amoeba	93
AIX	08	Amoeba BBT	94
AIX bootable	09	BSD/386	a5
OS/2 Boot Manager	0a	OpenBSD	a6
Win95 FAT32	0b	NEXTSTEP	a7
Win95 FAT32 (LBA)	0c	BSDI fs	b7
Win95 FAT16 (LBA)	0e	BSDI swap	b8
Win95 Extended (LBA)	0f	Syrinx	c7
Venix 80286	40	CP/M	db
Novell	51	DOS access	e1
PRep Boot	41	DOS R/O	e3

GNU HURD	63	DOS secondary	f2
Novell Netware 286	64	BBT	ff

3.2.6. GUID Partition Table

The GUID Partition Table (GPT) is a partitioning scheme based on using Globally Unique Identifier (GUID). GPT was developed to cope with limitations of the MBR partition table, especially with the limited maximum addressable storage space of a disk. Unlike MBR, which is unable to address storage larger than 2 TiB (equivalent to approximately 2.2 TB), GPT is used with hard disks larger than this; the maximum addressable disk size is 2.2 ZiB. In addition, GPT, by default, supports creating up to 128 primary partitions. This number could be extended by allocating more space to the partition table.



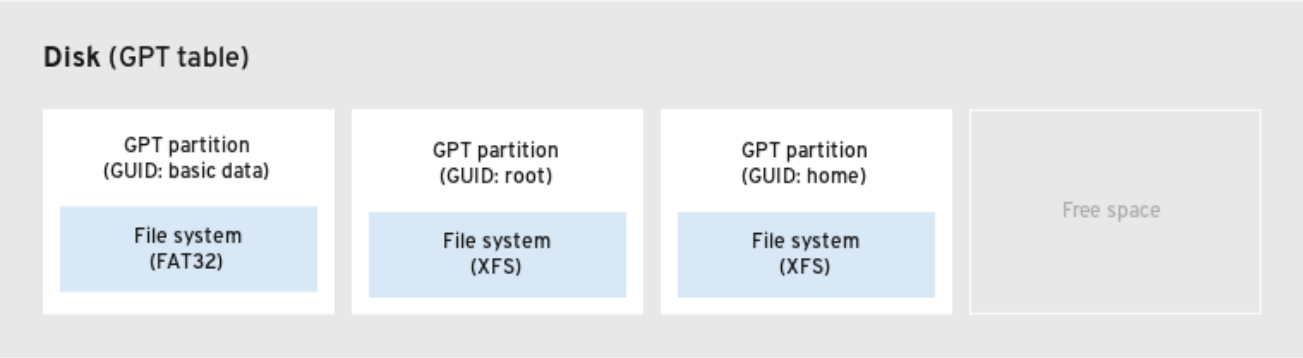
NOTE

A GPT has partition types based on GUIDs. Note that certain partitions require a specific GUID. For example, the system partition for EFI boot loaders require GUID **C12A7328-F81F-11D2-BA4B-00A0C93EC93B**.

GPT disks use logical block addressing (LBA) and the partition layout is as follows:

- To preserve backward compatibility with MBR disks, the first sector (LBA 0) of GPT is reserved for MBR data and it is called "protective MBR".
- The primary GPT header begins on the second logical block (LBA 1) of the device. The header contains the disk GUID, the location of the primary partition table, the location of the secondary GPT header, and CRC32 checksums of itself, and the primary partition table. It also specifies the number of partition entries on the table.
- The primary GPT includes, by default 128 partition entries, each with an entry size of 128 bytes, its partition type GUID and unique partition GUID.
- The secondary GPT is identical to the primary GPT. It is used mainly as a backup table for recovery in case the primary partition table is corrupted.
- The secondary GPT header is located on the last logical sector of the disk and it can be used to recover GPT information in case the primary header is corrupted. It contains the disk GUID, the location of the secondary partition table and the primary GPT header, CRC32 checksums of itself and the secondary partition table, and the number of possible partition entries.

Figure 3.4. Disk with a GUID Partition Table



IMPORTANT

There must be a BIOS boot partition for the boot loader to be installed successfully onto a disk that contains a GPT (GUID Partition table). This includes disks initialized by **Anaconda**. If the disk already contains a BIOS boot partition, it can be reused.

3.2.7. 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 3.2. 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

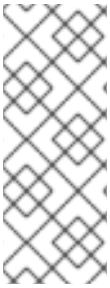
- **parted(8)** man page.

3.3. CREATING A PARTITION

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

3.3.1. 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.

NOTE

Red Hat recommends that, unless you have a reason for doing otherwise, you should *at least* create the following partitions: **swap**, /**boot**/, and / (root).

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.

3.3.2. 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.

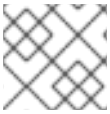
Partition file system type
The **parted** utility optionally accepts a file system type argument when creating a partition. The value is used to:

- 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**



NOTE

The only supported local file systems in RHEL 8 are **ext4** and **xfs**.

3.3.3. Partition naming scheme

Red Hat Enterprise Linux uses a file-based naming scheme, with file names in the form of /**dev**/***xxyN***.

Device and partition names consist of the following structure:

/dev/

This is the name of the directory in which all device files are located. Because partitions are placed on hard disks, and hard disks are devices, the files representing all possible partitions are located in /**dev**.

xx

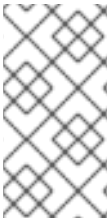
The first two letters of the partitions name indicate the type of device on which is the partition located, usually **sd**.

y

This letter indicates which device the partition is on. For example, /**dev**/**sda** for the first hard disk, /**dev**/**sdb** for the second, and so on. In systems with more than 26 drives, you can use more letters. For example, /**dev**/**sdaa1**.

N

The final letter indicates the number that represents the partition. The first four (primary or extended) partitions are numbered **1** through **4**. Logical partitions start at **5**. For example, /**dev**/**sda3** is the third primary or extended partition on the first hard disk, and /**dev**/**sdb6** is the second logical partition on the second hard disk. Drive partition numbering applies only to MBR partition tables. Note that ***N*** does not always mean partition.



NOTE

Even if Red Hat Enterprise Linux can identify and refer to *all* types of disk partitions, it might not be able to read the file system and therefore access stored data on every partition type. However, in many cases, it is possible to successfully access data on a partition dedicated to another operating system.

3.3.4. Mount points and disk partitions

In Red Hat Enterprise Linux, each partition is used to form part of the storage necessary to support a single set of files and directories. This is done using the process known as *mounting*, which associates a partition with a directory. Mounting a partition makes its storage available starting at the specified directory, known as a *mount point*.

For example, if partition **/dev/sda5** is mounted on **/usr/**, that would mean that all files and directories under **/usr/** physically reside on **/dev/sda5**. So the file **/usr/share/doc/FAQ/txt/Linux-FAQ** would be stored on **/dev/sda5**, while the file **/etc/gdm/custom.conf** would not.

Continuing the example, it is also possible that one or more directories below **/usr/** would be mount points for other partitions. For instance, a partition **/dev/sda7** could be mounted on **/usr/local**, meaning that **/usr/local/man/whatis** would then reside on **/dev/sda7** rather than **/dev/sda5**.

3.3.5. 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 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 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 3.3. 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

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

3.3.6. 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
```

3.4. 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.

3.4.1. 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.

NOTE

Red Hat recommends that, unless you have a reason for doing otherwise, you should *at least* create the following partitions: **swap**, /**boot**/, and / (root).

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.

3.4.2. 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

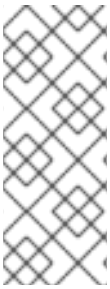
- **parted(8)** man page

3.5. 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.

3.5.1. 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.

NOTE

Red Hat recommends that, unless you have a reason for doing otherwise, you should *at least* create the following partitions: **swap**, /**boot**/, and / (root).

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.

3.5.2. 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 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 3.4. 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

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

3.6. STRATEGIES FOR REPARTITIONING A DISK

There are several different ways to repartition a disk. This section discusses the following possible approaches:

- Unpartitioned free space is available
- An unused partition is available
- Free space in an actively used partition is available

Note that this section discusses the previously mentioned concepts only theoretically and it does not include any procedural steps on how to perform disk repartitioning step-by-step.

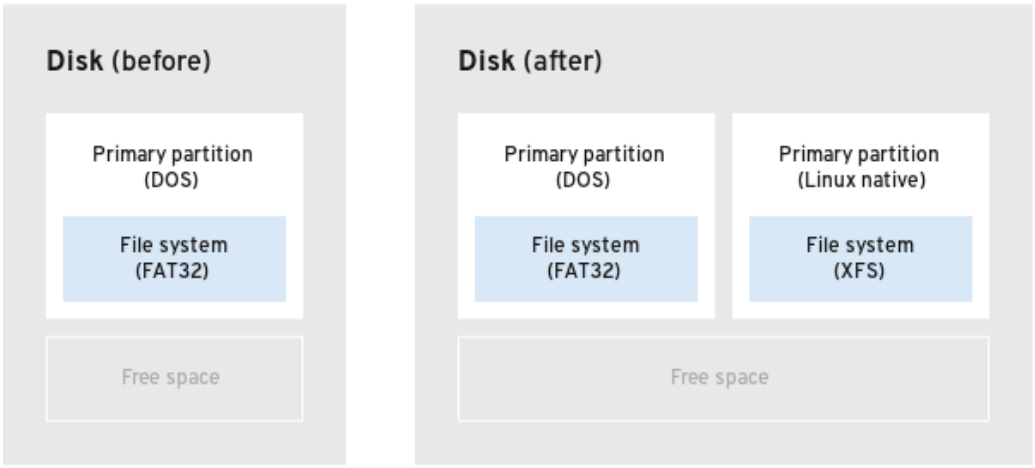
NOTE

The following illustrations are simplified in the interest of clarity and do not reflect the exact partition layout that you encounter when actually installing Red Hat Enterprise Linux.

3.6.1. Using unpartitioned free space

In this situation, the partitions that are already defined do not span the entire hard disk, leaving unallocated space that is not part of any defined partition. The following diagram shows what this might look like:

Figure 3.5. Disk with unpartitioned free space



In the previous example, the first diagram represents a disk with one primary partition and an undefined partition with unallocated space, and the second diagram represents a disk with two defined partitions with allocated space.

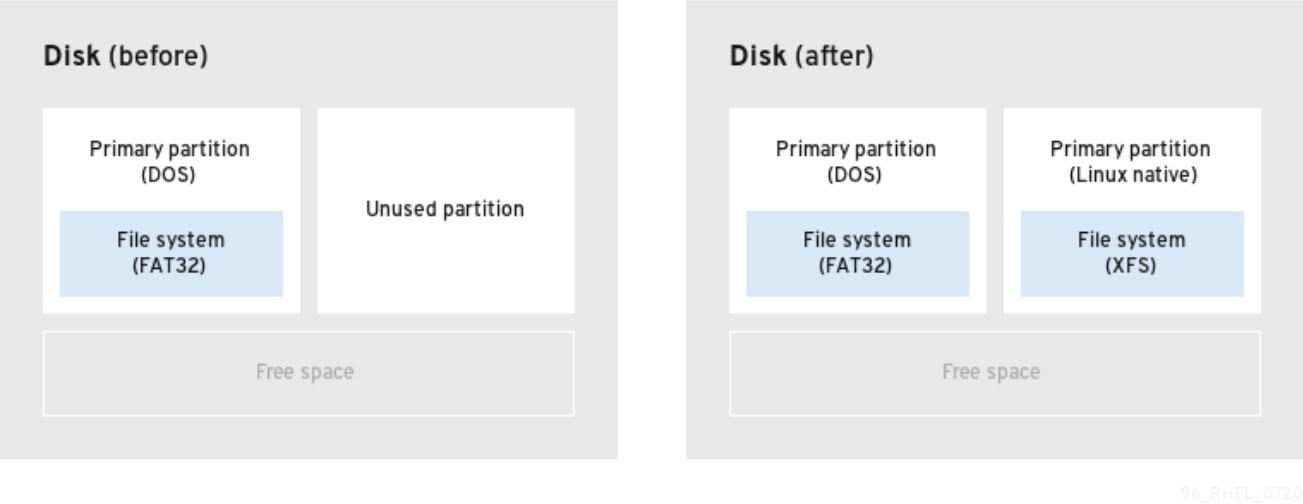
An unused hard disk also falls into this category. The only difference is that *all* the space is not part of any defined partition.

In any case, you can create the necessary partitions from the unused space. This scenario is mostly likely for a new disk. Most preinstalled operating systems are configured to take up all available space on a disk drive.

3.6.2. Using space from an unused partition

In this case, you can have one or more partitions that you no longer use. The following diagram illustrated such a situation.

Figure 3.6. Disk with an unused partition



In the previous example, the first diagram represents a disk with an unused partition, and the second diagram represents reallocating an unused partition for Linux.

In this situation, you can use the space allocated to the unused partition. You must delete the partition and then create the appropriate Linux partition(s) in its place. You can delete the unused partition and manually create new partitions during the installation process.

3.6.3. Using free space from an active partition

This is the most common situation. It is also the hardest to handle, because even if you have enough free space, it is presently allocated to a partition that is already in use. If you purchased a computer with preinstalled software, the hard disk most likely has one massive partition holding the operating system and data.

Aside from adding a new hard drive to your system, you can choose from destructive and non-destructive repartitioning.

3.6.3.1. Destructive repartitioning

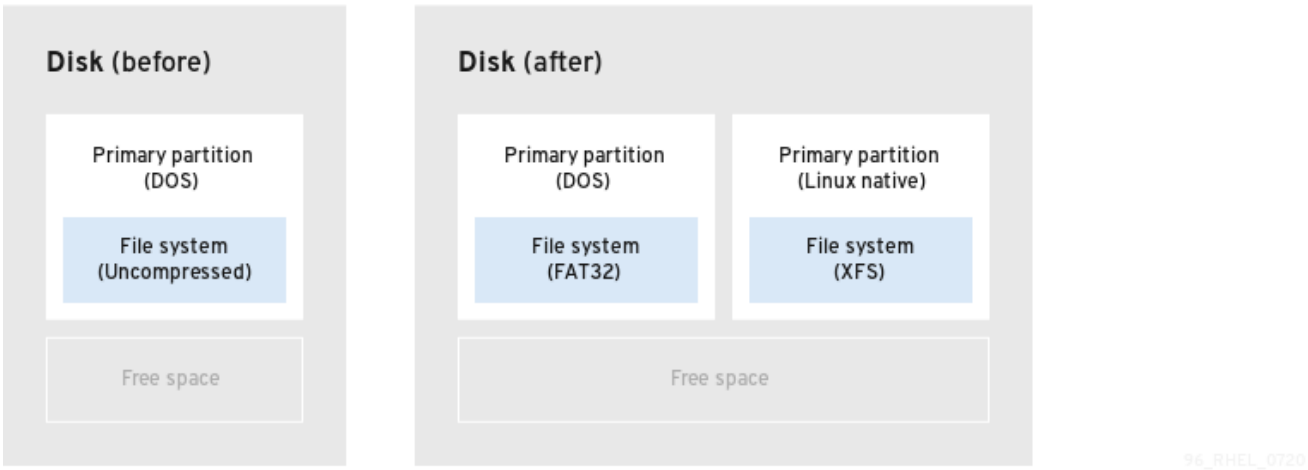
This deletes the partition and creates several smaller ones instead. You must make a complete backup because any data in the original partition is destroyed. Create two backups, use verification (if available in your backup software), and try to read data from the backup *before* deleting the partition.

WARNING

If an operating system was installed on that partition, it must be reinstalled if you want to use that system as well. Be aware that some computers sold with pre-installed operating systems might not include the installation media to reinstall the original operating system. You should check whether this applies to your system *before* you destroy your original partition and its operating system installation.

After creating a smaller partition for your existing operating system, you can reinstall software, restore your data, and start your Red Hat Enterprise Linux installation.

Figure 3.7. Destructive repartitioning action on disk



WARNING

Any data previously present in the original partition is lost.

3.6.3.2. Non-destructive repartitioning

With non-destructive repartitioning you execute a program that makes a big partition smaller without losing any of the files stored in that partition. This method is usually reliable, but can be very time-consuming on large drives.

The non-destructive repartitioning process is straightforward and consist of three steps:

1. Compress and backup existing data

2. Resize the existing partition

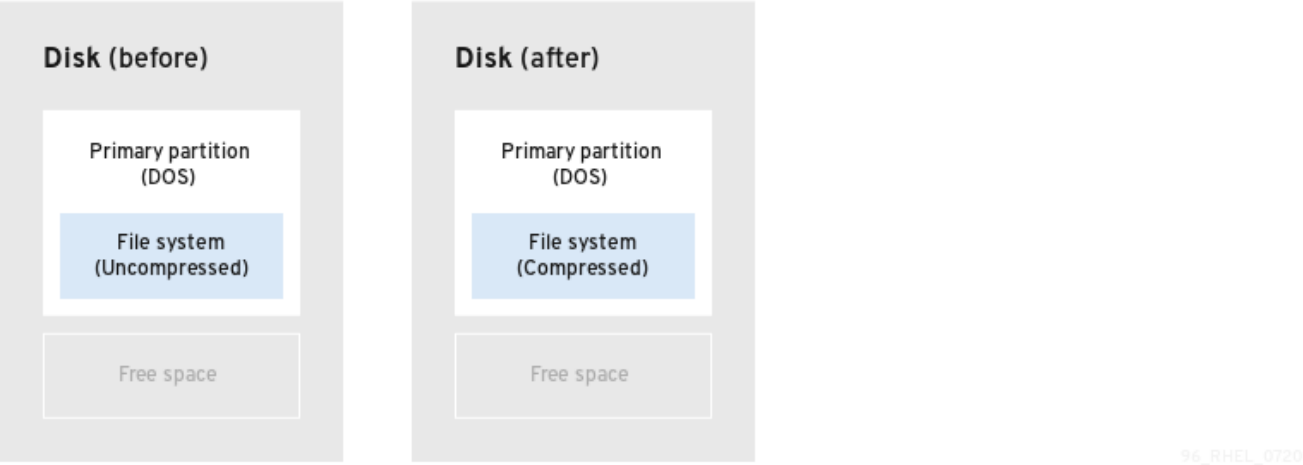
3. Create new partition(s)

Each step is described further in more detail.

3.6.3.2.1. Compressing existing data

The first step is to compress the data in your existing partition. The reason for doing this is to rearrange the data to maximize the available free space at the "end" of the partition.

Figure 3.8. Compression on disk



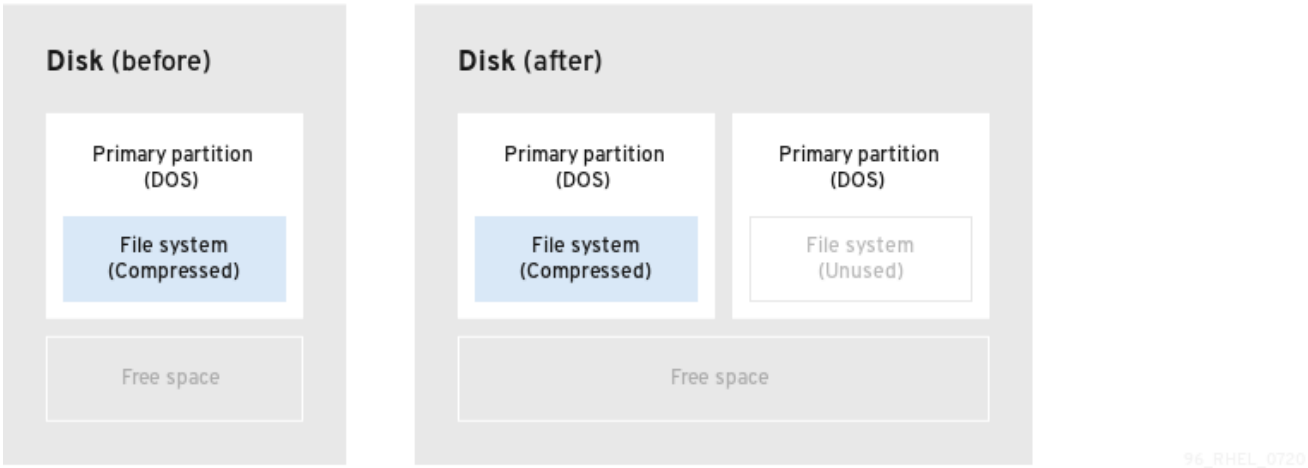
In the previous example, the first diagram represents disk before compression, and the second diagram after compression.

This step is crucial. Without it, the location of the data could prevent the partition from being resized to the desired extent. Note that some data cannot be moved. In this case, it severely restricts the size of your new partitions, and you might be forced to destructively repartition your disk.

3.6.3.2.2. Resizing the existing partition

The following figure shows the actual resizing process. While the actual result of the resizing operation varies, depending on the software used, in most cases the newly freed space is used to create an unformatted partition of the same type as the original partition.

Figure 3.9. Partition resizing on disk



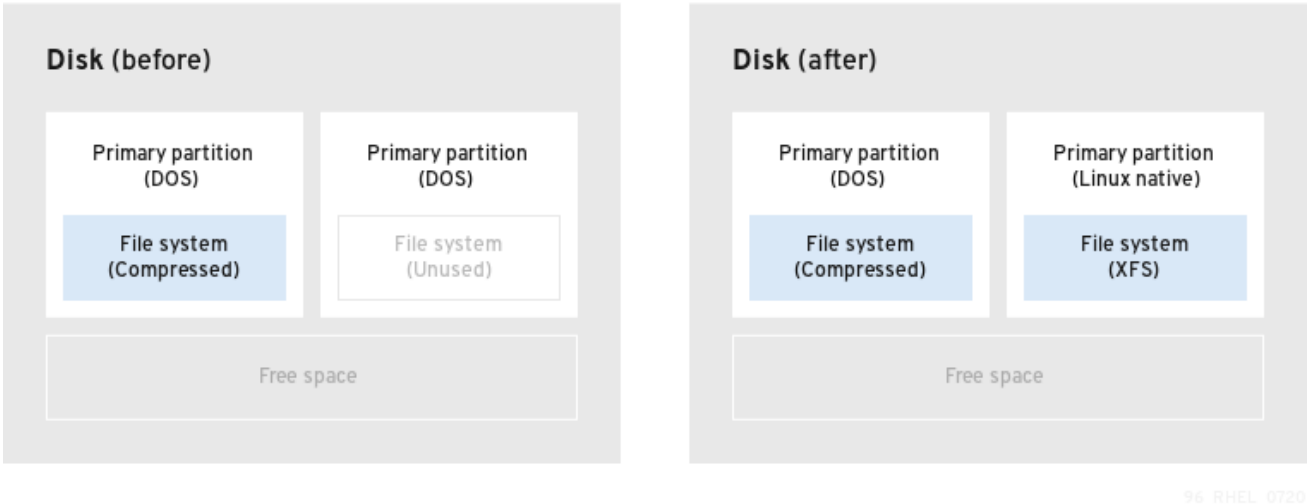
In the previous example, the first diagram represents partition before resizing, and the second diagram after resizing.

It is important to understand what the resizing software you use does with the newly freed space,so that you can take the appropriate steps. In the case illustrated here, it would be best to delete the new DOS partition and create the appropriate Linux partition or partitions.

3.6.3.2.3. Creating new partitions

As mentioned in the previous example, it might or might not be necessary to create new partitions. However, unless your resizing software supports systems with Linux installed, it is likely that you must delete the partition that was created during the resizing process.

Figure 3.10. Disk with final partition configuration



In the previous example, the first diagram represents disk before configuration, and the second diagram after configuration.

CHAPTER 4. 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.

4.1. 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.

4.2. 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.

4.3. 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.

4.3.1. 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.

4.3.2. 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 4.1. 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 4.2. 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.

The Path attribute fails if any part of the hardware path (for example, the PCI ID, target port, or LUN number) changes. The Path attribute is therefore unreliable. However, the Path attribute may be useful in one of the following scenarios:

- You need to identify a disk that you are planning to replace later.
- You plan to install a storage service on a disk in a specific location.

4.4. 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

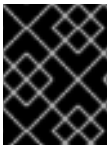
Example 4.3. WWID mappings in a multipath configuration

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.

4.5. 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.
- The device names managed by the **udev** mechanism in /dev/disk/ may change between major releases, requiring you to update the links.

4.6. 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 4.4. 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 4.5. 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 4.6. 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
```

4.7. 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 5. 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.

5.1. 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.

5.2. 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.

5.3. 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.

5.4. 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.

5.5. 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.

Prerequisites

- An NVDIMM device is attached to your system.

5.5.1. 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:

```
# {PackageManagerCommand} install ndctl
```

5.5.2. 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 5.5.1, “Installing ndctl”.

Procedure

1. Reconfigure the selected namespace to sector mode:

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

Example 5.1. 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

5.5.3. 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 5.5.1, "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 5.2. 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

5.6. 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.

Prerequisites

- An NVDIMM device is attached to your system.

5.6.1. 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.

5.6.2. 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:

```
# {PackageManagerCommand} install ndctl
```

5.6.3. 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 5.6.2, “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 5.3. 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

5.6.4. 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 5.6.2, “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=devdax \
      --region=regionN \
      --size=namespace-size
```

Example 5.4. 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

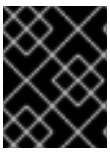
- The **ndctl-create-namespace(1)** man page

5.7. 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.

Prerequisites

- An NVDIMM device is attached to your system.

5.7.1. 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.

5.7.2. 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:

```
# {PackageManagerCommand} install ndctl
```

5.7.3. 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 5.7.2, “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 5.5. 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

5.7.4. 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 5.7.2, “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=fsdax \
      --region=regionN \
      --size=namespace-size
```

Example 5.6. 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

5.7.5. 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 3.3, "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, because they are incompatible with the dax mount option. Additionally, in order to increase the likelihood of large page mappings, set the stripe unit and stripe width.

   ```
   # mkfs.xfs -m reflink=0 -d su=2m,sw=1 fsdax-partition-or-device
   ```

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

   ```
   # mount -o dax 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

5.8. TROUBLESHOOTING NVDIMM PERSISTENT MEMORY

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

Prerequisites

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

5.8.1. Installing ndctl

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

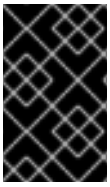
Procedure

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

```
# {PackageManagerCommand} install ndctl
```

5.8.2. 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 5.8.3, “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

5.8.3. 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 5.7. 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 5.8. 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 5.9. 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 5.8.2, "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 5.10. 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 6. DISCARDING UNUSED BLOCKS

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

6.1. 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.

6.2. 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.

6.3. 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

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

6.4. 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

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

6.5. 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 7. CONFIGURING AN ISCSI TARGET

Red Hat Enterprise Linux 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.

7.1. INSTALLING TARGETCLI

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

Procedure

1. Install the **targetcli** tool:

```
# {PackageManagerCommand} 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
```

Verification

- 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

- **targetcli(8)** man page

7.2. 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 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

- **targetcli(8)** man page

7.3. 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 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 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 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 Creating a Memory Copy RAM disk storage object .

Additional resources

- **targetcli(8)** man page

7.4. 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 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
```

Verification

- Verify the created **fileio** storage object:

```
/backstores> ls
```

Additional resources

- **targetcli(8)** man page

7.5. 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 such as, HDDs, SSDs, CDs, and DVDs, and logical devices such as, software or hardware RAID volumes, or LVM volumes.

Prerequisites

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

Procedure

1. Navigate to the backstores directory:

```
/> backstores/
```

2. Create a **block** backstore:

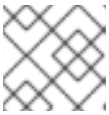
```
/> backstores/block create name=block_backend dev=/dev/sdb

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

Verification

- Verify the created **block** storage object:

```
/backstores> ls
```



NOTE

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

Additional resources

- **targetcli(8)** man page

7.6. 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 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
```

Verification

- Verify the created **pscsi** storage object:

```
/backstores> ls
```

Additional resources

- **targetcli(8)** man page

7.7. 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 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.
```

Verification

- Verify the created **ramdisk** storage object:

```
/backstores> ls
```

Additional resources

- **targetcli(8)** man page

7.8. 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 Installing targetcli.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Creating an iSCSI target.

Procedure

1. Navigate to the TPG directory:

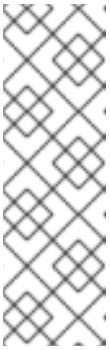
```
/iscsi> iqn.2006-04.example:444/tpg1/
```

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, use the following command:

```
/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
```

Verification

- 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

- **targetcli(8)** man page

7.9. 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 Installing targetcli.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Creating an iSCSI target.
- Created storage objects. For more information, see 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 link:Creating a read-only iSCSI LUN .

3. Configure ACLs. For more information, see Creating an iSCSI ACL .

Additional resources

- **targetcli(8)** man page

7.10. 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 Installing targetcli.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see Creating an iSCSI target.
- Created storage objects. For more information, see 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.

4. Configure ACLs. For more information, see Creating an iSCSI ACL .

Additional resources

- **targetcli(8)** man page

7.11. 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 Installing targetcli.
- An iSCSI target associated with a Target Portal Groups (TPG). For more information, see 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 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
```

Verification

- 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

- **targetcli(8)** man page

7.12. SETTING UP THE CHALLENGE-HANDSHAKE AUTHENTICATION PROTOCOL FOR THE TARGET

By using the **Challenge-Handshake Authentication Protocol (CHAP)**, users can 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 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

- **targetcli(8)** man page

7.13. 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 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
```

Verification

- View the changes:

```
/> iscsi/ ls
```

Additional resources

- **targetcli(8)** man page

CHAPTER 8. CONFIGURING AN ISCSI INITIATOR

An iSCSI initiator forms a session to connect to the 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.

Execute the **systemctl start iscsid.service** command as root to force the **iscsid** daemon to run and iSCSI kernel modules to load.

8.1. CREATING AN ISCSI INITIATOR

This section describes how to create an iSCSI initiator.

Prerequisites

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

Procedure

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

```
# {PackageManagerCommand} 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 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 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

- **targetcli(8)** and **iscsiadm(8)** man pages

8.2. SETTING UP THE CHALLENGE-HANDSHAKE AUTHENTICATION PROTOCOL FOR THE INITIATOR

By using the **Challenge-Handshake Authentication Protocol (CHAP)**, users can 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 Creating an iSCSI initiator .
- Set the **CHAP** for the target. For more information, see 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

- **iscsiadm(8)** man page

8.3. 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.

Execute the **systemctl start iscsid.service** command as root to force the **iscsid** daemon to run and iSCSI kernel modules to load.

Procedure

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

```
{PackageManagerCommand} install iscsi-initiator-utils
```

2. 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

- **iscsiadm(8)** man page

8.4. DM MULTIPATH OVERRIDES OF THE DEVICE TIMEOUT

The **recovery_tmo sysfs** option controls the timeout for a particular iSCSI device. The following options globally override **recovery_tmo** values:

- The **replacement_timeout** configuration option globally overrides the **recovery_tmo** value for all iSCSI devices.

- For all iSCSI devices that are managed by DM Multipath, the **fast_io_fail_tmo** option in DM Multipath globally overrides the **recovery_tmo** value.
  The **fast_io_fail_tmo** option in DM Multipath also overrides the **fast_io_fail_tmo** option in Fibre Channel devices.

The DM Multipath **fast_io_fail_tmo** option takes precedence over **replacement_timeout**. Red Hat does not recommend using **replacement_timeout** to override **recovery_tmo** in devices managed by DM Multipath because DM Multipath always resets **recovery_tmo** when the **multipathd** service reloads.

CHAPTER 9. USING FIBRE CHANNEL DEVICES

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

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

9.1. RESIZING FIBRE CHANNEL LOGICAL UNITS

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

Procedure

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

```
multipath -ll
```

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

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

Additional resources

- **multipath(8)** man page

9.2. 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

- Determine the state of a remote port:

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

  This command returns 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 **sd_N_** devices are removed. Typically, the Fibre Channel class will leave the device as is, that is /**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*** device will be restored only if no additional change on in-storage box configuration of LUNs is made.

Additional resources

- **multipath.conf(5)** man page
- Recommended tuning at scsi,multipath and at application layer while configuring Oracle RAC cluster Knowledgebase article

9.3. FIBRE CHANNEL CONFIGURATION FILES

The following is the list of configuration files in the /**sys**/**class**/ directory that provide the user-space API to Fibre Channel.

The items use the following variables:

H

Host number

B

Bus number

T

Target

L

Logical unit (LUNs)

R

Remote port number

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 configuration in /sys/class/fc_transport/target*H*:*B*:*T*/

port_id

24-bit port ID/address

node_name

64-bit node name

port_name

64-bit port name

Remote port configuration in /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 configuration in /sys/class/fc_host/host*H*/

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

9.4. DM MULTIPATH OVERRIDES OF THE DEVICE TIMEOUT

The **recovery_tmo sysfs** option controls the timeout for a particular iSCSI device. The following options globally override **recovery_tmo** values:

- The **replacement_timeout** configuration option globally overrides the **recovery_tmo** value for all iSCSI devices.
- For all iSCSI devices that are managed by DM Multipath, the **fast_io_fail_tmo** option in DM Multipath globally overrides the **recovery_tmo** value.
  The **fast_io_fail_tmo** option in DM Multipath also overrides the **fast_io_fail_tmo** option in Fibre Channel devices.

The DM Multipath **fast_io_fail_tmo** option takes precedence over **replacement_timeout**. Red Hat does not recommend using **replacement_timeout** to override **recovery_tmo** in devices managed by DM Multipath because DM Multipath always resets **recovery_tmo** when the **multipathd** service reloads.

CHAPTER 10. CONFIGURING FIBRE CHANNEL OVER ETHERNET

Based on the IEEE T11 FC-BB-5 standard, Fibre Channel over Ethernet (FCoE) is a protocol to transmit Fibre Channel frames over Ethernet networks. Typically, data centers have a dedicated LAN and Storage Area Network (SAN) that are separated from each other with their own specific configuration. FCoE combines these networks into a single and converged network structure. Benefits of FCoE are, for example, lower hardware and energy costs.

10.1. USING HARDWARE FCOE HBAS IN RHEL

In Red Hat Enterprise Linux you can use hardware FCoE Host Bus Adapter (HBA) supported by the following drivers:

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

If you use such a HBA, you configure the FCoE settings in the setup of the HBA. For details, see the documentation of the adapter.

After you configured the HBA in its setup, the exported Logical Unit Numbers (LUN) from the Storage Area Network (SAN) are automatically available to RHEL as **/dev/sd*** devices. You can use these devices similar to local storage devices.

10.2. SETTING UP A SOFTWARE FCOE DEVICE

A software FCoE device enables you to access Logical Unit Numbers (LUN) over FCoE using an Ethernet adapter that partially supports FCoE offload.

IMPORTANT

RHEL does not support software FCoE devices that require the **fcoe.ko** kernel module.

After you complete this procedure, the exported LUNs from the Storage Area Network (SAN) are automatically available to RHEL as **/dev/sd*** devices. You can use these devices similar to local storage devices.

Prerequisites

- The Host Bus Adapter (HBA) uses the **qedf**, **bnx2fc**, or **fnic** driver and does not require the **fcoe.ko** kernel module.
- The SAN uses a VLAN to separate the storage traffic from normal Ethernet traffic.
- The network switch has been configured to support the VLAN.
- The HBA of the server is configured in its BIOS. For details, see the documentation of your HBA.
- The HBA is connected to the network and the link is up.

Procedure

1. Install the **fcoe-utils** package:

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

2. Copy the **/etc/fcoe/cfg-ethx** template file to **/etc/fcoe/cfg-*interface_name***. For example, if you want to configure the **enp1s0** interface to use FCoE, enter:

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

3. Enable and start the **fcoe** service:

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

4. Discover the FCoE VLAN ID, start the initiator, and create a network device for the discovered VLAN:

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

5. Optional: To display details about the discovered targets, the LUNs, and the devices associated with the LUNs, enter:

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

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

This example shows that LUN 0 from the SAN has been attached to the host as the **/dev/sdb** device.

Verification steps

- Use the **fcoeadm -i** command to display information about all active FCoE interfaces:

```
# fcoeadm -i
Description:      BCM57840 NetXtreme II 10 Gigabit Ethernet
```

```
Revision:         11
Manufacturer:     Broadcom Inc. and subsidiaries
Serial Number:    000AG703A9B7

Driver:           bnx2x Unknown
Number of Ports:  1

    Symbolic Name:     bnx2fc (QLogic BCM57840) v2.12.13 over enp1s0.200
    OS Device Name:    host11
    Node Name:         0x2000000af70ae935
    Port Name:         0x2001000af70ae935
    Fabric Name:       0x20c8002a6aa7e701
    Speed:             10 Gbit
    Supported Speed:   1 Gbit, 10 Gbit
    MaxFrameSize:      2048 bytes
    FC-ID (Port ID):   0xba02c0
    State:             Online
```

Additional resources

- **fcoeadm(8)** man page
- **/usr/share/doc/fcoe-utils/README**

10.3. ADDITIONAL RESOURCES

- Using Fibre Channel devices

CHAPTER 11. CONFIGURING MAXIMUM TIME FOR STORAGE ERROR RECOVERY WITH EH_DEADLINE

You can configure the maximum allowed time to recover failed SCSI devices. This configuration guarantees an I/O response time even when storage hardware becomes unresponsive due to a failure.

11.1. THE EH_DEADLINE PARAMETER

The SCSI error handling (EH) mechanism attempts to perform error recovery on failed SCSI devices. The SCSI host object **eh_deadline** parameter enables you to configure the maximum amount of time for the recovery. After the configured time expires, SCSI EH stops and resets the entire host bus adapter (HBA).

Using **eh_deadline** can reduce the time:

- to shut off a failed path,
- to switch a path, or
- to disable a RAID slice.

WARNING

When **eh_deadline** expires, SCSI EH resets the HBA, which affects all target paths on that HBA, not only the failing one. If some of the redundant paths are not available for other reasons, I/O errors might occur. Enable **eh_deadline** only if you have a fully redundant multipath configuration on all targets.

Scenarios when eh_deadline is useful

In most scenarios, you do not need to enable **eh_deadline**. Using **eh_deadline** can be useful in certain specific scenarios, for example if a link loss occurs between a Fibre Channel (FC) switch and a target port, and the HBA does not receive Registered State Change Notifications (RSCNs). In such a case, I/O requests and error recovery commands all time out rather than encounter an error. Setting **eh_deadline** in this environment puts an upper limit on the recovery time. That enables the failed I/O to be retried on another available path by DM Multipath.

Under the following conditions, the **eh_deadline** functionality provides no additional benefit, because the I/O and error recovery commands fail immediately, which allows DM Multipath to retry:

- If RSCNs are enabled
- If the HBA does not register the link becoming unavailable

Possible values

The value of the **eh_deadline** is specified in seconds.

The default setting is **off**, which disables the time limit and allows all of the error recovery to take place.

11.2. SETTING THE EH_DEADLINE PARAMETER

This procedure configures the value of the **eh_deadline** parameter to limit the maximum SCSI recovery time.

Procedure

- You can configure **eh_deadline** using either of the following methods:

  sysfs

  Write the number of seconds into the **/sys/class/scsi_host/host*/eh_deadline** files.

  Kernel parameter

  Set a default value for all SCSI HBAs using the **scsi_mod.eh_deadline** kernel parameter.

Additional resources

- How to set eh_deadline and eh_timeout persistently, using a udev rule

CHAPTER 12. GETTING STARTED WITH SWAP

This section describes swap space, and how to add and remove it.

12.1. OVERVIEW OF SWAP SPACE

Swap space in Linux is used when the amount of physical memory (RAM) is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space. While swap space can help machines with a small amount of RAM, it should not be considered a replacement for more RAM.

Swap space is located on hard drives, which have a slower access time than physical memory. Swap space can be a dedicated swap partition (recommended), a swap file, or a combination of swap partitions and swap files.

In years past, the recommended amount of swap space increased linearly with the amount of RAM in the system. However, modern systems often include hundreds of gigabytes of RAM. As a consequence, recommended swap space is considered a function of system memory workload, not system memory.

Adding swap space

The following are the different ways to add a swap space:

- Extending swap on an LVM2 logical volume
- Creating an LVM2 logical volume for swap
- Creating a swap file
  For example, you may upgrade the amount of RAM in your system from 1 GB to 2 GB, but there is only 2 GB of swap space. It might be advantageous to increase the amount of swap space to 4 GB if you perform memory-intense operations or run applications that require a large amount of memory.

Removing swap space

The following are the different ways to remove a swap space:

- Reducing swap on an LVM2 logical volume
- Removing an LVM2 logical volume for swap
- Removing a swap file
  For example, you have downgraded the amount of RAM in your system from 1 GB to 512 MB, but there is 2 GB of swap space still assigned. It might be advantageous to reduce the amount of swap space to 1 GB, since the larger 2 GB could be wasting disk space.

12.2. RECOMMENDED SYSTEM SWAP SPACE

This section describes the recommended size of a swap partition depending on the amount of RAM in your system and whether you want sufficient memory for your system to hibernate. The recommended swap partition size is established automatically during installation. To allow for hibernation, however, you need to edit the swap space in the custom partitioning stage.

The following recommendation are especially important on systems with low memory such as 1 GB and less. Failure to allocate sufficient swap space on these systems can cause issues such as instability or even render the installed system unbootable.

Table 12.1. Recommended swap space

Amount of RAM in the system	Recommended swap space	Recommended swap space if allowing for hibernation
⩽ 2 GB	2 times the amount of RAM	3 times the amount of RAM
> 2 GB – 8 GB	Equal to the amount of RAM	2 times the amount of RAM
> 8 GB – 64 GB	At least 4 GB	1.5 times the amount of RAM
> 64 GB	At least 4 GB	Hibernation not recommended

At the border between each range listed in this table, for example a system with 2 GB, 8 GB, or 64 GB of system RAM, discretion can be exercised with regard to chosen swap space and hibernation support. If your system resources allow for it, increasing the swap space may lead to better performance.

Note that distributing swap space over multiple storage devices also improves swap space performance, particularly on systems with fast drives, controllers, and interfaces.

IMPORTANT

File systems and LVM2 volumes assigned as swap space *should not* be in use when being modified. Any attempts to modify swap fail if a system process or the kernel is using swap space. Use the **free** and **cat /proc/swaps** commands to verify how much and where swap is in use.

Resizing swap space requires temporarily removing the swap space from the system. This can be problematic if running applications rely on the additional swap space and might run into low-memory situations. Preferably, perform swap resizing from rescue mode, see Debug boot options in the *Performing an advanced RHEL installation* . When prompted to mount the file system, select **Skip**.

12.3. EXTENDING SWAP ON AN LVM2 LOGICAL VOLUME

This procedure describes how to extend swap space on an existing LVM2 logical volume. Assuming */dev/VolGroup00/LogVol01* is the volume you want to extend by *2 GB*.

Prerequisites

- You have sufficient disk space.

Procedure

1. Disable swapping for the associated logical volume:

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

2. Resize the LVM2 logical volume by *2 GB*:

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

3. Format the new swap space:

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

4. Enable the extended logical volume:

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

Verification

- To test if the swap logical volume was successfully extended and activated, inspect active swap space by using the following command:

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

12.4. CREATING AN LVM2 LOGICAL VOLUME FOR SWAP

This procedure describes how to create an LVM2 logical volume for swap. Assuming */dev/VolGroup00/LogVol02* is the swap volume you want to add.

Prerequisites

- You have sufficient disk space.

Procedure

1. Create the LVM2 logical volume of size *2 GB*:

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

2. Format the new swap space:

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

3. Add the following entry to the **/etc/fstab** file:

```
/dev/VolGroup00/LogVol02 swap swap defaults 0 0
```

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

```
# systemctl daemon-reload
```

5. Activate swap on the logical volume:

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

Verification

- To test if the swap logical volume was successfully created and activated, inspect active swap space by using the following command:

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

12.5. CREATING A SWAP FILE

This procedure describes how to create a swap file.

Prerequisites

- You have sufficient disk space.

Procedure

1. Determine the size of the new swap file in megabytes and multiply by 1024 to determine the number of blocks. For example, the block size of a 64 MB swap file is 65536.
2. Create an empty file:

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

   Replace *65536* with the value equal to the desired block size.
3. Set up the swap file with the command:

   ```
   # mkswap /swapfile
   ```
4. Change the security of the swap file so it is not world readable.

   ```
   # chmod 0600 /swapfile
   ```
5. Edit the **/etc/fstab** file with the following entries to enable the swap file at boot time:

   ```
   /swapfile swap swap defaults 0 0
   ```

   The next time the system boots, it activates the new swap file.
6. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

   ```
   # systemctl daemon-reload
   ```
7. Activate the swap file immediately:

   ```
   # swapon /swapfile
   ```

Verification

- To test if the new swap file was successfully created and activated, inspect active swap space by using the following command:

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

12.6. REDUCING SWAP ON AN LVM2 LOGICAL VOLUME

This procedure describes how to reduce swap on an LVM2 logical volume. Assuming */dev/VolGroup00/LogVol01* is the volume you want to reduce.

Procedure

1. Disable swapping for the associated logical volume:

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

2. Reduce the LVM2 logical volume by 512 MB:

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

3. Format the new swap space:

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

4. Activate swap on the logical volume:

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

Verification

- To test if the swap logical volume was successfully reduced, inspect active swap space by using the following command:

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

12.7. REMOVING AN LVM2 LOGICAL VOLUME FOR SWAP

This procedure describes how to remove an LVM2 logical volume for swap. Assuming */dev/VolGroup00/LogVol02* is the swap volume you want to remove.

Procedure

1. Disable swapping for the associated logical volume:

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

2. Remove the LVM2 logical volume:

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

3. Remove the following associated entry from the **/etc/fstab** file:

```
/dev/VolGroup00/LogVol02 swap swap defaults 0 0
```

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

```
# systemctl daemon-reload
```

Verification

- To test if the logical volume was successfully removed, inspect active swap space by using the following command:

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

12.8. REMOVING A SWAP FILE

This procedure describes how to remove a swap file.

Procedure

1. At a shell prompt, execute the following command to disable the swap file, where **/swapfile** is the swap file:

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

2. Remove its entry from the **/etc/fstab** file accordingly.
3. Regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

4. Remove the actual file:

```
# rm /swapfile
```

CHAPTER 13. MANAGING SYSTEM UPGRADES WITH SNAPSHOTS

As a system administrator, you can perform rollback-capable upgrades of Red Hat Enterprise Linux systems using the **Boom** boot manager, the **Leapp** utility, and OS modernization framework.

The procedures mentioned in this user story have following limitations:

- It does not work on multiple file systems in your system tree, for example, a separate /**var** or /**usr** partition.
- It does not work for the RHUI systems. Instead of using the **Boom** utility, consider creating snapshots of your VMs.
- Currently, this user story covers only on premises systems with Legacy BIOS and the Intel architecture. You can use it only with Red Hat Enterprise Linux 7 systems that use the BIOS for booting.

13.1. OVERVIEW OF THE BOOM PROCESS

Using **Boom**, you can create boot entries, which can then be accessed and selected from the GRUB 2 boot loader menu. By creating boot entries, the process of preparing for a rollback capable upgrade is now simplified.

The following are the different boot entries, which are part of the upgrade and rollback process:

Upgrade boot entry

Boots the **Leapp** upgrade environment. Use the **leapp** utility to create and manage this boot entry. This boot entry is automatically removed in the course of the **leapp** upgrade.

Red Hat Enterprise Linux 8 boot entry

Boots the upgraded system environment. Use the **leapp** utility to create this boot entry after a successful upgrade process.

Snapshot boot entry

Boots the snapshot of the original system and can be used to review and test the previous system state following a successful or unsuccessful upgrade attempt. Before upgrading the system, use the **boom** command to create this boot entry.

Rollback boot entry

Boots the original system environment and rolls back any upgrade to the previous system state. Use **boom** command to create this boot entry when initiating a rollback of the upgrade procedure.

Rollback-capable upgrades are done using the following process without editing any configuration files:

1. Create a snapshot or copy of the root file system.
2. Use the **boom** command to create a boot entry for the current (older) environment.
3. Upgrade your Red Hat Enterprise Linux system.
4. Reboot the system, and select the version that you want to use.

The Red Hat Enterprise Linux 8, snapshot, and rollback entries should be cleaned up at the end of the procedure depending on the outcome of the update process:

- If you want to keep the updated Red Hat Enterprise Linux 8 system, remove the created snapshot and rollback entries using the **boom** command, and remove snapshot logical volumes with the **lvremove** command. For more information, see Section 13.4, "Deleting the snapshot".
- If you want to rollback to the original system state, merge the snapshot and rollback boot entry, and after rebooting the system remove the unused snapshot and rollback boot entries. For more information, see Section 13.5, "Creating rollback boot entry".

Additional resources

- The **boom** man page.

13.2. UPGRADING TO ANOTHER VERSION USING BOOM

In addition to **Boom**, the following Red Hat Enterprise Linux components are used in this upgrade process:

- Logical volume manager (LVM)
- GRUB 2 boot loader
- **Leapp** upgrade tool

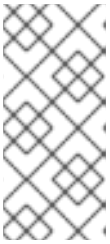
This procedure describes how to upgrade from Red Hat Enterprise Linux 7 to Red Hat Enterprise Linux 8 using the **boom** command.

Prerequisites

- Install the **boom** package:

```
# {PackageManagerCommand} install lvm2-python-boom
```

Ensure that the **lvm2-python-boom** package's version is at least **boom-0.9** (ideally **boom-1.2**).



NOTE

If you want to install **boom** package on Red Hat Enterprise Linux 8, execute the following command:

```
# {PackageManagerCommand} install boom-boot
```

- Sufficient space must be available for the snapshot. Use the following commands to find the free space on the volume groups and logical volumes:

```
# vgs
VG  #PV  #LV  #SN  Attr  VSize    VFree
rhel 4 2 0 wz--n- 103.89g 29.99g

# lvs
LV    VG    Attr     LSize  Pool Origin Data% Meta% Move Log Cpy%Sync Convert
root rhel -wi-ao--- 68.88g
swap rhel -wi-ao--- 5.98g
```

Here, *rhel* is the system volume group, and *root* and *swap* are the system logical volumes.

- Find all the mounted logical volumes:

```
# mount | grep rhel

/dev/mapper/rhel-root on / type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

NOTE

If more than one entry is present and the mount points of the additional entries include '/usr' or '/var', the mentioned steps cannot be followed without executing additional steps that are beyond the scope of this user story.

- The **Leapp** package is installed and the software repositories are enabled. For more information, see the Preparing a RHEL 7 system for the upgrade section, to download the required packages for the upgrade.

Procedure

1. Create a snapshot of your *root* logical volume:
   - If your root file system uses thin provisioning, create a thin snapshot:
     While creating a thin snapshot, do not define the snapshot size. Snapshot is allocated from the thin pool.

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

     Here:
     - **-s** is used to create the snapshot
     - **rhel/root** is the file system being copied in the logical volume
     - **-n** *root_snapshot_before_changes* is the name of the snapshot
   - If your root file system uses thick provisioning, create a thick snapshot:
     While creating a thick snapshot, define the snapshot size that is able to hold all the changes during the upgrade.

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

     Here:
     - **-s** is used to create the snapshot
     - **rhel/root** is the file system being copied
     - **-n** *root_snapshot_before_changes* is the name of the snapshot
     - **-L** *25g* is the snapshot size. This snapshot must be able to hold all the changes during the upgrade

IMPORTANT

After creating the snapshot, any additional system changes are not included.

2. Create the profile:

```
# boom profile create --from-host --uname-pattern el7
```

NOTE

If you want to create the **boom** profile on Red Hat Enterprise Linux 8, use the **el8** as the uname-pattern.

3. Create a snapshot boot entry of the original system, using backup copies of the original boot images:

   a. For the **boom-1.2** or later version:

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

   Here:

   - **--title** *Root LV snapshot before changes* is the name of the boot entry, which displays in the list during system startup
   - **--rootlv** is the root logical volume that corresponds to the new boot entry

   b. For the **boom-1.1** or earlier version:

```
# cp /boot/vmlinuz-$(uname r) /boot/vmlinuz$(uname -r).bak

# cp /boot/initramfs-$(uname r).img /boot/initramfs$(uname -r).img.bak

# boom create -title "Root LV snapshot before changes" --rootlv
rhel/root_snapshot_before_changes --linux /boot/vmlinuz$(uname r).bak --initrd
/boot/initramfs$(uname -r).img.bak
```

   If you execute the **boom create** command for the first time, the following message displays:

```
WARNING - Boom configuration not found in grub.cfg

WARNING - Run 'grub2-mkconfig > /boot/grub2/grub.cfg' to enable
```

   To enable Boom in GRUB 2:

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

4. Upgrade to Red Hat Enterprise Linux 8 using the **Leapp** utility:

```
# leapp preupgrade
```

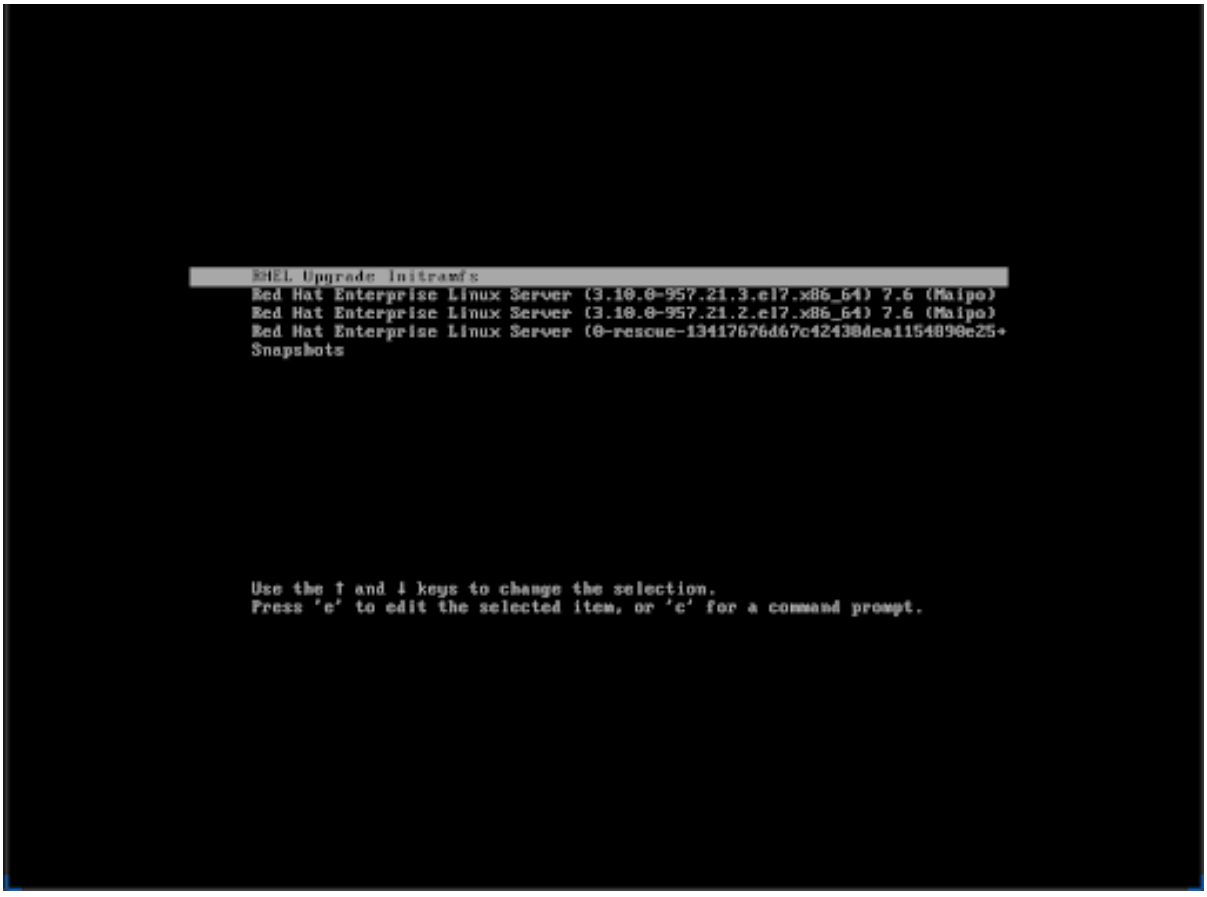
Review and address if any blockers indicated in the **leapp preupgrade** command report.

5. After resolving the blockers identified in the pre-upgrade reports, re-run the upgrade command with the **--reboot** option:

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

This command reboots into the upgraded boot entry created by the **leapp** utility and proceeds to execute the in-place upgrade to Red Hat Enterprise Linux 8. The reboot argument initiates an automatic system restart after the upgrade process.

During reboot, the GRUB 2 screen is displayed:

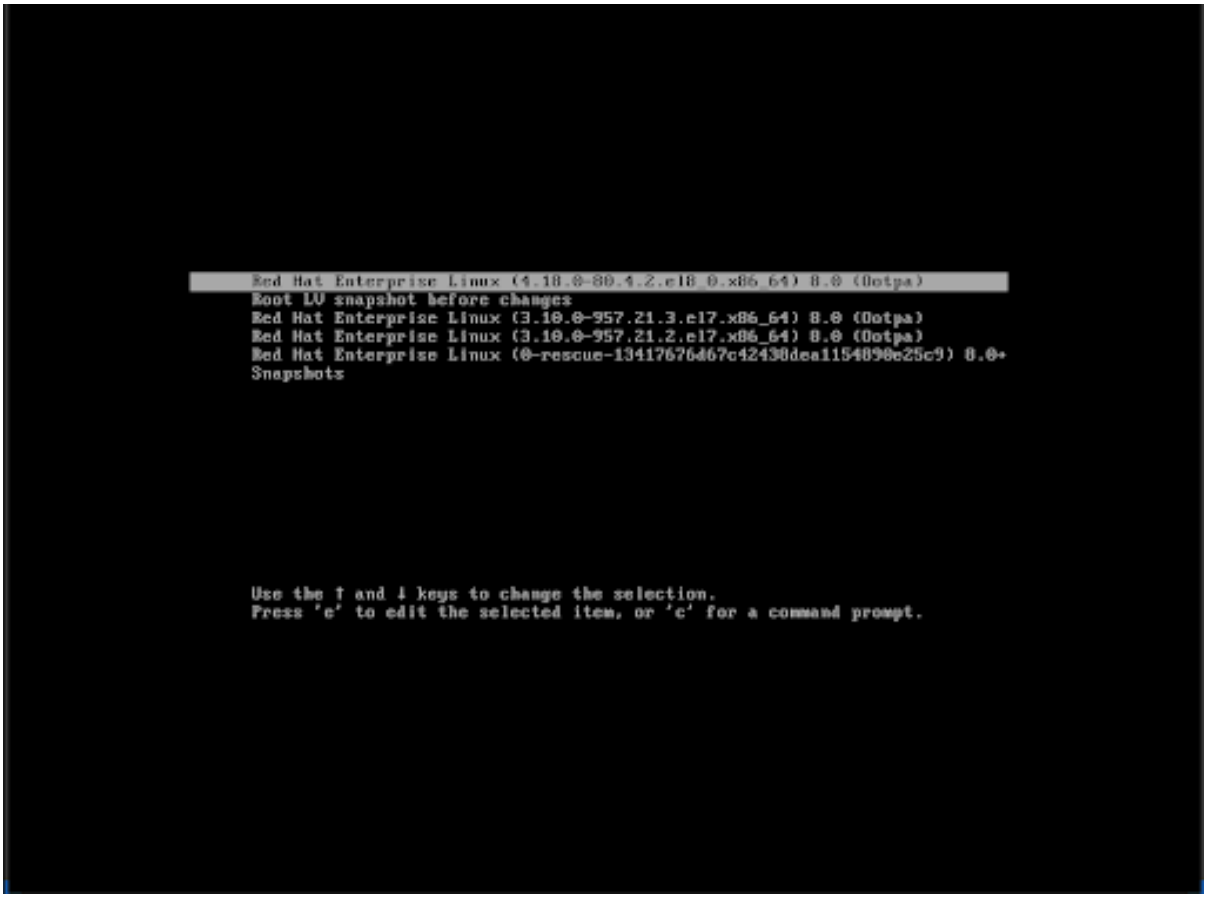


NOTE

If you are on the Red Hat Enterprise Linux 8 system, the **Snapshots** sub-menu from the GRUB2 boot screen is not available.

Verification steps

- Select the **RHEL Upgrade Initramfs** entry and press ENTER. The upgrade continues and new Red Hat Enterprise Linux 8 RPM packages are installed. After the upgrade is complete, the system automatically reboots and the GRUB 2 screen displays the upgraded and the older version of the available system. The upgraded system version is the default selection.



Additionally, the created ***Root LV snapshot before changes*** boot entry is present, which provides an instant access to the system state prior to the upgrade.

Additional resources

- The **boom** man page.
- What is BOOM and how to install it? Knowledgebase article.
- How to create a BOOM boot entry Knowledgebase article.
- Data required by the Leapp utility for an in-place upgrade from RHEL 7 to RHEL 8 Knowledgebase article.

13.3. SWITCHING BETWEEN NEW AND OLD RED HAT ENTERPRISE LINUX VERSIONS

The **Boom** boot manager reduces the risks associated with upgrading a system and also helps to reduce hardware downtime. For example, you can upgrade a Red Hat Enterprise Linux 7 system to Red Hat Enterprise Linux 8, while retaining the original Red Hat Enterprise Linux 7 environment. This ability to switch between environments allows you to:

- Quickly compare both environments in a side-by-side fashion and switch between them with minimal overhead.
- Recover the older file system's content.
- Continue accessing the old system while the upgraded host is running.
- Halt and revert the update process at any time, even while the update itself is running.

This procedure describes how to switch between the new and the old Red Hat Enterprise Linux versions after the upgrade is complete.

Prerequisites

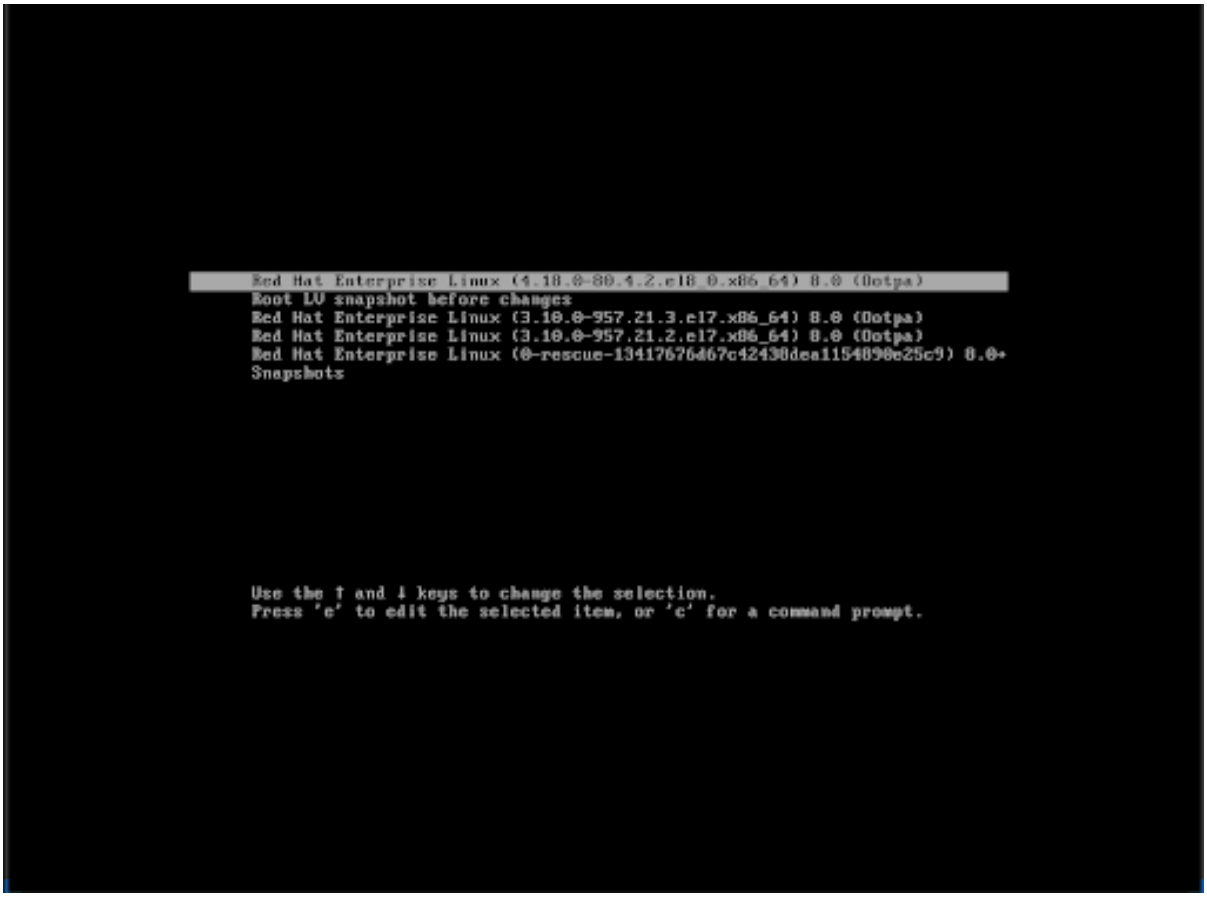
- Upgraded Red Hat Enterprise Linux version. For more information, see Section 13.2, "Upgrading to another version using Boom".

Procedure

1. Reboot the system:

```
# reboot
```

2. Select the desired boot entry from the GRUB 2 boot loader screen.



Verification steps

- Verify that the selected boot volume is displayed:

```
# cat /proc/cmdline

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

Additional resources

- The **boom** man page.

13.4. DELETING THE SNAPSHOT

Snapshot boot entry boots the snapshot of the original system and can be used to review and test the previous system state following a successful or unsuccessful upgrade attempt. This procedure describes steps to delete the snapshot.

Prerequisites

- Upgraded to a new RHEL version. For more information, see Section 13.2, "Upgrading to another version using Boom".

Procedure

1. Boot into Red Hat Enterprise Linux 8 from the GRUB 2 entry. The following output confirms that the new snapshot is selected:

```
# boom list
BootID  Version                 Name                        RootDevice
6d2ec72 3.10.0-957.21.3.el7.x86_64 Red Hat Enterprise Linux Server
/dev/rhel/root_snapshot_before_changes
```

2. Delete the snapshot entry using the **BootID** value:

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

   This deletes the entry from the GRUB 2 menu.

3. Remove the LV snapshot:

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

Additional resources

- The **boom** man page.

13.5. CREATING ROLLBACK BOOT ENTRY

Rollback boot entry boots the original system environment and rolls back any upgrade to the previous system state. Reverting the upgraded and rollback boot entry to the original environment after reviewing it, is now available via the snapshot boot entry.

A rollback boot entry may be prepared either from the upgraded system or from the snapshot environment.

Prerequisites

- Upgraded to a new RHEL version. For more information, see Section 13.2, "Upgrading to another version using Boom".

Procedure

1. Merge the snapshot:

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

2. Create a rollback boot entry for the merged snapshot:

   a. For the **boom-1.2** or later version:

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

   b. For the **boom-1.1** or earlier version:

```
boom create --title "RHEL Rollback" --rootlv rhel/root --linux /boot/vmlinuz$(uname r).bak
--initrd /boot/initramfs$(uname -r).img.bak
```

3. Optional: Boot rollback environment and restore the system state:

```
# reboot
```

   Once the system is rebooted, select the ***RHEL Rollback*** boot entry using the arrow keys and press **Enter** to boot this entry.

   The system automatically starts the snapshot merge operation once the **root** logical volume is activated.

NOTE

Once the merge operation is started, the snapshot volume is no longer available. After successfully booting the *RHEL Rollback* boot entry, the *Root LV snapshot before changes* boot entry no longer works because it is now merged into the original logical volume. Merging a snapshot logical volume, destroys the snapshot and restores the prior state of the origin volume.

4. Optional: Once the merge operation is completed, remove the unused entries and restore the original boot entry:

   a. Remove the unused Red Hat Enterprise Linux 8 boot entries from the **/boot** file system and refresh the **Grub2** configuration:

```
# rm -f /boot/el8
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

   b. Restore the original Red Hat Enterprise Linux 7 boot entry:

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

5. After successful rollback to the system, delete the **boom** boot entry:

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

Additional resources

- The **boom** man page.

CHAPTER 14. NVME OVER FABRICS USING RDMA

In an NVMe over RDMA (NVMe/RDMA) setup, you configure an NVMe target and an NVMe initiator.

As a system administrator, complete the following tasks to deploy the NVMe/RDMA setup:

- Setting up an NVMe/RDMA target using configfs
- Setting up the NVMe/RDMA target using nvmetcli
- Configuring an NVMe/RDMA client

14.1. OVERVIEW OF NVME OVER FABRIC DEVICES

Non-volatile Memory Express (NVMe) is an interface that allows host software utility to communicate with solid state drives.

Use the following types of fabric transport to configure NVMe over fabric devices:

NVMe over Remote Direct Memory Access (NVMe/RDMA)

For information on how to configure NVMe/RDMA, see NVMe over fabrics using RDMA .

NVMe over Fibre Channel (FC-NVMe)

For information on how to configure FC-NVMe, see NVMe over fabrics using FC .

When using Fibre Channel (FC) and Remote Direct Memory Access (RDMA), the solid-state drive does not have to be local to your system; it can be configured remotely through a FC or RDMA controller.

14.2. SETTING UP AN NVME/RDMA TARGET USING CONFIGFS

Use this procedure to configure an NVMe/RDMA target using **configfs**.

Prerequisites

- Verify that you have a block device to assign to the **nvmet** subsystem.

Procedure

1. Create the **nvmet-rdma** subsystem:

```
# modprobe nvmet-rdma

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

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

   Replace *testnqn* with the subsystem name.

2. Allow any host to connect to this target:

```
# echo 1 > attr_allow_any_host
```

3. Configure a namespace:

```
# mkdir namespaces/10

# cd namespaces/10
```

Replace *10* with the namespace number

4. Set a path to the NVMe device:

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

5. Enable the namespace:

```
# echo 1 > enable
```

6. Create a directory with an NVMe port:

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

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

7. Display the IP address of *mlx5_ib0*:

```
# ip addr show mlx5_ib0

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

8. Set the transport address for the target:

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

9. Set RDMA as the transport type:

```
# echo rdma > addr_trtype

# echo 4420 > addr_trsvcid
```

10. Set the address family for the port:

```
# echo ipv4 > addr_adrfam
```

11. Create a soft link:

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

Verification

- Verify that the NVMe target is listening on the given port and ready for connection requests:

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

Additional resources

- **nvme(1)** man page

14.3. SETTING UP THE NVME/RDMA TARGET USING NVMETCLI

Use the **nvmetcli** utility to edit, view, and start an NVMe target. The **nvmetcli** utility provides a command line and an interactive shell option. Use this procedure to configure the NVMe/RDMA target by **nvmetcli**.

Prerequisites

- Verify that you have a block device to assign to the **nvmet** subsystem.
- Execute the following **nvmetcli** operations as a root user.

Procedure

1. Install the **nvmetcli** package:

```
# {PackageManagerCommand} install nvmetcli
```

2. Download the **rdma.json** file:

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

3. Edit the **rdma.json** file and change the **traddr** value to **172.31.0.202**.
4. Setup the target by loading the NVMe target configuration file:

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

NOTE

If the NVMe target configuration file name is not specified, the **nvmetcli** uses the **/etc/nvmet/config.json** file.

Verification

- Verify that the NVMe target is listening on the given port and ready for connection requests:

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

- Optional: Clear the current NVMe target:

```
# nvmetcli clear
```

Additional resources

- **nvmetcli** and **nvme(1)** man pages

14.4. CONFIGURING AN NVME/RDMA CLIENT

Use this procedure to configure an NVMe/RDMA client using the NVMe management command line interface (**nvme-cli**) tool.

Procedure

1. Install the **nvme-cli** tool:

```
# {PackageManagerCommand} install nvme-cli
```

2. Load the **nvme-rdma** module if it is not loaded:

```
# modprobe nvme-rdma
```

3. Discover available subsystems on the NVMe target:

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

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

4. Connect to the discovered subsystems:

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

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

```
  └─rhel_rdma--virt--03-home 253:2    0 410.8G  0 lvm  /home
nvme0n1

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

Replace *testnqn* with the NVMe subsystem name.

Replace *172.31.0.202* with the target IP address.

Replace *4420* with the port number.

Verification

- List the NVMe devices that are currently connected:

```
# nvme list
```

- Optional: Disconnect from the target:

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

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

Additional resources

- **nvme(1)** man page
- Nvme-cli Github repository

14.5. NEXT STEPS

- Enable multipathing on your FC-NVMe devices. For more information, see Enabling multipathing on NVMe devices.

CHAPTER 15. NVME OVER FABRICS USING FC

The NVMe over Fibre Channel (FC-NVMe) transport is fully supported in initiator mode when used with certain Broadcom Emulex and Marvell Qlogic Fibre Channel adapters. As a system administrator, complete the tasks in the following sections to deploy the FC-NVMe setup:

- Configuring the NVMe initiator for Broadcom adapters
- Configuring the NVMe initiator for QLogic adapters

15.1. OVERVIEW OF NVME OVER FABRIC DEVICES

Non-volatile Memory Express (NVMe) is an interface that allows host software utility to communicate with solid state drives.

Use the following types of fabric transport to configure NVMe over fabric devices:

NVMe over Remote Direct Memory Access (NVMe/RDMA)

For information on how to configure NVMe/RDMA, see NVMe over fabrics using RDMA .

NVMe over Fibre Channel (FC-NVMe)

For information on how to configure FC-NVMe, see NVMe over fabrics using FC .

When using Fibre Channel (FC) and Remote Direct Memory Access (RDMA), the solid-state drive does not have to be local to your system; it can be configured remotely through a FC or RDMA controller.

15.2. CONFIGURING THE NVME INITIATOR FOR BROADCOM ADAPTERS

Use this procedure to configure the NVMe initiator for Broadcom adapters client using the NVMe management command line interface (**nvme-cli**) tool.

Procedure

1. Install the **nvme-cli** tool:

```
# {PackageManagerCommand} install nvme-cli
```

This creates the **hostnqn** file in the **/etc/nvme/** directory. The **hostnqn** file identifies the NVMe host.

To generate a new **hostnqn**, use the following command:

```
# nvme gen-hostnqn
```

2. Find the WWNN and WWPN identifiers of the local and remote ports and use the output to find the subsystem NQN:

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

NVME Initiator Enabled
XRI Dist lpfc0 Total 6144 IO 5894 ELS 250
NVME LPORT lpfc0 WWPN x10000090fae0b5f5 WWNN x20000090fae0b5f5 DID x010f00
```

```
ONLINE
NVME RPORT       WWPN x204700a098cbcac6 WWNN x204600a098cbcac6 DID x01050e
TARGET DISCSRVC ONLINE

NVME Statistics
LS: Xmt 000000000e Cmpl 000000000e Abort 00000000
LS XMIT: Err 00000000  CMPL: xb 00000000 Err 00000000
Total FCP Cmpl 00000000000008ea Issue 00000000000008ec OutIO 0000000000000002
    abort 00000000 noxri 00000000 nondlp 00000000 qdepth 00000000 wqerr 00000000 err
00000000
FCP CMPL: xb 00000000 Err 00000000
```

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

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

Replace *nn-0x204600a098cbcac6:pn-0x204700a098cbcac6* with the **traddr**.

Replace *nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5* with the **host-traddr**.

3. Connect to the NVMe target using the **nvme-cli**:

```
# nvme connect --transport fc \
              --traddr nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 \
              --host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 \
              -n nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
```

Replace *nn-0x204600a098cbcac6:pn-0x204700a098cbcac6* with the **traddr**.

Replace *nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5* with the **host-traddr**.

Replace *nqn.1992-08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1* with the ***subnqn***.

Verification

- List the NVMe devices that are currently connected:

```
# nvme list
Node             SN                   Model                                    Namespace Usage
Format           FW Rev
```

```
---------------- -------------------- ---------------------------------------- --------- ------------------------
- ---------------- --------
/dev/nvme0n1     80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller                   1
107.37  GB / 107.37  GB      4 KiB +  0 B   FFFFFFFF
```

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

Additional resources

- **nvme(1)** man page
- Nvme-cli Github repository

15.3. CONFIGURING THE NVME INITIATOR FOR QLOGIC ADAPTERS

Use this procedure to configure NVMe initiator for Qlogic adapters client using the NVMe management command line interface (**nvme-cli**) tool.

Procedure

1. Install the **nvme-cli** tool:

```
# {PackageManagerCommand} install nvme-cli
```

This creates the **hostnqn** file in the **/etc/nvme/** directory. The **hostnqn** file identifies the NVMe host.

To generate a new **hostnqn**, use the following command:

```
# nvme gen-hostnqn
```

2. Reload the **qla2xxx** module:

```
# rmmod qla2xxx
# modprobe qla2xxx
```

3. Find the WWNN and WWPN identifiers of the local and remote ports:

```
# dmesg |grep traddr

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

Using these **host-traddr** and **traddr** values, find the subsystem NQN:

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

Discovery Log Number of Records 2, Generation counter 49530
```

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

Replace *nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6* with the **traddr**.

Replace *nn-0x20000024ff19bb62:pn-0x21000024ff19bb62* with the **host-traddr**.

4. Connect to the NVMe target using the **nvme-cli** tool:

```
# nvme connect  --transport fc \
        --traddr nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 \
        --host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 \
        -n nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_su
bsystem_468
```

Replace *nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6* with the **traddr**.

Replace *nn-0x20000024ff19bb62:pn-0x21000024ff19bb62* with the **host-traddr**.

Replace *nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsyste*
with the ***subnqn***.

Verification

- List the NVMe devices that are currently connected:

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

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

Additional resources

- **nvme(1)** man page
- Nvme-cli Github repository

15.4. NEXT STEPS

- Enable multipathing on your FC-NVMe devices. For more information, see Enabling multipathing on NVMe devices.

CHAPTER 16. ENABLING MULTIPATHING ON NVME DEVICES

You can multipath NVMe devices that are connected to your system over a fabric transport, such as Fibre Channel (FC). You can select between multiple multipathing solutions.

16.1. NATIVE NVME MULTIPATHING AND DM MULTIPATH

NVMe devices support a native multipathing functionality. When configuring multipathing on NVMe, you can select between the standard DM Multipath framework and the native NVMe multipathing.

Both DM Multipath and native NVMe multipathing support the Asymmetric Namespace Access (ANA) multipathing scheme of NVMe devices. ANA identifies optimized paths between the target and the initiator and improves performance.

When native NVMe multipathing is enabled, it applies globally to all NVMe devices. It can provide higher performance, but does not contain all of the functionality that DM Multipath provides. For example, native NVMe multipathing supports only the **failover** and **round-robin** path selection methods.

Red Hat recommends that you use DM Multipath in Red Hat Enterprise Linux 8 as your default multipathing solution.

16.2. ENABLING NATIVE NVME MULTIPATHING

You can enable multipathing on connected NVMe devices using the native NVMe multipathing solution.

Prerequisites

- The NVMe devices are connected to your system.
  For more information on connecting NVMe over fabric transports, see Overview of NVMe over fabric devices.

Procedure

1. Check if native NVMe multipathing is enabled in the kernel:

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

   The command displays one of the following:

   N
   Native NVMe multipathing is disabled.

   Y
   Native NVMe multipathing is enabled.

2. If native NVMe multipathing is disabled, enable it using one of the following methods:

   - Using a kernel option:

     i. Add the **nvme_core.multipath=Y** option on the kernel command line:

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

     ii. On the 64-bit IBM Z architecture, update the boot menu:

```
# zipl
```

iii. Reboot the system.

- Using a kernel module configuration file:

i. Create the **/etc/modprobe.d/nvme_core.conf** configuration file with the following content:

```
options nvme_core multipath=Y
```

ii. Back up the **initramfs** file system:

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

iii. Rebuild the **initramfs** file system:

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

iv. Reboot the system.

3. Optional: On the running system, change the I/O policy on NVMe devices to distribute the I/O on all available paths:

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

4. Optional: Set the I/O policy persistently using **udev** rules. Create the **/etc/udev/rules.d/71-nvme-io-policy.rules** file with the following content:

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

Verification

1. Check that your system recognizes the NVMe devices:

```
# nvme list

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

2. List all connected NVMe subsystems:

```
# nvme list-subsys

nvme-subsys0 - NQN=testnqn
\
```

```
 +- nvme0 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-
0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme1 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-
0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme2 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
 +- nvme3 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
```

Check the active transport type. For example, **nvme0 fc** indicates that the device is connected over the Fibre Channel transport, and **nvme tcp** indicates that the device is connected over TCP.

3. If you edited the kernel options, check that native NVMe multipathing is enabled on the kernel command line:

```
# cat /proc/cmdline

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

4. Check that DM Multipath reports the NVMe namespaces as, for example, **nvme0c0n1** through **nvme0c3n1**, and *not* as, for example, **nvme0n1** through **nvme3n1**:

```
# multipath -e -ll | grep -i nvme

uuid.8ef20f70-f7d3-4f67-8d84-1bb16b2bfe03 [nvme]:nvme0n1 NVMe,Linux,4.18.0-2
| `- 0:0:1    nvme0c0n1 0:0     n/a   optimized live
| `- 0:1:1    nvme0c1n1 0:0     n/a   optimized live
| `- 0:2:1    nvme0c2n1 0:0     n/a   optimized live
  `- 0:3:1    nvme0c3n1 0:0     n/a   optimized live

uuid.44c782b4-4e72-4d9e-bc39-c7be0a409f22 [nvme]:nvme0n2 NVMe,Linux,4.18.0-2
| `- 0:0:1    nvme0c0n1 0:0     n/a   optimized live
| `- 0:1:1    nvme0c1n1 0:0     n/a   optimized live
| `- 0:2:1    nvme0c2n1 0:0     n/a   optimized live
  `- 0:3:1    nvme0c3n1 0:0     n/a   optimized live
```

5. If you changed the I/O policy, check that **round-robin** is the active I/O policy on NVMe devices:

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

round-robin
```

Additional resources

- Configuring kernel command-line parameters

16.3. ENABLING DM MULTIPATH ON NVME DEVICES

You can enable multipathing on connected NVMe devices using the DM Multipath solution.

Prerequisites

- The NVMe devices are connected to your system.

For more information on connecting NVMe over fabric transports, see Overview of NVMe over fabric devices.

Procedure

1. Check that native NVMe multipathing is disabled:

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

The command displays one of the following:

N
Native NVMe multipathing is disabled.

Y
Native NVMe multipathing is enabled.

2. If native NVMe multipathing is enabled, disable it:

   a. Remove the **nvme_core.multipath=Y** option from the kernel command line:

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

   b. On the 64-bit IBM Z architecture, update the boot menu:

```
# zipl
```

   c. Remove the **options nvme_core multipath=Y** line from the **/etc/modprobe.d/nvme_core.conf** file, if it is present.

   d. Reboot the system.

3. Make sure that DM Multipath is enabled:

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

4. Distribute I/O on all available paths. Add the following content in the **/etc/multipath.conf** file:

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

NOTE

The **/sys/class/nvme-subsystem/nvme-subsys0/iopolicy** configuration file has no effect on the I/O distribution when DM Multipath manages the NVMe devices.

5. Reload the **multipathd** service to apply the configuration changes:

```
# multipath -r
```

6. Back up the **initramfs** file system:

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

7. Rebuild the **initramfs** file system:

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

Verification

1. Check that your system recognizes the NVMe devices:

```
# nvme list

Node             SN                   Model                                    Namespace Usage
Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- -------------------------
- ---------------- --------
/dev/nvme0n1     a34c4f3a0d6f5cec     Linux                                    1         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme0n2     a34c4f3a0d6f5cec     Linux                                    2         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme1n1     a34c4f3a0d6f5cec     Linux                                    1         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme1n2     a34c4f3a0d6f5cec     Linux                                    2         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme2n1     a34c4f3a0d6f5cec     Linux                                    1         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme2n2     a34c4f3a0d6f5cec     Linux                                    2         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme3n1     a34c4f3a0d6f5cec     Linux                                    1         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
/dev/nvme3n2     a34c4f3a0d6f5cec     Linux                                    2         250.06  GB /
250.06  GB    512   B +  0 B   4.18.0-2
```

2. List all connected NVMe subsystems. Check that the command reports them as, for example, **nvme0n1** through **nvme3n2**, and *not* as, for example, **nvme0c0n1** through **nvme0c3n1**:

```
# nvme list-subsys

nvme-subsys0 - NQN=testnqn
\
 +- nvme0 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-
0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme1 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-
0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme2 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
 +- nvme3 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-
0x20000090fac7e1de:pn-0x10000090fac7e1de live
```

```
# multipath -ll
```

```
mpathae (uuid.8ef20f70-f7d3-4f67-8d84-1bb16b2bfe03) dm-36 NVME,Linux
size=233G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=50 status=active
  |- 0:1:1:1  nvme0n1 259:0   active ready running
  |- 1:2:1:1  nvme1n1 259:2   active ready running
  |- 2:3:1:1  nvme2n1 259:4   active ready running
  `- 3:4:1:1  nvme3n1 259:6   active ready running

mpathaf (uuid.44c782b4-4e72-4d9e-bc39-c7be0a409f22) dm-39 NVME,Linux
size=233G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=50 status=active
  |- 0:1:2:2  nvme0n2 259:1   active ready running
  |- 1:2:2:2  nvme1n2 259:3   active ready running
  |- 2:3:2:2  nvme2n2 259:5   active ready running
  `- 3:4:2:2  nvme3n2 259:7   active ready running
```

Additional resources

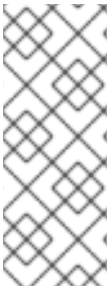
- Configuring kernel command-line parameters
- Setting up DM Multipath

CHAPTER 17. SETTING THE DISK SCHEDULER

The disk scheduler is responsible for ordering the I/O requests submitted to a storage device.

You can configure the scheduler in several different ways:

- Set the scheduler using **TuneD**, as described in Setting the disk scheduler using TuneD
- Set the scheduler using **udev**, as described in Setting the disk scheduler using udev rules
- Temporarily change the scheduler on a running system, as described in Temporarily setting a scheduler for a specific disk



NOTE

In Red Hat Enterprise Linux 8, block devices support only multi-queue scheduling. This enables the block layer performance to scale well with fast solid-state drives (SSDs) and multi-core systems.

The traditional, single-queue schedulers, which were available in Red Hat Enterprise Linux 7 and earlier versions, have been removed.

17.1. AVAILABLE DISK SCHEDULERS

The following multi-queue disk schedulers are supported in Red Hat Enterprise Linux 8:

none

Implements a first-in first-out (FIFO) scheduling algorithm. It merges requests at the generic block layer through a simple last-hit cache.

mq-deadline

Attempts to provide a guaranteed latency for requests from the point at which requests reach the scheduler.
The **mq-deadline** scheduler sorts queued I/O requests into a read or write batch and then schedules them for execution in increasing logical block addressing (LBA) order. By default, read batches take precedence over write batches, because applications are more likely to block on read I/O operations. After **mq-deadline** processes a batch, it checks how long write operations have been starved of processor time and schedules the next read or write batch as appropriate.

This scheduler is suitable for most use cases, but particularly those in which the write operations are mostly asynchronous.

bfq

Targets desktop systems and interactive tasks.
The **bfq** scheduler ensures that a single application is never using all of the bandwidth. In effect, the storage device is always as responsive as if it was idle. In its default configuration, **bfq** focuses on delivering the lowest latency rather than achieving the maximum throughput.

bfq is based on **cfq** code. It does not grant the disk to each process for a fixed time slice but assigns a *budget* measured in number of sectors to the process.

This scheduler is suitable while copying large files and the system does not become unresponsive in this case.

kyber

The scheduler tunes itself to achieve a latency goal by calculating the latencies of every I/O request submitted to the block I/O layer. You can configure the target latencies for read, in the case of cache-misses, and synchronous write requests.
This scheduler is suitable for fast devices, for example NVMe, SSD, or other low latency devices.

17.2. DIFFERENT DISK SCHEDULERS FOR DIFFERENT USE CASES

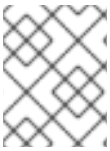
Depending on the task that your system performs, the following disk schedulers are recommended as a baseline prior to any analysis and tuning tasks:

Table 17.1. Disk schedulers for different use cases

Use case	Disk scheduler
Traditional HDD with a SCSI interface	Use **mq-deadline** or **bfq**.
High-performance SSD or a CPU-bound system with fast storage	Use **none**, especially when running enterprise applications. Alternatively, use **kyber**.
Desktop or interactive tasks	Use **bfq**.
Virtual guest	Use **mq-deadline**. With a host bus adapter (HBA) driver that is multi-queue capable, use **none**.

17.3. THE DEFAULT DISK SCHEDULER

Block devices use the default disk scheduler unless you specify another scheduler.



NOTE

For **non-volatile Memory Express (NVMe)** block devices specifically, the default scheduler is **none** and Red Hat recommends not changing this.

The kernel selects a default disk scheduler based on the type of device. The automatically selected scheduler is typically the optimal setting. If you require a different scheduler, Red Hat recommends to use **udev** rules or the **TuneD** application to configure it. Match the selected devices and switch the scheduler only for those devices.

17.4. DETERMINING THE ACTIVE DISK SCHEDULER

This procedure determines which disk scheduler is currently active on a given block device.

Procedure

- Read the content of the **/sys/block/*device*/queue/scheduler** file:

```
# cat /sys/block/device/queue/scheduler

[mq-deadline] kyber bfq none
```

In the file name, replace *device* with the block device name, for example **sdc**.

The active scheduler is listed in square brackets (**[ ]**).

17.5. SETTING THE DISK SCHEDULER USING TUNED

This procedure creates and enables a **TuneD** profile that sets a given disk scheduler for selected block devices. The setting persists across system reboots.

In the following commands and configuration, replace:

- *device* with the name of the block device, for example **sdf**
- *selected-scheduler* with the disk scheduler that you want to set for the device, for example **bfq**

Prerequisites

- The **tuned** service is installed and enabled. For details, see Installing and enabling TuneD.

Procedure

1. Optional: Select an existing **TuneD** profile on which your profile will be based. For a list of available profiles, see TuneD profiles distributed with RHEL.
   To see which profile is currently active, use:

```
$ tuned-adm active
```

2. Create a new directory to hold your **TuneD** profile:

```
# mkdir /etc/tuned/my-profile
```

3. Find the system unique identifier of the selected block device:

```
$ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```

NOTE

The command in the this example will return all values identified as a World Wide Name (WWN) or serial number associated with the specified block device. Although it is preferred to use a WWN, the WWN is not always available for a given device and any values returned by the example command are acceptable to use as the *device system unique ID*.

4. Create the **/etc/tuned/*my-profile*/tuned.conf** configuration file. In the file, set the following options:

   a. Optional: Include an existing profile:

```
[main]
include=existing-profile
```

b. Set the selected disk scheduler for the device that matches the WWN identifier:

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

Here:

- Replace *IDNAME* with the name of the identifier being used (for example, **ID_WWN**).
- Replace *device system unique id* with the value of the chosen identifier (for example, **0x5002538d00000000**).
  To match multiple devices in the **devices_udev_regex** option, enclose the identifiers in parentheses and separate them with vertical bars:

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. Enable your profile:

```
# tuned-adm profile my-profile
```

Verification steps

- Verify that the TuneD profile is active and applied:

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

Additional resources

- Customizing TuneD profiles

17.6. SETTING THE DISK SCHEDULER USING UDEV RULES

This procedure sets a given disk scheduler for specific block devices using **udev** rules. The setting persists across system reboots.

In the following commands and configuration, replace:

- *device* with the name of the block device, for example **sdf**
- *selected-scheduler* with the disk scheduler that you want to set for the device, for example **bfq**

Procedure

1. Find the system unique identifier of the block device:

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
E: ID_SERIAL_SHORT=20120501030900000
```

NOTE

The command in the this example will return all values identified as a World Wide Name (WWN) or serial number associated with the specified block device. Although it is preferred to use a WWN, the WWN is not always available for a given device and any values returned by the example command are acceptable to use as the *device system unique ID*.

2. Configure the **udev** rule. Create the **/etc/udev/rules.d/99-scheduler.rules** file with the following content:

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

Here:

- Replace *IDNAME* with the name of the identifier being used (for example, **ID_WWN**).
- Replace *device system unique id* with the value of the chosen identifier (for example, **0x5002538d00000000**).

3. Reload **udev** rules:

```
# udevadm control --reload-rules
```

4. Apply the scheduler configuration:

```
# udevadm trigger --type=devices --action=change
```

Verification steps

- Verify the active scheduler:

```
# cat /sys/block/device/queue/scheduler
```

17.7. TEMPORARILY SETTING A SCHEDULER FOR A SPECIFIC DISK

This procedure sets a given disk scheduler for specific block devices. The setting does not persist across system reboots.

Procedure

- Write the name of the selected scheduler to the **/sys/block/*device*/queue/scheduler** file:

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

In the file name, replace *device* with the block device name, for example **sdc**.

Verification steps

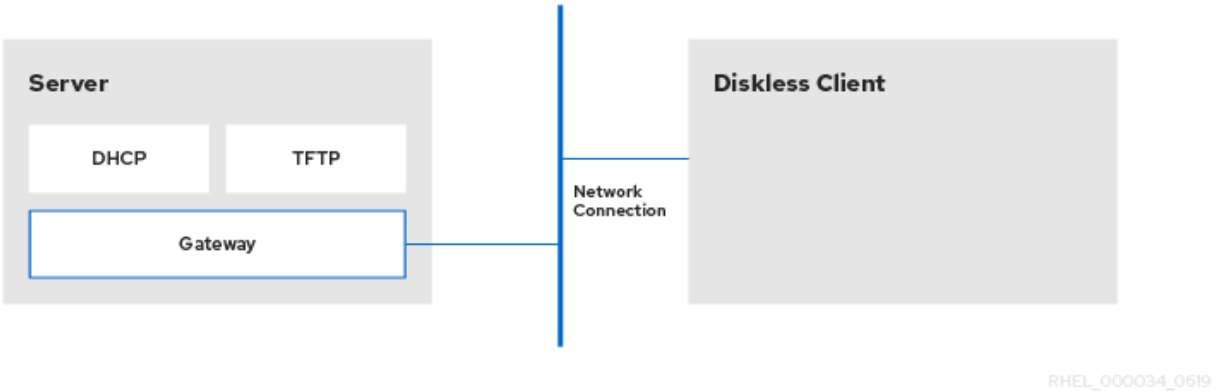
- Verify that the scheduler is active on the device:

```
# cat /sys/block/device/queue/scheduler
```

CHAPTER 18. SETTING UP A REMOTE DISKLESS SYSTEM

The following sections outline the necessary procedures for deploying remote diskless systems in a network environment. It is useful to implement this solution when you require multiple clients with identical configuration. Also, that will save the cost for hard drives for the number of the clients. Assuming, the server has Red Hat Enterprise Linux 8 operating system installed.

Figure 18.1. Remote diskless system settings diagram



Note, that gateway might be configured on a separate server.

18.1. PREPARING AN ENVIRONMENT FOR THE REMOTE DISKLESS SYSTEM

This procedure describes the preparation of the environment for the remote diskless system.

Remote diskless system booting requires both a **tftp** service (provided by **tftp-server**) and a DHCP service (provided by **dhcp**). The **tftp** service is used to retrieve kernel image and **initrd** over the network via the PXE loader.

Prerequisites

- Install the following packages:
  - **tftp-server**
  - **xinetd**
  - **dhcp-server**
  - **syslinux**
- Set up the network connection.

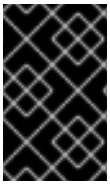
Procedure

1. Install the **dracut-network** package:

```
# yum install dracut-network
```

2. After installing the **dracut-network** package, add the following line to **/etc/dracut.conf**:

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



IMPORTANT

Some RPM packages have started using file capabilities (such as **setcap** and **getcap**). However, NFS does not currently support these so attempting to install or update any packages that use file capabilities will fail.

At this point you have the server ready to continue with remote diskless system implementation.

18.2. CONFIGURING A TFTP SERVICE FOR DISKLESS CLIENTS

This procedure describes how to configure a tftp service for a diskless client.

Prerequisites

- Install the necessary packages. See prerequisites in Section 18.1, "Preparing an environment for the remote diskless system".

To Configure tftp

1. Enable PXE booting over the network:

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

2. The **tftp** root directory (**chroot**) is located in **/var/lib/tftpboot**. Copy **/usr/share/syslinux/pxelinux.0** to **/var/lib/tftpboot/**:

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

3. Copy **/usr/share/syslinux/ldlinux.c32** to **/var/lib/tftpboot/**:

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

4. Create a **pxelinux.cfg** directory inside the **tftp** root directory:

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

5. After configuring **tftp** for diskless clients, configure DHCP, NFS, and the exported file system accordingly.

18.3. CONFIGURING DHCP SERVER FOR DISKLESS CLIENTS

This procedure describes how to configure DHCP for a diskless system.

Prerequisites

- Install the necessary packages. See prerequisites in Section 18.1, "Preparing an environment for the remote diskless system".
- Configure **tftp**. See Section 18.2, "Configuring a tftp service for diskless clients" .

Procedure

1. Set up a DHCP server and enable PXE booting by adding the following configuration to **/etc/dhcp/dhcpd.conf**:

```
allow booting;
allow bootp;
subnet 192.168.205.0 netmask 255.255.255.0 {
  pool
  {
    range 192.168.205.10 192.168.205.25;
  }

  option subnet-mask 255.255.255.0;
  option routers 192.168.205.1;
}
class "pxeclients" {
   match if substring(option vendor-class-identifier, 0, 9) = "PXEClient";
   next-server server-ip;
   filename "pxelinux.0";
}
```

This configuration will not boot over UEFI. To perform installation for UEFI, follow the procedure from this documentation: Configuring a TFTP server for UEFI-based clients . Also, note that the **/etc/dhcp/dhcpd.conf** is an example file.

NOTE

When **libvirt** virtual machines are used as a diskless client, **libvirt** provides the DHCP service and the stand alone DHCP server is not used. In this situation, network booting must be enabled with the **bootp file='*filename*'** option in the **libvirt** network configuration, **virsh net-edit**.

2. Enable **dhcpd.service** by entering the following command:

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

18.4. CONFIGURING AN EXPORTED FILE SYSTEM FOR DISKLESS CLIENTS

This procedure describes how to configure an exported file system for diskless client.

Prerequisites

- Install the necessary packages. See prerequisites in Section 18.1, “Preparing an environment for the remote diskless system”.
- Configure **tftp**. See Section 18.2, “Configuring a tftp service for diskless clients” .
- Configure DHCP. See Section 18.3, “Configuring DHCP server for diskless clients” .

Procedure

1. Configure the NFS server to export the root directory by adding it to **/etc/exports**. For the instructions see NFS server configuration.

2. To accommodate completely diskless clients, the root directory should contain a complete Red Hat Enterprise Linux installation. You can either install a new base system or clone an existing installation:

   - To install Red Hat Enterprise Linux to the exported location, use the **yum** utility with the **--installroot** option:

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

   - To synchronize with a running system, use the **rsync** utility:

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

     - Replace *example.com* with the hostname of the running system with which to synchronize via the **rsync** utility.

     - Replace *exported-root-directory* with the path to the exported file system.
       Note, that for this option you must have a separate existing running system, which you will clone to the server by the command above.

The file system to be exported still needs to be configured further before it can be used by diskless clients. To do this, perform the following procedure:

Configure File System

1. Select the kernel that diskless clients should use (**vmlinuz-*kernel-version***) and copy it to the **tftp** boot directory:

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

2. Create the **initrd** (that is, **initramfs-*kernel-version*.img**) with NFS support:

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

3. Change file permissions for **initrd** to 644 using the following command:

   ```
   # chmod 644 /exported-root-directory/boot/initramfs-<kernel-version>.img
   ```

   WARNING

   If you do not change the initrd's file permissions, the **pxelinux.0** boot loader will fail with a "file not found" error.

4. Copy the resulting **initramfs-*kernel-version*.img** into the **tftp** boot directory:

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

5. Edit the default boot configuration to use the **initrd** and kernel in the **/var/lib/tftpboot/** directory. This configuration should instruct the diskless client's root to mount the exported file system (***/exported-root-directory***) as read-write. Add the following configuration in the **/var/lib/tftpboot/pxelinux.cfg/default** file:

```
default rhel8

label rhel8
  kernel vmlinuz-kernel-version
  append initrd=initramfs-kernel-version.img root=nfs:server-ip:/exported-root-directory rw
```

   Replace ***server-ip*** with the IP address of the host machine on which the **tftp** and DHCP services reside.

6. Optionally, you can mount the system in *read-only* format by using the following configuration in the **/var/lib/tftpboot/pxelinux.cfg/default** file:

```
default rhel8

label rhel8
  kernel vmlinuz-kernel-version
  append initrd=initramfs-kernel-version.img root=nfs:server-ip:/exported-root-directory ro
```

7. Reboot the NFS server.

The NFS share is now ready for exporting to diskless clients. These clients can boot over the network via PXE.

18.5. RE-CONFIGURING A REMOTE DISKLESS SYSTEM

You need to re-configure the system in some cases. The steps below show how to change the password for a user, how to install software on a system and describe how to split system into a **/usr** that is in read-only mode and a **/var** that is in read-write mode.

Prerequisites

- **no_root_squash** option is enabled in the exported file system.

Procedure

1. To change the user password, follow the steps below:
   - Change the command line to */exported/root/directory*:

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

   - Change the password for the user you want:

     ```
     # passwd <username>
     ```

     Replace the *<username>* with a real user to whom you want to change the password.

- Exit the command line:

```
# exit
```

2. To install software to a remote diskless system, use the following command:

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

Replace *<package>* with the actual package you want to install.

3. To split a remote diskless system into a **/usr** and a **/var** you must configure two separate exports. Read NFS server configuration documentation for details.

18.6. THE MOST COMMON ISSUES WITH LOADING A REMOTE DISKLESS SYSTEM

The following section describes the issues during loading the remote diskless system on a diskless client and shows the possible solution for them.

18.6.1. The client does not get an IP address

To troubleshoot that problem:

1. Check if the DHCP service is enabled on the server.
   - Check if the **dhcp.service** is running:

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

   - If the **dhcp.service** is inactive, you must enable and start it:

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

   Reboot the diskless client.

2. If the problem remains, check the DHCP configurational file */etc/dhcp/dhcpd.conf* on a server. For more information, see Section 18.3, “Configuring DHCP server for diskless clients” .

3. Check if the Firewall ports are opened.
   - Check if the **tftp.service** is listed in active services:

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

   - If the **tftp.service** is not listed in active services, add it to the list:

```
# firewall-cmd --add-service=tftp
```

   - Check if the **nfs.service** is listed in active services:

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

- If the **nfs.service** is not listed in active services, add it to the list:

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

18.6.2. The files are not available during the booting a remote diskless system

To troubleshoot this problem:

1. Check if the file is in place. The location on a server */var/lib/tftpboot/*.
2. If the file is in place, check its permissions:

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

3. Check if the Firewall ports are opened.

18.6.3. System boot failed after loading kernel/initrd

To troubleshoot this problem:

1. Check if NFS service is enabled on a server.
   - Check if **nfs.service** is running:

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

   - If the **nfs.service** is inactive, you must enable and start it:

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

2. Check if the parameters are correct in *pxelinux.cfg*. For more details, see Section 18.4, “Configuring an exported file system for diskless clients”.
3. Check if the Firewall ports are opened.

CHAPTER 19. MANAGING RAID

This chapter describes Redundant Array of Independent Disks (RAID). User can use RAID to store data across multiple drives. It also helps to avoid data loss if a drive has failed.

19.1. REDUNDANT ARRAY OF INDEPENDENT DISKS (RAID)

The basic idea behind RAID is to combine multiple devices, such as **HDD**, **SSD** or **NVMe**, into an array to accomplish performance or redundancy goals not attainable with one large and expensive drive. This array of devices appears to the computer as a single logical storage unit or drive.

RAID allows information to be spread across several devices. RAID uses techniques such as *disk striping* (RAID Level 0), *disk mirroring* (RAID Level 1), and *disk striping with parity* (RAID Levels 4, 5 and 6) to achieve redundancy, lower latency, increased bandwidth, and maximized ability to recover from hard disk crashes.

RAID distributes data across each device in the array by breaking it down into consistently-sized chunks (commonly 256K or 512k, although other values are acceptable). Each chunk is then written to a hard drive in the RAID array according to the RAID level employed. When the data is read, the process is reversed, giving the illusion that the multiple devices in the array are actually one large drive.

System Administrators and others who manage large amounts of data would benefit from using RAID technology. Primary reasons to deploy RAID include:

- Enhances speed
- Increases storage capacity using a single virtual disk
- Minimizes data loss from disk failure
- RAID layout and level online conversion

19.2. RAID TYPES

There are three possible RAID approaches: Firmware RAID, Hardware RAID, and Software RAID.

Firmware RAID

Firmware RAID, also known as ATARAID, is a type of software RAID where the RAID sets can be configured using a firmware-based menu. The firmware used by this type of RAID also hooks into the BIOS, allowing you to boot from its RAID sets. Different vendors use different on-disk metadata formats to mark the RAID set members. The Intel Matrix RAID is a good example of a firmware RAID system.

Hardware RAID

The hardware-based array manages the RAID subsystem independently from the host. It may present multiple devices per RAID array to the host.

Hardware RAID devices may be internal or external to the system. Internal devices commonly consisting of a specialized controller card that handles the RAID tasks transparently to the operating system. External devices commonly connect to the system via SCSI, Fibre Channel, iSCSI, InfiniBand, or other high speed network interconnect and present volumes such as logical units to the system.

RAID controller cards function like a SCSI controller to the operating system, and handle all the actual drive communications. The user plugs the drives into the RAID controller (just like a normal SCSI controller) and then adds them to the RAID controller's configuration. The operating system will not be

able to tell the difference.

Software RAID

Software RAID implements the various RAID levels in the kernel block device code. It offers the cheapest possible solution, as expensive disk controller cards or hot-swap chassis [1] are not required. Software RAID also works with any block storage which are supported by the Linux kernel, such as **SATA**, **SCSI**, and **NVMe**. With today's faster CPUs, Software RAID also generally outperforms Hardware RAID, unless you use high-end storage devices.

The Linux kernel contains a *multiple device* (MD) driver that allows the RAID solution to be completely hardware independent. The performance of a software-based array depends on the server CPU performance and load.

Key features of the Linux software RAID stack:

- Multithreaded design
- Portability of arrays between Linux machines without reconstruction
- Backgrounded array reconstruction using idle system resources
- Hot-swappable drive support
- Automatic CPU detection to take advantage of certain CPU features such as streaming Single Instruction Multiple Data (SIMD) support
- Automatic correction of bad sectors on disks in an array
- Regular consistency checks of RAID data to ensure the health of the array
- Proactive monitoring of arrays with email alerts sent to a designated email address on important events
- Write-intent bitmaps which drastically increase the speed of resync events by allowing the kernel to know precisely which portions of a disk need to be resynced instead of having to resync the entire array after a system crash
  Note that *resync* is a process to synchronize the data over the devices in the existing RAID to achieve redundancy
- Resync checkpointing so that if you reboot your computer during a resync, at startup the resync will pick up where it left off and not start all over again
- The ability to change parameters of the array after installation, which is called *reshaping*. For example, you can grow a 4-disk RAID5 array to a 5-disk RAID5 array when you have a new device to add. This grow operation is done live and does not require you to reinstall on the new array
- Reshaping supports changing the number of devices, the RAID algorithm or size of the RAID array type, such as RAID4, RAID5, RAID6 or RAID10
- Takeover supports RAID level converting, such as RAID0 to RAID6

19.3. RAID LEVELS AND LINEAR SUPPORT

RAID supports various configurations, including levels 0, 1, 4, 5, 6, 10, and linear. These RAID types are defined as follows:

Level 0

RAID level 0, often called *striping*, is a performance-oriented striped data mapping technique. This means the data being written to the array is broken down into stripes and written across the member disks of the array, allowing high I/O performance at low inherent cost but provides no redundancy. Many RAID level 0 implementations only stripe the data across the member devices up to the size of the smallest device in the array. This means that if you have multiple devices with slightly different sizes, each device gets treated as though it was the same size as the smallest drive. Therefore, the common storage capacity of a level 0 array is equal to the capacity of the smallest member disk in a Hardware RAID or the capacity of smallest member partition in a Software RAID multiplied by the number of disks or partitions in the array.

Level 1

RAID level 1, or *mirroring*, provides redundancy by writing identical data to each member disk of the array, leaving a "mirrored" copy on each disk. Mirroring remains popular due to its simplicity and high level of data availability. Level 1 operates with two or more disks, and provides very good data reliability and improves performance for read-intensive applications but at a relatively high cost. RAID level 1 comes at a high cost because you write the same information to all of the disks in the array, provides data reliability, but in a much less space-efficient manner than parity based RAID levels such as level 5. However, this space inefficiency comes with a performance benefit: parity-based RAID levels consume considerably more CPU power in order to generate the parity while RAID level 1 simply writes the same data more than once to the multiple RAID members with very little CPU overhead. As such, RAID level 1 can outperform the parity-based RAID levels on machines where software RAID is employed and CPU resources on the machine are consistently taxed with operations other than RAID activities.

The storage capacity of the level 1 array is equal to the capacity of the smallest mirrored hard disk in a Hardware RAID or the smallest mirrored partition in a Software RAID. Level 1 redundancy is the highest possible among all RAID types, with the array being able to operate with only a single disk present.

Level 4

Level 4 uses parity concentrated on a single disk drive to protect data. Parity information is calculated based on the content of the rest of the member disks in the array. This information can then be used to reconstruct data when one disk in the array fails. The reconstructed data can then be used to satisfy I/O requests to the failed disk before it is replaced and to repopulate the failed disk after it has been replaced.
Because the dedicated parity disk represents an inherent bottleneck on all write transactions to the RAID array, level 4 is seldom used without accompanying technologies such as write-back caching, or in specific circumstances where the system administrator is intentionally designing the software RAID device with this bottleneck in mind (such as an array that will have little to no write transactions once the array is populated with data). RAID level 4 is so rarely used that it is not available as an option in Anaconda. However, it could be created manually by the user if truly needed.

The storage capacity of Hardware RAID level 4 is equal to the capacity of the smallest member partition multiplied by the number of partitions *minus one*. Performance of a RAID level 4 array is always asymmetrical, meaning reads outperform writes. This is because writes consume extra CPU and main memory bandwidth when generating parity, and then also consume extra bus bandwidth when writing the actual data to disks because you are writing not only the data, but also the parity. Reads need only read the data and not the parity unless the array is in a degraded state. As a result, reads generate less traffic to the drives and across the buses of the computer for the same amount of data transfer under normal operating conditions.

Level 5

This is the most common type of RAID. By distributing parity across all the member disk drives of an

array, RAID level 5 eliminates the write bottleneck inherent in level 4. The only performance bottleneck is the parity calculation process itself. With modern CPUs and Software RAID, that is usually not a bottleneck at all since modern CPUs can generate parity very fast. However, if you have a sufficiently large number of member devices in a software RAID5 array such that the combined aggregate data transfer speed across all devices is high enough, then this bottleneck can start to come into play.
As with level 4, level 5 has asymmetrical performance, and reads substantially outperforming writes.
The storage capacity of RAID level 5 is calculated the same way as with level 4.

Level 6

This is a common level of RAID when data redundancy and preservation, and not performance, are the paramount concerns, but where the space inefficiency of level 1 is not acceptable. Level 6 uses a complex parity scheme to be able to recover from the loss of any two drives in the array. This complex parity scheme creates a significantly higher CPU burden on software RAID devices and also imposes an increased burden during write transactions. As such, level 6 is considerably more asymmetrical in performance than levels 4 and 5.
The total capacity of a RAID level 6 array is calculated similarly to RAID level 5 and 4, except that you must subtract 2 devices (instead of 1) from the device count for the extra parity storage space.

Level 10

This RAID level attempts to combine the performance advantages of level 0 with the redundancy of level 1. It also helps to alleviate some of the space wasted in level 1 arrays with more than 2 devices. With level 10, it is possible for instance to create a 3-drive array configured to store only 2 copies of each piece of data, which then allows the overall array size to be 1.5 times the size of the smallest devices instead of only equal to the smallest device (like it would be with a 3-device, level 1 array). This avoids CPU process usage to calculate parity like with RAID level 6, but it is less space efficient.
The creation of RAID level 10 is not supported during installation. It is possible to create one manually after installation.

Linear RAID

Linear RAID is a grouping of drives to create a larger virtual drive.
In linear RAID, the chunks are allocated sequentially from one member drive, going to the next drive only when the first is completely filled. This grouping provides no performance benefit, as it is unlikely that any I/O operations split between member drives. Linear RAID also offers no redundancy and decreases reliability. If any one member drive fails, the entire array cannot be used. The capacity is the total of all member disks.

19.4. LINUX RAID SUBSYSTEMS

The following subsystems compose RAID in Linux:

19.4.1. Linux Hardware RAID Controller Drivers

Hardware RAID controllers have no specific RAID subsystem in Linux. Because they use special RAID chipsets, hardware RAID controllers come with their own drivers; these drivers allow the system to detect the RAID sets as regular disks.

19.4.2. mdraid

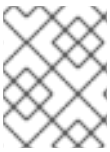
The **mdraid** subsystem was designed as a software RAID solution for Linux; it is also the preferred solution for software RAID under Linux. This subsystem uses its own metadata format, generally referred to as native MD metadata.

mdraid also supports other metadata formats, known as external metadata. Red Hat Enterprise Linux 8 uses **mdraid** with external metadata to access ISW / IMSM (Intel firmware RAID) sets and SNIA DDF. **mdraid** sets are configured and controlled through the **mdadm** utility.

19.5. CREATING SOFTWARE RAID

Follow the steps in this procedure to create a Redundant Arrays of Independent Disks (RAID) device. RAID devices are constructed from multiple storage devices that are arranged to provide increased performance and, in some configurations, greater fault tolerance.

A RAID device is created in one step and disks are added or removed as necessary. You can configure one RAID partition for each physical disk in your system, so the number of disks available to the installation program determines the levels of RAID device available. For example, if your system has two hard drives, you cannot create a RAID 10 device, as it requires a minimum of three separate disks.



NOTE

On 64-bit IBM Z, the storage subsystem uses RAID transparently. You do not have to configure software RAID manually.

Prerequisites

- You have selected two or more disks for installation before RAID configuration options are visible. At least two disks are required to create a RAID device.
- You have created a mount point. By configuring a mount point, you configure the RAID device.
- You have selected the **Custom** radio button on the **Installation Destination** window.

Procedure

1. From the left pane of the **Manual Partitioning** window, select the required partition.
2. Under the **Device(s)** section, click **Modify**. The **Configure Mount Point** dialog box opens.
3. Select the disks that you want to include in the RAID device and click **Select**.
4. Click the **Device Type** drop-down menu and select **RAID**.
5. Click the **File System** drop-down menu and select your preferred file system type.
6. Click the **RAID Level** drop-down menu and select your preferred level of RAID.
7. Click **Update Settings** to save your changes.
8. Click **Done** to apply the settings and return to the **Installation Summary** window.

A message is displayed at the bottom of the window if the specified RAID level requires more disks.

To create and configure a RAID volume using the Storage system role, see Section 2.12, “Configuring a RAID volume using the storage system role”

To learn more about soft corruption and how you can protect your data when configuring a RAID LV, see Using DM integrity with RAID LV.

19.6. CREATING SOFTWARE RAID AFTER INSTALLATION

This procedure describes how to create a software Redundant Array of Independent Disks (RAID) on an existing system using **mdadm** utility.

Prerequisites

- The **mdadm** package installed.
- Two or more partitions exist on your system. For detailed instruction, see Section 3.3, "Creating a partition".

Procedure

1. To create RAID of two block devices, for example **/dev/sda1** and **/dev/sdc1**, use the following command:

```
# mdadm --create /dev/md0 --level=<level_value> --raid-devices=2 /dev/sda1 /dev/sdc1
```

   Replace *<level_value>* to a RAID level option. For more information, see **mdadm(8)** man page.

2. Optionally, to check the status of RAID, use the following command:

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

3. Optionally, to observe the detailed information about each RAID device, use the following command:

```
# mdadm --examine /dev/sda1 /dev/sdc1
```

4. To create a file system on a RAID drive, use the following command:

```
# mkfs -t <file-system-name> /dev/md0
```

   where *<file-system-name>* is a specific file system that you chose to format the drive with. For more information, see **mkfs** man page.

5. To create a mount point for RAID drive and mount it, use the following commands:

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

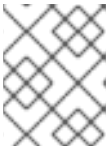
After you finish the steps above, the RAID is ready to be used.

19.7. CONFIGURING A RAID VOLUME USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure a RAID volume on RHEL using Red Hat Ansible Automation Platform. In this section you will learn how to set up an Ansible playbook with the available parameters to configure a RAID volume to suit your requirements.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.



NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to deploy the **storage** solution.

- You have the **rhel-system-roles** package installed on the system from which you want to run the playbook.
- You have an inventory file detailing the systems on which you want to deploy a RAID volume using the **storage** System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_volumes:
      - name: data
        type: raid
        disks: [sdd, sde, sdf, sdg]
        raid_level: raid0
        raid_chunk_size: 32 KiB
        mount_point: /mnt/data
        state: present
  roles:
    - name: rhel-system-roles.storage
```

WARNING

Device names can change in certain circumstances; for example, when you add a new disk to a system. Therefore, to prevent data loss, we do not recommend using specific disk names in the playbook.

2. Optional. Verify playbook syntax.

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

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

19.8. RECONFIGURING RAID

The section below describes how to modify an existing RAID. To do so, choose one of the methods:

- Changing RAID attributes (also known as RAID *reshape*).
- Converting RAID level (also known as RAID *takeover*).

19.8.1. Reshaping RAID

This chapter below describes how to reshape RAID. You can choose one of the methods of resizing RAID:

- Enlarging (extending) RAID.
- Shrinking RAID.

19.8.1.1. Resizing RAID (extending)

This procedure describes how to enlarge RAID. Assuming **/dev/md0** is RAID you want to enlarge.

Prerequisites

- Enough disk space.
- The package **parted** is installed.

Procedure

1. Extend RAID partitions. To do so, follow the instruction in Resizing a partition documentation.
2. To extend RAID to the maximum of the partition capacity, use this command:

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

   Note that to determine a specific size, you must write the *--size* parameter in kB (for example *--size=524228*).
3. Increase the size of file system. For more information, check the Managing file systems documentation.

19.8.1.2. Resizing RAID (shrinking)

This procedure describes how to shrink RAID. Assuming **/dev/md0** is the RAID you want to shrink to 512 MB.

Prerequisites

- The package **parted** is installed.

Procedure

1. Shrink the file system. To do so, check the Managing file systems documentation.

IMPORTANT

The *XFS* file system does not support shrinking.

2. To decrease the RAID to the size of 512 MB, use this command:

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

Note, you must write the *--size* parameter in kB.

3. Shrink the partition to the size you need. To do so, follow the instruction in the Resizing a partition documentation.

19.8.2. RAID takeover

This chapter describes supported conversions in RAID and contains procedures to accomplish those conversions.

19.8.2.1. Supported RAID conversions

It is possible to convert from one RAID level to another. This section provides a table that lists supported RAID conversions.

Source level	Destination
RAID0	RAID4, RAID5, RAID10
RAID1	RAID5
RAID4	RAID5
RAID5	RAID0, RAID1, RAID4, RAID6, RAID10
RAID6	RAID5
RAID10	RAID0

Additional resources

- For more information about RAID level conversion, read **mdadm** man page.

19.8.2.2. Converting RAID level

This procedure describes how to convert RAID to a different RAID level. Assuming, you want to convert RAID **/dev/md0** level 0 to RAID level 5 and add one more disk **/dev/sdd** to the array.

Prerequisites

- Enough disks for conversion.
- The package **mdadm** is installed.
- Ensure the intended conversion is supported. To check if that is the case, see the table in Section 19.8.2.1, “Supported RAID conversions” .

Procedure

1. To convert the RAID **/dev/md0** to RAID level 5, use the following command:

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

2. To add a new disk to the array, use the following command:

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

3. To check new details of the converted array, use the following command:

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

Additional resources

- For more information about RAID level conversion, read **mdadm** man page.

19.9. CONVERTING A ROOT DISK TO RAID1 AFTER INSTALLATION

This section describes how to convert a non-RAID root disk to a RAID1 mirror after installing Red Hat Enterprise Linux 8.

On the PowerPC (PPC) architecture, take the following additional steps:

Prerequisites

- The instructions in the following Red Hat Knowledgebase article are completed: How do I convert my root disk to RAID1 after installation of Red Hat Enterprise Linux 7?.

Procedure

1. Copy the contents of the PowerPC Reference Platform (PReP) boot partition from **/dev/sda1** to **/dev/sdb1**:

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

2. Update the Prep and boot flag on the first partition on both disks:

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

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

NOTE

Running the **grub2-install /dev/sda** command does not work on a PowerPC machine and returns an error, but the system boots as expected.

19.10. CREATING ADVANCED RAID DEVICES

In some cases, you may wish to install the operating system on an array that can not be created after the installation completes. Usually, this means setting up the **/boot** or root file system arrays on a complex RAID device; in such cases, you may need to use array options that are not supported by **Anaconda** installer. To work around this, perform the following procedure:

Procedure

1. Insert the install disk.
2. During the initial boot up, select **Rescue Mode** instead of **Install** or **Upgrade**. When the system fully boots into *Rescue mode*, the user will be presented with a command line terminal.
3. From this terminal, use **parted** to create RAID partitions on the target hard drives. Then, use **mdadm** to manually create raid arrays from those partitions using any and all settings and options available. For more information on how to do these, see **man parted** and **man mdadm**.
4. Once the arrays are created, you can optionally create file systems on the arrays as well.
5. Reboot the computer and select **Install** or **Upgrade** to install as normal. As **Anaconda** installer searches the disks in the system, it will find the pre-existing RAID devices.
6. When asked about how to use the disks in the system, select **Custom Layout** and click **Next**. In the device listing, the pre-existing MD RAID devices will be listed.
7. Select a RAID device, click **Edit** and configure its mount point and (optionally) the type of file system it should use (if you did not create one earlier) then click **Done**. **Anaconda** will perform the install to this pre-existing RAID device, preserving the custom options you selected when you created it in *Rescue Mode*.

NOTE

The limited *Rescue Mode* of the installer does not include **man** pages. Both the **man mdadm** and **man md** contain useful information for creating custom RAID arrays, and may be needed throughout the workaround. As such, it can be helpful to either have access to a machine with these **man** pages present, or to print them out prior to booting into *Rescue Mode* and creating your custom arrays.

19.11. MONITORING RAID

This module describes how to set up the RAID monitoring option with **mdadm** tool.

Prerequisites

- The package **mdadm** is installed
- The mail service is set up.

Procedure

1. To create a configuration file for monitoring array you must scan the details and forward the result to **/etc/mdadm.conf** file. To do so, use the following command:

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

   Note, that *ARRAY* and *MAILADDR* are mandatory variables.

2. Open the configuration file **/etc/mdadm.conf** with a text editor of your choice.

3. Add the *MAILADDR* variable with the mail address for the notification. For example, add new line:

```
MAILADDR <example@example.com>
```

   where *example@example.com* is an email address to which you want to receive the alerts from the array monitoring.

4. Save changes in the **/etc/mdadm.conf** file and close it.

After you complete the steps above, the monitoring system will send the alerts to the email address.

Additional resources

- For more information, read the **mdadm.conf 5** man page.

19.12. MAINTAINING RAID

This section provides various procedures for RAID maintenance.

19.12.1. Replacing a faulty disk in a RAID

This procedure describes how to replace the faulty disk in a redundant array of independent disks (RAID). Assuming, you have **/dev/md0** RAID level 10. In this scenario, the **/dev/sdg** disk is faulty and you need to replace it with new disk **/dev/sdh**.

Prerequisites

- Additional spare disk.
- The **mdadm** package is installed.
- A notification about a faulty disk in an array. To set up the array monitoring, see Section 19.11, “Monitoring RAID”.

Procedure

1. Ensure which disk is failing. To do so, enter the following command:

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

   You will find a message showing you which disk has failed:

```
md/raid:md0: Disk failure on sdg, disabling device.
md/raid:md0: Operation continuing on 5 devices.
```

2. Press **Ctrl+C** on your keyboard to exit the **journalctl** program.

3. Add a new disk to the array. To do so, enter the following command:

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

4. Mark the failed disk as faulty. To do so, enter the following command:

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

5. Check if the faulty disk was masked correctly by using the following command:

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

At the end of the last command output you will see information about RAID disks similar to this where disk **/dev/sdg** has a **faulty** status:

```
Number   Major   Minor   RaidDevice State
   0       8      16        0      active sync   /dev/sdb
   1       8      32        1      active sync   /dev/sdc
   2       8      48        2      active sync   /dev/sdd
   3       8      64        3      active sync   /dev/sde
   4       8      80        4      active sync   /dev/sdf
   6       8     112        5      active sync   /dev/sdh

   5       8      96        -      faulty   /dev/sdg
```

6. Finally, remove the faulty disk from the array. To do so, enter the following command:

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

7. Check RAID details by using following command:

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

At the end of the last command output you will see information about RAID disks similar to this:

```
Number   Major   Minor   RaidDevice State
   0       8      16        0      active sync   /dev/sdb
   1       8      32        1      active sync   /dev/sdc
   2       8      48        2      active sync   /dev/sdd
   3       8      64        3      active sync   /dev/sde
   4       8      80        4      active sync   /dev/sdf
   6       8     112        5      active sync   /dev/sdh
```

After completing the steps above you will have RAID **/dev/md0** with a new disk **/dev/sdh**.

19.12.2. Replacing a broken disk in array

This procedure describes how to replace the broken disk in a redundant array of independent disks (RAID). Assuming, you have **/dev/md0** RAID level 6. In this scenario, the **/dev/sdb** disk has hardware issue and could not be used any longer. You need to replace it with the new disk **/dev/sdi**.

Prerequisites

- New disk for replacement.
- The **mdadm** package is installed.

Procedure

1. Check the log message by using the following command:

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

   You will find a message showing you which disk has failed:

```
md/raid:md0: Disk failure on sdb, disabling device.
md/raid:md0: Operation continuing on 5 devices.
```

2. Press **Ctrl+C** on your keyboard to exit the **journalctl** program.
3. Add the new disk to the array as a spare one. To do so, enter the following command:

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

4. Mark the broken disk as **faulty**. To do so, enter the following command:

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

5. Remove the faulty disk from the array. To do so, enter the following command:

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

6. Check the status of the array by using the following command:

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

   At the end of the last command output you will see information about RAID disks similar to this:

```
Number   Major   Minor   RaidDevice State
   7       8      128       0      active sync   /dev/sdi
   1       8       32       1      active sync   /dev/sdc
   2       8       48       2      active sync   /dev/sdd
   3       8       64       3      active sync   /dev/sde
   4       8       80       4      active sync   /dev/sdf
   6       8      112       5      active sync   /dev/sdh
```

After completing the steps above you will have RAID **/dev/md0** with a new disk **/dev/sdi**.

19.12.3. Resynchronizing RAID disks

This procedure describes how to resynchronize disks in a RAID array. Assuming, you have **/dev/md0** RAID.

Prerequisites

- Package **mdadm** is installed.

Procedure

1. To check the array for the failed disks behavior, enter the following command:

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

   That action will check the array and write the result into the **/sys/block/md0/md/sync_action** file.

2. Open file **/sys/block/md0/md/sync_action** with the text editor of your choice and see if there is any message about disk synchronization failures.

3. To resynchronize the disks in the array, enter the following command:

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

   This action will resynchronize the disks in the array and write the result into the **/sys/block/md0/md/sync_action** file.

4. To view the synchronization progress, enter the following command:

```
# cat /proc/mdstat
```

[1] A hot-swap chassis allows you to remove a hard drive without having to power-down your system.

CHAPTER 20. ENCRYPTING BLOCK DEVICES USING LUKS

Disk encryption protects the data on a block device by encrypting it. To access the device's decrypted contents, a user must provide a passphrase or key as authentication. This is particularly important when it comes to mobile computers and removable media: it helps to protect the device's contents even if it has been physically removed from the system. The LUKS format is a default implementation of block device encryption in RHEL.

20.1. LUKS DISK ENCRYPTION

The Linux Unified Key Setup-on-disk-format (LUKS) enables you to encrypt block devices and it provides a set of tools that simplifies managing the encrypted devices. LUKS allows multiple user keys to decrypt a master key, which is used for the bulk encryption of the partition.

RHEL utilizes LUKS to perform block device encryption. By default, the option to encrypt the block device is unchecked during the installation. If you select the option to encrypt your disk, the system prompts you for a passphrase every time you boot the computer. This passphrase "unlocks" the bulk encryption key that decrypts your partition. If you choose to modify the default partition table, you can choose which partitions you want to encrypt. This is set in the partition table settings.

What LUKS does

- LUKS encrypts entire block devices and is therefore well-suited for protecting contents of mobile devices such as removable storage media or laptop disk drives.
- The underlying contents of the encrypted block device are arbitrary, which makes it useful for encrypting swap devices. This can also be useful with certain databases that use specially formatted block devices for data storage.
- LUKS uses the existing device mapper kernel subsystem.
- LUKS provides passphrase strengthening, which protects against dictionary attacks.
- LUKS devices contain multiple key slots, allowing users to add backup keys or passphrases.

What LUKS does*not* do

- Disk-encryption solutions like LUKS protect the data only when your system is off. Once the system is on and LUKS has decrypted the disk, the files on that disk are available to anyone who would normally have access to them.
- LUKS is not well-suited for scenarios that require many users to have distinct access keys to the same device. The LUKS1 format provides eight key slots, LUKS2 up to 32 key slots.
- LUKS is not well-suited for applications requiring file-level encryption.

Ciphers

The default cipher used for LUKS is **aes-xts-plain64**. The default key size for LUKS is 512 bits. The default key size for LUKS with **Anaconda** (XTS mode) is 512 bits. Ciphers that are available are:

- AES - Advanced Encryption Standard
- Twofish (a 128-bit block cipher)
- Serpent

Additional resources

- LUKS Project Home Page
- LUKS On-Disk Format Specification
- FIPS PUB 197

20.2. LUKS VERSIONS IN RHEL

In RHEL, the default format for LUKS encryption is LUKS2. The legacy LUKS1 format remains fully supported and it is provided as a format compatible with earlier RHEL releases.

The LUKS2 format is designed to enable future updates of various parts without a need to modify binary structures. LUKS2 internally uses JSON text format for metadata, provides redundancy of metadata, detects metadata corruption and allows automatic repairs from a metadata copy.

IMPORTANT

Do not use LUKS2 in systems that must be compatible with legacy systems that support only LUKS1. Note that RHEL 7 supports the LUKS2 format since version 7.6.

WARNING

LUKS2 and LUKS1 use different commands to encrypt the disk. Using the wrong command for a LUKS version might cause data loss.

LUKS version	Encryption command
LUKS2	**cryptsetup reencrypt**
LUKS1	**cryptsetup-reencrypt**

Online re-encryption

The LUKS2 format supports re-encrypting encrypted devices while the devices are in use. For example, you do not have to unmount the file system on the device to perform the following tasks:

- Change the volume key
- Change the encryption algorithm

When encrypting a non-encrypted device, you must still unmount the file system. You can remount the file system after a short initialization of the encryption.

The LUKS1 format does not support online re-encryption.

Conversion

The LUKS2 format is inspired by LUKS1. In certain situations, you can convert LUKS1 to LUKS2. The conversion is not possible specifically in the following scenarios:

- A LUKS1 device is marked as being used by a Policy-Based Decryption (PBD - Clevis) solution. The **cryptsetup** tool refuses to convert the device when some **luksmeta** metadata are detected.
- A device is active. The device must be in the inactive state before any conversion is possible.

20.3. OPTIONS FOR DATA PROTECTION DURING LUKS2 RE-ENCRYPTION

LUKS2 provides several options that prioritize performance or data protection during the re-encryption process:

checksum

This is the default mode. It balances data protection and performance.
This mode stores individual checksums of the sectors in the re-encryption area, so the recovery process can detect which sectors LUKS2 already re-encrypted. The mode requires that the block device sector write is atomic.

journal

That is the safest mode but also the slowest. This mode journals the re-encryption area in the binary area, so LUKS2 writes the data twice.

none

This mode prioritizes performance and provides no data protection. It protects the data only against safe process termination, such as the **SIGTERM** signal or the user pressing **Ctrl**+**C**. Any unexpected system crash or application crash might result in data corruption.

You can select the mode using the **--resilience** option of **cryptsetup**.

If a LUKS2 re-encryption process terminates unexpectedly by force, LUKS2 can perform the recovery in one of the following ways:

- Automatically, during the next LUKS2 device open action. This action is triggered either by the **cryptsetup open** command or by attaching the device with **systemd-cryptsetup**.
- Manually, by using the **cryptsetup repair** command on the LUKS2 device.

20.4. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2

This procedure encrypts existing data on a not yet encrypted device using the LUKS2 format. A new LUKS header is stored in the head of the device.

Prerequisites

- The block device contains a file system.
- You have backed up your data.

WARNING

You might lose your data during the encryption process: due to a hardware, kernel, or human failure. Ensure that you have a reliable backup before you start encrypting the data.

Procedure

1. Unmount all file systems on the device that you plan to encrypt. For example:

```
# umount /dev/sdb1
```

2. Make free space for storing a LUKS header. Choose one of the following options that suits your scenario:

   - In the case of encrypting a logical volume, you can extend the logical volume without resizing the file system. For example:

```
# lvextend -L+32M vg00/lv00
```

   - Extend the partition using partition management tools, such as **parted**.
   - Shrink the file system on the device. You can use the **resize2fs** utility for the ext2, ext3, or ext4 file systems. Note that you cannot shrink the XFS file system.

3. Initialize the encryption. For example:

```
# cryptsetup reencrypt \ --encrypt \ --init-only \ --reduce-device-size 32M \ /dev/sdb1
sdb1_encrypted
```

   The command asks you for a passphrase and starts the encryption process.

4. Mount the device:

```
# mount /dev/mapper/sdb1_encrypted /mnt/sdb1_encrypted
```

5. Start the online encryption:

```
# cryptsetup reencrypt --resume-only /dev/sdb1
```

Additional resources

- **cryptsetup(8)**, **lvextend(8)**, **resize2fs(8)**, and **parted(8)** man pages

20.5. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2 WITH A DETACHED HEADER

This procedure encrypts existing data on a block device without creating free space for storing a LUKS header. The header is stored in a detached location, which also serves as an additional layer of security. The procedure uses the LUKS2 encryption format.

Prerequisites

- The block device contains a file system.
- You have backed up your data.

WARNING

You might lose your data during the encryption process: due to a hardware, kernel, or human failure. Ensure that you have a reliable backup before you start encrypting the data.

Procedure

1. Unmount all file systems on the device. For example:

```
# umount /dev/sdb1
```

2. Initialize the encryption:

```
# cryptsetup reencrypt \ --encrypt \ --init-only \ --header /path/to/header \ /dev/sdb1
sdb1_encrypted
```

Replace */path/to/header* with a path to the file with a detached LUKS header. The detached LUKS header has to be accessible so that the encrypted device can be unlocked later.

The command asks you for a passphrase and starts the encryption process.

3. Mount the device:

```
# mount /dev/mapper/sdb1_encrypted /mnt/sdb1_encrypted
```

4. Start the online encryption:

```
# cryptsetup reencrypt --resume-only --header /path/to/header /dev/sdb1
```

Additional resources

- **cryptsetup(8)** man page

20.6. ENCRYPTING A BLANK BLOCK DEVICE USING LUKS2

This procedure provides information about encrypting a blank block device using the LUKS2 format.

Prerequisites

- A blank block device.

Procedure

1. Setup a partition as an encrypted LUKS partition:

```
# cryptsetup luksFormat /dev/sdb1
```

2. Open an encrypted LUKS partition:

```
# cryptsetup open /dev/sdb1 sdb1_encrypted
```

This unlocks the partition and maps it to a new device using the device mapper. This alerts kernel that **device** is an encrypted device and should be addressed through LUKS using the ***/dev/mapper/device_mapped_name*** so as not to overwrite the encrypted data.

3. To write encrypted data to the partition, it must be accessed through the device mapped name. To do this, you must create a file system. For example:

```
# mkfs -t ext4 /dev/mapper/sdb1_encrypted
```

4. Mount the device:

```
# mount /dev/mapper/sdb1_encrypted mount-point
```

Additional resources

- **cryptsetup(8)** man page

20.7. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE ROLE

You can use the **storage** role to create and configure a volume encrypted with LUKS by running an Ansible playbook.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

NOTE

You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to create the volume.

- You have the **rhel-system-roles** package installed on the Ansible controller.
- You have an inventory file detailing the systems on which you want to deploy a LUKS encrypted volume using the storage System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

2. Optional: Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Encrypting block devices using LUKS
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file

CHAPTER 21. MANAGING TAPE DEVICES

A tape device is a magnetic tape where data is stored and accessed sequentially. Data is written to this tape device with the help of a tape drive. There is no need to create a file system in order to store data on a tape device. Tape drives can be connected to a host computer with various interfaces like, SCSI, FC, USB, SATA, and other interfaces.

The following are the different types of tape devices:

- **/dev/st** is a rewinding tape device.
- **/dev/nst** is a non-rewinding tape device. Use non-rewinding devices for daily backups.

Advantages of tape devices:

- Cost efficient
- Resilient against data corruption
- Data retention
- Stable

21.1. INSTALLING TAPE DRIVE MANAGEMENT TOOL

Use the **mt** command to wind the data back and forth. The **mt** utility controls magnetic tape drive operations and the **st** utility is used for SCSI tape driver. This procedure describes how to install the **mt-st** package for tape drive operations.

Procedure

- Install the **mt-st** package:

```
# {PackageManagerCommand} install mt-st
```

Additional resources

- The **mt** man page.
- The **st** man page.

21.2. WRITING TO TAPE DEVICES

This procedure describes how to back up data using the **tar** command. By default, **block size** is 10KB (**bs=10k**) in tape devices. The **-f** device option specifies the tape device file, but this option is not required if you have set the **TAPE** environment variable using the **export TAPE=*/dev/st0*** attribute.

Prerequisites

1. The **mt-st** package is installed. For more information, see Section 21.1, “Installing tape drive management tool”.
2. Load the tape drive:

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

Procedure

1. Check the tape head:

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

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

   Here:

   - the current **file number** is -1.
   - the **block number** defines the tape head. By default, it is set to -1.
   - the **block size** 0 indicates that the tape device does not have a fixed block size.
   - the **Soft error count** indicates the number of encountered errors after executing the mt status command.
   - the **General status bits** explains the stats of the tape device.
   - **DR_OPEN** indicates that the door is open and the tape device is empty. **IM_REP_EN** is the immediate report mode.

2. If the tape device is not empty, specify the tape head:

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

# tar -czf /dev/st0 /etc
```

   This command overwrites the data on a tape device with the content of the */etc* directory

   Optional: To append the data on the tape device:

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

3. Back up the */etc* directory to the tape device:

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

4. View the status of the tape device:

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

Verification steps

- View the list of all files on the tape device:

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

Additional resources

- The **mt** man page.
- The **tar** man page.
- The **st** man page.
- Tape drive media detected as write protected Red Hat Knowlegebase article.
- How to check if tape drives are detected in the system Red Hat Knowlegebase article.

21.3. SWITCHING TAPE HEAD IN TAPE DEVICES

Use the following procedure to switch the tape head in the tape device. While appending the data to tape devices, use the **eod** option to switch the tape head.

Prerequisites

1. The **mt-st** package is installed. For more information, see Section 21.1, "Installing tape drive management tool".
2. Data is written to the tape device. Fore more information, see Section 21.2, "Writing to tape devices".

Procedure

- To go to the end of the data:

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

- To go to the previous record:

```
# mt -f /dev/st0 bsfm 1
```

- To go to the forward record:

```
# mt -f /dev/st0 fsf 1
```

Additional resources

- The **mt** man page.

21.4. RESTORING DATA FROM TAPE DEVICES

To restore data from a tape device, use the **tar** command.

Prerequisites

1. The **mt-st** package is installed. For more information, see Section 21.1, "Installing tape drive management tool".
2. Data is written to the tape device. Fore more information, see Section 21.2, "Writing to tape devices".

Procedure

1. Rewind the tape device:

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

2. Restore the */etc* directory:

```
# tar -xzf /dev/st0 /etc
```

Additional resources

- The **mt** man page.
- The **tar** man page.

21.5. ERASING DATA FROM TAPE DEVICES

To erase data from a tape device, use the **erase** option.

Prerequisites

1. The **mt-st** package is installed. For more information, see Section 21.1, "Installing tape drive management tool".
2. Data is written to the tape device. Fore more information, see Section 21.2, "Writing to tape devices".

Procedure

1. Erase data from the tape device:

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

2. Unload the tape device:

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

Additional resources

- The **mt** man page.

21.6. TAPE COMMANDS

The following are the common **mt** commands:

Table 21.1. mt commands

Command	Description
mt -f /dev/*st0* status	Displays the status of the tape device.
mt -f /dev/*st0* rewind	Rewinds the tape device.
mt -f /dev/*st0* erase	Erases the entire tape.
mt -f /dev/*st0* fsf *n*	Switches the tape head to the forward record. Here, n is an optional file count. If a file count is specified, tape head skips n records.
mt -f /dev/*st0* bsfm *n*	Switches the tape head to the previous record.
mt -f /dev/*st0* eod	Switches the tape head to the end of the data.

CHAPTER 22. REMOVING STORAGE DEVICES

You can safely remove a storage device from a running system, which helps prevent system memory overload and data loss.

NOTE

Before you remove a storage device, you must ensure that you have enough free system memory due to the increased system memory load during an I/O flush. Use the **vmstat 1 100** command to view the system's current memory load. You can also use the **free** command to view the system's free memory. Red Hat does not recommend removing a storage device on a system where:

- Free memory is less than 5% of the total memory in more than 10 samples per 100.
- Swapping is active (non-zero **si** and **so** columns in the **vmstat** command output).

22.1. SAFE REMOVAL OF STORAGE DEVICES

Safely removing a storage device from a running system requires a top-to-bottom approach, starting from the top layer, which typically is an application or a filesystem, and working towards the bottom layer, which is the physical device.

You can use storage devices in multiple ways, and they can have different virtual configurations on top of physical devices. For example, you can configure a virtual device as a multipath device, make it part of a RAID, or you can make it part of an LVM group. Alternatively, devices can be accessed via a filesystem, or they can be accessed directly (as a "raw" device).

Working from the top layer down, you must ensure that the device that you want to remove is not in use, all pending I/O to the device is flushed, and the operating system is not referencing the storage device.

22.2. REMOVING A BLOCK DEVICE

You can safely remove a block device from a running system to help prevent system memory overload and data loss.

WARNING

Rescanning the SCSI bus or performing any other action that changes the state of the operating system's state without following the procedure documented here can cause delays due to I/O timeouts, devices to be removed unexpectedly, or data loss.

Prerequisites

- If you want to remove a multipath device, and you are unable to access its path devices, disable the multipath device's queue:

```
# multipathd disablequeueing map <multipath-device>
```

This enables the device's I/O to fail, allowing the applications that are using the device to shut down.

Procedure

1. Ensure that no other applications or services are using the device that you want to remove.
2. Back up the data from the device that you want to remove.
3. Unmount any filesystems that are mounted on the device using the **umount** command.
4. Remove the device from any **md** RAID array or from any LVM volume that it belongs to. For example, if the device is a member of an LVM group, and it is a multipath device:
   a. Move the data to another device:

      ```
      # pvmove -b /dev/mapper/<from-multipath-device> /dev/mapper/<to-multipath-device>
      ```

   b. Remove the device from the volume group:

      ```
      # vgreduce <volume-group> /dev/mapper/<from-multipath-device>
      ```

   c. (Optional) Remove the LVM metadata from the physical device:

      ```
      # pvremove /dev/mapper/<from-multipath-device>
      ```

5. If you are removing a multipath device:
   a. View all the paths to the device using the following command:

      ```
      # multipath -l
      ```

      The output of this command is required in a later step.

   b. Flush the I/O and remove the multipath device:

      ```
      # multipath -f <multipath-device>
      ```

6. If the device is not configured as a multipath device, or if the device is configured as a multipath device and you have previously passed I/O to its individual paths, flush any outstanding I/O to all device paths that are used:

   ```
   # blockdev --flushbufs <device>
   ```

   This is important for devices accessed directly where the **umount** or **vgreduce** commands do not flush the I/O.

7. If you are removing a SCSI device, complete the following steps:
   a. Remove any reference to the device's path-based name; for example, */dev/sd*, */dev/disk/by-path* or the **major:minor** number, in applications, scripts, or utilities on the system. This ensures that different devices added in the future are not mistaken for the

current device.

b. Remove each path to the device from the SCSI subsystem:

```
# echo 1 > /sys/block/<device-name>/device/delete
```

where ***<device-name>*** is retrieved from the output of the **multipath -l** command in step 5 if the device was previously used as a multipath device.

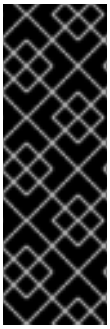
8. Remove the physical device form a running system. Note that the I/O to other devices does not stop when you remove this device.

Additional resources

- The **multipath**, **pvmove**, **vgreduce**, **blockdev**, and **umount** man pages.

CHAPTER 23. 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 a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview.

23.1. SETTING UP STRATIS FILE SYSTEMS

Stratis runs as a service to manage pools of physical storage devices, simplifying local storage management with ease of use while helping you set up and manage complex storage configurations.

23.1.1. What is 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

Additional resources

- Stratis website

23.1.2. Components of a Stratis volume

Learn about the components that comprise 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 /**dev**/**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 /**dev**/**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.

23.1.3. Block devices usable with Stratis

Storage devices that can be used with Stratis.

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

- LUKS
- LVM logical volumes
- MD RAID
- DM Multipath

- iSCSI
- HDDs and SSDs
- NVMe devices

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.

23.1.4. Installing Stratis

Install the required packages for 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
```

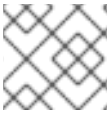
23.1.5. Creating an unencrypted Stratis pool

You can create an unencrypted Stratis pool from one or more block devices.

Prerequisites

- Stratis is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- The block devices on which you are creating a Stratis pool are not in use and are not mounted.
- Each block device on which you are creating a Stratis pool is at least 1 GB.
- On the IBM Z architecture, the /**dev**/**dasd*** block devices must be partitioned. Use the partition in the Stratis pool.

For information on partitioning DASD devices, see Configuring a Linux instance on IBM Z .



NOTE

You cannot encrypt an unencrypted Stratis pool.

Procedure

1. Erase any file system, partition table, or RAID signatures that exist on each block device that you want to use in the Stratis pool:

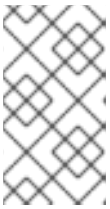
```
# wipefs --all block-device
```

where ***block-device*** is the path to the block device; for example, **/dev/sdb**.

2. Create the new unencrypted Stratis pool on the selected block device:

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

where ***block-device*** is the path to an empty or wiped block device.



NOTE

Specify multiple block devices on a single line:

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

3. Verify that the new Stratis pool was created:

```
# stratis pool list
```

23.1.6. Creating an encrypted Stratis pool

To secure your data, your can create an encrypted Stratis pool from one or more block devices.

When you create an encrypted Stratis pool, the kernel keyring is used as the primary encryption mechanism. After subsequent system reboots this kernel keyring is used to unlock the encrypted Stratis pool.

When creating an encrypted Stratis pool from one or more block devices, note the following:

- Each block device is encrypted using the **cryptsetup** library and implements the **LUKS2** format.
- Each Stratis pool can either have a unique key or share the same key with other pools. These keys are stored in the kernel keyring.
- The block devices that comprise a Stratis pool must be either all encrypted or all unencrypted. It is not possible to have both encrypted and unencrypted block devices in the same Stratis pool.
- Block devices added to the data tier of an encrypted Stratis pool are automatically encrypted.

Prerequisites

- Stratis v2.1.0 or later is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- The block devices on which you are creating a Stratis pool are not in use and are not mounted.
- The block devices on which you are creating a Stratis pool are at least 1GB in size each.
- On the IBM Z architecture, the **/dev/dasd*** block devices must 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. Erase any file system, partition table, or RAID signatures that exist on each block device that you want to use in the Stratis pool:

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

   where ***block-device*** is the path to the block device; for example, /**dev**/**sdb**.

2. If you have not created a key set already, run the following command and follow the prompts to create a key set to use for the encryption.

```
# stratis key set --capture-key key-description
```

   where ***key-description*** is a reference to the key that gets created in the kernel keyring.

3. Create the encrypted Stratis pool and specify the key description to use for the encryption. You can also specify the key path using the **--keyfile-path** option instead instead of using the ***key-description*** option.

```
# stratis pool create --key-desc key-description my-pool block-device
```

   where

   key-description

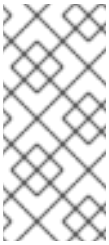
   References the key that exists in the kernel keyring, which you created in the previous step.

   my-pool

   Specifies the name of the new Stratis pool.

   block-device

   Specifies the path to an empty or wiped block device.



NOTE

Specify multiple block devices on a single line:

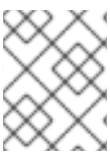
```
# stratis pool create --key-desc key-description my-pool block-device-1
block-device-2
```

4. Verify that the new Stratis pool was created:

```
# stratis pool list
```

23.1.7. Binding a Stratis pool to NBDE

Binding an encrypted Stratis pool to Network Bound Disk Encryption (NBDE) requires a Tang server. When a system containing the Stratis pool reboots, it connects with the Tang server to automatically unlock the encrypted pool without you having to provide the kernel keyring description.



NOTE

Binding a Stratis pool to a supplementary Clevis encryption mechanism does not remove the primary kernel keyring encryption.

Prerequisites

- Stratis v2.3.0 or later is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- You have created an encrypted Stratis pool, and you have the key description of the key that was used for the encryption. For more information, see Creating an encrypted Stratis pool .
- You can connect to the Tang server. For more information, see Deploying a Tang server with SELinux in enforcing mode

Procedure

- Bind an encrypted Stratis pool to NBDE:

```
# stratis pool bind nbde my-pool key-description tang-server
```

where

my-pool

Specifies the name of the encrypted Stratis pool.

key-description

References the key that exists in the kernel keyring, which was generated when you created the encrypted Stratis pool.

tang-server

Specifies the IP address or URL of the Tang server.

Additional resources

- Configuring automated unlocking of encrypted volumes using policy-based decryption

23.1.8. Binding a Stratis pool to TPM

When you bind an encrypted Stratis pool to the Trusted Platform Module (TPM) 2.0, when the system containing the pool reboots, the pool is automatically unlocked without you having to provide the kernel keyring description.

Prerequisites

- Stratis v2.3.0 or later is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- You have created an encrypted Stratis pool. For more information, see Creating an encrypted Stratis pool.

Procedure

- Bind an encrypted Stratis pool to TPM:

```
# stratis pool bind tpm my-pool key-description
```

where

my-pool

Specifies the name of the encrypted Stratis pool.

key-description

References the key that exists in the kernel keyring, which was generated when you created the encrypted Stratis pool.

23.1.9. Unlocking an encrypted Stratis pool with kernel keyring

After a system reboot, your encrypted Stratis pool or the block devices that comprise it might not be visible. You can unlock the pool using the kernel keyring that was used to encrypt the pool.

Prerequisites

- Stratis v2.1.0 is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- You have created an encrypted Stratis pool. For more information, see Creating an encrypted Stratis pool.

Procedure

1. Re-create the key set using the same key description that was used previously:

```
# stratis key set --capture-key key-description
```

where *key-description* references the key that exists in the kernel keyring, which was generated when you created the encrypted Stratis pool.

2. Unlock the Stratis pool and the block device that comprise it:

```
# stratis pool unlock keyring
```

3. Verify that the Stratis pool is visible:

```
# stratis pool list
```

23.1.10. Unlocking an encrypted Stratis pool with Clevis

After a system reboot, your encrypted Stratis pool or the block devices that comprise it might not be visible. You can unlock an encrypted Stratis pool with the supplementary encryption mechanism that the pool is bound to.

Prerequisites

- Stratis v2.3.0 or later is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- You have created an encrypted Stratis pool. For more information, see Creating an encrypted Stratis pool.

- The encrypted Stratis pool is bound to a supported, supplementary encryption mechanism. For more information, see Binding an encrypted Stratis pool to NBDE or Binding an encrypted Stratis pool to TPM.

Procedure

1. Unlock the Stratis pool and the block devices that comprise it:

```
# stratis pool unlock clevis
```

2. Verify that the Stratis pool is visible:

```
# stratis pool list
```

23.1.11. Unbinding a Stratis pool from supplementary encryption

When you unbind an encrypted Stratis pool from a supported supplementary encryption mechanism, the primary kernel keyring encryption remains in place.

Prerequisites

- Stratis v2.3.0 or later is installed on your system. For more information, see Installing Stratis.
- You have created an encrypted Stratis pool. For more information, see Creating an encrypted Stratis pool.
- The encrypted Stratis pool is bound to a supported supplementary encryption mechanism.

Procedure

- Unbind an encrypted Stratis pool from a supplementary encryption mechanism:

```
# stratis pool unbind clevis my-pool
```

where

my-pool specifies the name of the Stratis pool you want to unbind.

Additional resources

- Binding an encrypted Stratis pool to NBDE
- Binding an encrypted Stratis pool to TPM

23.1.12. Creating a Stratis file system

Create a Stratis file system on an existing Stratis pool.

Prerequisites

- Stratis is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.

- You have created a Stratis pool. See Creating an unencrypted Stratis pool or Creating an encrypted Stratis pool.

Procedure

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

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

where

my-pool
Specifies the name of the Stratis pool.

my-fs
Specifies an arbitrary name for the file system.

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

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

Additional resources

- Mounting a Stratis file system.

23.1.13. Mounting a Stratis file system

Mount an existing Stratis file system to access the content.

Prerequisites

- Stratis is installed. For more information, see Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis file system. For more information, see Creating a Stratis filesystem.

Procedure

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

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

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

Additional resources

- Creating a Stratis file system.

23.1.14. 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 Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis file system. See Creating a Stratis filesystem.

Procedure

1. Determine the UUID attribute of the file system:

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

For example:

Example 23.1. 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 23.2. 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.

23.2. 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.

23.2.1. Components of a Stratis volume

Learn about the components that comprise 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 /**dev**/**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 /**dev**/**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.

23.2.2. 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 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

- **stratis(8)** man page

23.2.3. Related information

- The *Stratis Storage* website

23.3. 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.

23.3.1. 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

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

23.3.2. 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 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  /dev/stratis/my-pool/my-fs
```

Additional resources

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

23.3.3. Related information

- The *Stratis Storage* website.

23.4. 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.

23.4.1. 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.

23.4.2. 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 Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis file system. See Creating a Stratis filesystem.

Procedure

- To create a Stratis snapshot, use:

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

Additional resources

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

23.4.3. 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 Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Creating a Stratis filesystem.

Procedure

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

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

Additional resources

- Mounting a Stratis file system.

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

23.4.4. 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 Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis snapshot. See 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 /dev/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 /dev/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

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

23.4.5. Removing a Stratis snapshot

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

Prerequisites

- Stratis is installed. See Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Creating a Stratis snapshot.

Procedure

1. Unmount the snapshot:

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

2. Destroy the snapshot:

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

Additional resources

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

23.4.6. Related information

- The *Stratis Storage* website.

23.5. REMOVING STRATIS FILE SYSTEMS

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

23.5.1. Components of a Stratis volume

Learn about the components that comprise 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 **/dev/stratis/*my-pool*/** directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

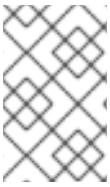
Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin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

23.5.2. Removing a Stratis file system

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

Prerequisites

- Stratis is installed. See Installing Stratis.
- The **stratisd** service is running.
- You have created a Stratis file system. See Creating a Stratis filesystem.

Procedure

1. Unmount the file system:

```
# umount /dev/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

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

23.5.3. Removing a Stratis pool

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

Prerequisites

- Stratis is installed. See Installing Stratis.

- The **stratisd** service is running.
- You have created a Stratis pool:
  - To create an unencrypted pool, see Creating an unencrypted Stratis pool
  - To create an encrypted pool, see Creating an encrypted Stratis pool .

Procedure

1. List file systems on the pool:

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

2. Unmount all file systems on the pool:

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

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

23.5.4. Related information

- The *Stratis Storage* website.