Red Hat Enterprise Linux 10

Configuring and managing logical volumes

Configuring and managing LVM

Red Hat Enterprise Linux 10 Configuring and managing logical volumes

Configuring and managing LVM

Legal Notice

Abstract

Logical Volume Manager (LVM) is a storage virtualization software designed to enhance the management and flexibility of physical storage devices. By abstracting the physical hardware, LVM allows you to dynamically create, resize, and remove of virtual storage devices. Within this framework, physical volumes (PVs) represent the raw storage devices that are grouped together to form a volume group (VG). Within this VG, LVM allocates space to create a logical volume (LV). An LV is a virtual block storage device that a file system, database, or application can use.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.

2. Click **Create** in the top navigation bar

3. Enter a descriptive title in the **Summary** field.

4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.

5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. OVERVIEW OF LOGICAL VOLUME MANAGEMENT

Logical Volume Manager (LVM) creates a layer of abstraction over physical storage, which helps you to create logical storage volumes. This offers more flexibility compared to direct physical storage usage.

In addition, the hardware storage configuration is hidden from the software so you can resize and move it without stopping applications or unmounting file systems. This can reduce operational costs.

1.1. LVM ARCHITECTURE

LVM creates a layer of abstraction over physical storage, which helps you to create logical storage volumes. This offers more flexibility compared to direct physical storage usage.

The following are the components of LVM:

Physical volume

A physical volume (PV) is a partition or whole disk designated for LVM use.

Volume group

A volume group (VG) is a collection of physical volumes (PVs), which creates a pool of disk space out of which you can allocate logical volumes.

Logical volume

A logical volume represents a usable storage device.

The following diagram illustrates the components of LVM:

Figure 1.1. LVM logical volume components
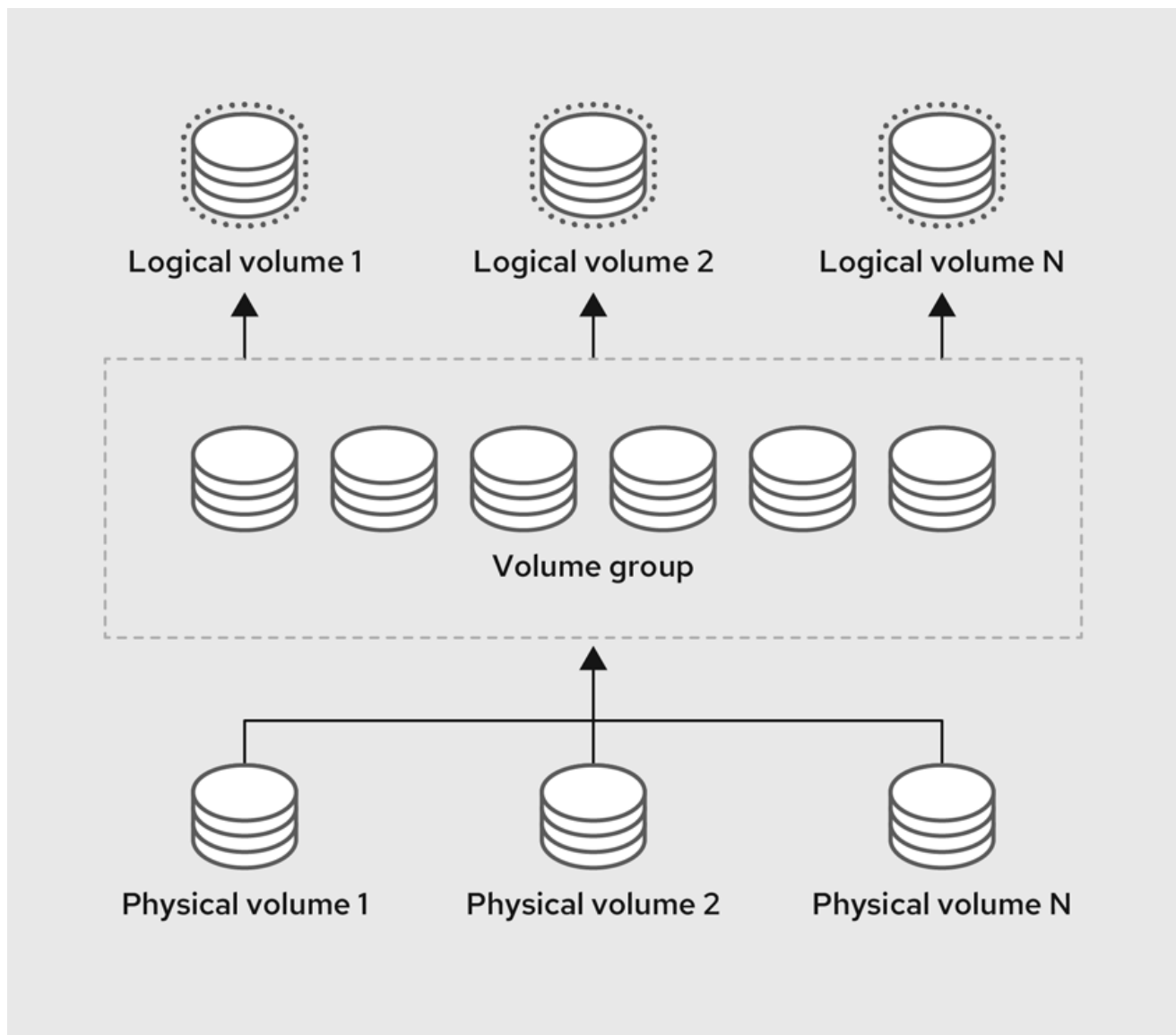
Additional resources

- Managing LVM physical volumes

- Managing LVM volume groups

- Basic logical volume management

- Advanced logical volume management

1.2. ADVANTAGES OF LVM

Logical Volume Manager (LVM) provides a flexible layer of abstraction over physical storage that enables dynamic storage management. LVM allows you to resize, move, and manage storage without downtime while providing advanced features like snapshots and striping.

Logical volumes provide the following advantages over using physical storage directly:

Flexible capacity

When using logical volumes, you can aggregate devices and partitions into a single logical volume. With this functionality, file systems can extend across multiple devices as though they were a single, large one.

Convenient device naming

Logical storage volumes can be managed with user-defined and custom names.

Resizeable storage volumes

You can extend logical volumes or reduce logical volumes in size with simple software commands, without reformatting and repartitioning the underlying devices.

Online data relocation

To deploy newer, faster, or more resilient storage subsystems, you can move data while your system is active using the **pvmove** command. Data can be rearranged on disks while the disks are in use. For example, you can empty a hot-swappable disk before removing it.
For more information on how to migrate the data, see the **pvmove(8)** man page on your system.

Striped Volumes

You can create a logical volume that stripes data across two or more devices. This can increase throughput.

RAID volumes

Logical volumes provide a convenient way to configure RAID for your data. This delivers protection against device failure and improves performance.

Volume snapshots

You can take snapshots, which is a point-in-time copy of logical volumes for consistent backups or to test the effect of changes without affecting the real data.

Thin volumes

Logical volumes can be thin-provisioned. This allows you to create logical volumes that are larger than the available physical space.

Caching

Caching uses fast devices, like SSDs, to cache data from logical volumes, boosting performance.

Additional resources

- Resizing logical volumes

- Removing physical volumes from a volume group

- Creating a striped logical volume

- Configuring RAID logical volumes

- Managing logical volume snapshots

- Creating a thin logical volume

- Caching logical volumes

CHAPTER 2. MANAGING LVM PHYSICAL VOLUMES

A physical volume (PV) is a physical storage device or a partition on a storage device that LVM uses.

During the initialization process, an LVM disk label and metadata are written to the device, which allows LVM to track and manage it as part of the logical volume management scheme.
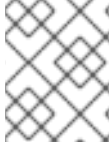
> **NOTE**
>
> You cannot increase the size of the metadata after the initialization. If you need larger metadata, you must set the appropriate size during the initialization process.

When initialization process is complete, you can allocate the PV to a volume group (VG). You can divide this VG into logical volumes (LVs), which are the virtual block devices that operating systems and applications can use for storage.

To ensure optimal performance, partition the whole disk as a single PV for LVM use.

2.1. CREATING AN LVM PHYSICAL VOLUME

You can use the **pvcreate** command to initialize a physical volume LVM usage.

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

Procedure

1. Identify the storage device you want to use as a physical volume. To list all available storage devices, use:

   ```
   $ lsblk
   ```

2. Create an LVM physical volume:

   ```
   # pvcreate /dev/sdb
   ```

   Replace /dev/sdb with the name of the device you want to initialize as a physical volume.

Verification

- Display the created physical volume:

   ```
   # pvs

    PV        VG  Fmt  Attr PSize  PFree
    /dev/sdb      lvm2 a--  28.87g 13.87g
   ```

2.2. RESIZING PHYSICAL VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** system role, you can resize LVM physical volumes after resizing the underlying storage or disks from outside of the host. For example, you increased the size of a virtual disk and want to use the extra space in an existing LVM.

Prerequisites

- You have prepared the control node and the managed nodes .

- You are logged in to the control node as a user who can run playbooks on the managed nodes.

- The account you use to connect to the managed nodes has **sudo** permissions for these nodes.

- The size of the underlying block storage has been changed.

Procedure

1. Create a playbook file, for example, ~/**playbook.yml**, with the following content:

   ```
   ---
   - name: Manage local storage
     hosts: managed-node-01.example.com
     tasks:
       - name: Resize LVM PV size
         ansible.builtin.include_role:
           name: redhat.rhel_system_roles.storage
         vars:
           storage_pools:
             - name: myvg
               disks: ["sdf"]
               type: lvm
               grow_to_fill: true
   ```

   The settings specified in the example playbook include the following:

   grow_to_fill

   > **true** The role automatically expands the storage volume to use any new capacity on the disk.
   > **false** The role leaves the storage volume at its current size, even if the underlying disk has grown.

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

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

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

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

Verification

1. Verify the **grow_to_fill** setting works as expected. Prepare a test PV and VG:

   ```
   # pvcreate /dev/sdf
   # vgcreate myvg /dev/sdf
   ```

2. Check and record the initial physical volume size:

   ```
   # pvs
   ```

3. Edit the playbook to set **grow_to_fill: false** and run the playbook.

4. Check the volume size and verify that it remained unchanged.

5. Edit the playbook to set **grow_to_fill: true** and re-run the playbook.

6. Check the volume size and verify that it has expanded.

2.3. REMOVING LVM PHYSICAL VOLUMES

You can use the **pvremove** command to remove a physical volume for LVM usage.

Prerequisites

- Administrative access.

Procedure

1. List the physical volumes to identify the device you want to remove:

   ```
   # pvs

     PV         VG Fmt  Attr PSize  PFree
     /dev/sdb      lvm2 ---  28.87g 28.87g
   ```

2. Remove the physical volume:

   ```
   # pvremove /dev/sdb
   ```

   Replace */dev/sdb* with the name of the device associated with the physical volume.

NOTE

If your physical volume is part of the volume group, you need to remove it from the volume group first.

- If you volume group contains more that one physical volume, use the **vgreduce** command:

  ```
  # vgreduce VolumeGroupName /dev/sdb
  ```

  Replace *VolumeGroupName* with the name of the volume group. Replace */dev/sdb* with the name of the device.

- If your volume group contains only one physical volume, use the **vgremove** command:

  ```
  # vgremove VolumeGroupName
  ```

  Replace *VolumeGroupName* with the name of the volume group.

Verification

- Verify the physical volume is removed:

  ```
  # pvs
  ```

2.4. CREATING LOGICAL VOLUMES IN THE WEB CONSOLE

Logical volumes act as physical drives. You can use the RHEL 10 web console to create LVM logical volumes in a volume group.

Prerequisites

- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .

- The **cockpit-storaged** package is installed on your system.

- The volume group is created.

Procedure

1. Log in to the RHEL 10 web console.

2. Click **Storage**.

3. In the **Storage** table, click the volume group in which you want to create logical volumes.

4. On the **Logical volume group** page, scroll to the **LVM2 logical volumes** section and click **Create new logical volume**.

5. In the **Name** field, enter a name for the new logical volume. Do not include spaces in the name.

6. In the **Purpose** drop-down menu, select **Block device for filesystems**.

This configuration enables you to create a logical volume with the maximum volume size which is equal to the sum of the capacities of all drives included in the volume group.

7. Define the size of the logical volume. Consider:

   - How much space the system using this logical volume will need.

   - How many logical volumes you want to create.

   You do not have to use the whole space. If necessary, you can grow the logical volume later.

8. Click **Create**.
   The logical volume is created. To use the logical volume you must format and mount the volume.

Verification

- On the **Logical volume** page, scroll to the **LVM2 logical volumes** section and verify whether the new logical volume is listed.

2.5. FORMATTING LOGICAL VOLUMES IN THE WEB CONSOLE

Logical volumes act as physical drives. To use them, you must format them with a file system.

> **WARNING**
>
> Formatting logical volumes erases all data on the volume.

The file system you select determines the configuration parameters you can use for logical volumes. For example, the XFS file system does not support shrinking volumes.

Prerequisites

- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .

- The **cockpit-storaged** package is installed on your system.

- The logical volume created.

- You have root access privileges to the system.

Procedure

1. Log in to the RHEL 10 web console.

2. Click **Storage**.

3. In the **Storage** table, click the volume group in the logical volumes is created.

4. On the **Logical volume group** page, scroll to the **LVM2 logical volumes** section.

5. Click the menu button, ⋮ , next to the volume group you want to format.

6. From the drop-down menu, select **Format**.

7. In the **Name** field, enter a name for the file system.

8. In the **Mount Point** field, add the mount path.

9. In the **Type** drop-down menu, select a file system:

   - **XFS** file system supports large logical volumes, switching physical drives online without outage, and growing an existing file system. Leave this file system selected if you do not have a different strong preference.
     XFS does not support reducing the size of a volume formatted with an XFS file system.

   - **ext4** file system supports:

     ○ Logical volumes

     ○ Switching physical drives online without an outage

     ○ Growing a file system

     ○ Shrinking a file system

10. Select the **Overwrite existing data with zeros** checkbox if you want the RHEL web console to rewrite the whole disk with zeros. This option is slower because the program has to go through the whole disk, but it is more secure. Use this option if the disk includes any data and you need to overwrite it.
    If you do not select the **Overwrite existing data with zeros** checkbox, the RHEL web console rewrites only the disk header. This increases the speed of formatting.

11. From the **Encryption** drop-down menu, select the type of encryption if you want to enable it on the logical volume.
    You can select a version with either the LUKS1 (Linux Unified Key Setup) or LUKS2 encryption, which allows you to encrypt the volume with a passphrase.

12. In the **At boot** drop-down menu, select when you want the logical volume to mount after the system boots.

13. Select the required **Mount options**.

14. Format the logical volume:

   - If you want to format the volume and immediately mount it, click **Format and mount**.

   - If you want to format the volume without mounting it, click **Format only**.
     Formatting can take several minutes depending on the volume size and which formatting options are selected.

Verification

1. On the **Logical volume group** page, scroll to the **LVM2 logical volumes** section and click the logical volume to check the details and additional options.

2. If you selected the **Format only** option, click the menu button at the end of the line of the logical volume, and select **Mount** to use the logical volume.

2.6. RESIZING LOGICAL VOLUMES IN THE WEB CONSOLE

You can extend or reduce logical volumes in the RHEL 10 web console. The example procedure demonstrates how to grow and shrink the size of a logical volume without taking the volume offline.

> **WARNING**
>
> You cannot reduce volumes that contains GFS2 or XFS filesystem.

Prerequisites

- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .

- The **cockpit-storaged** package is installed on your system.

- An existing logical volume containing a file system that supports resizing logical volumes.

Procedure

1. Log in to the RHEL 10 web console.

2. Click **Storage**.

3. In the **Storage** table, click the volume group in the logical volumes is created.

4. On the **Logical volume group** page, scroll to the **LVM2 logical volumes** section and click the menu button, ⋮ , next to volume group you want to resize.

5. From the menu, select **Grow** or **Shrink** to resize the volume:

   - Growing the Volume:

     a. Select **Grow** to increase the size of the volume.

     b. In the **Grow logical volume** dialog box, adjust the size of the logical volume.

     c. Click **Grow**.
        LVM grows the logical volume without causing a system outage.

   - Shrinking the Volume:

     a. Select **Shrink** to reduce the size of the volume.

     b. In the **Shrink logical volume** dialog box, adjust the size of the logical volume.

     c. Click **Shrink**.
        LVM shrinks the logical volume without causing a system outage.

2.7. ADDITIONAL RESOURCES

- Creating a partition table on a disk with parted

CHAPTER 3. MANAGING LVM VOLUME GROUPS

You can create and use volume groups (VGs) to manage and resize multiple physical volumes (PVs) combined into a single storage entity.

Extents are the smallest units of space that you can allocate in LVM. Physical extents (PE) and logical extents (LE) has the default size of 4 MiB that you can configure. All extents have the same size.

When you create a logical volume (LV) within a VG, LVM allocates physical extents on the PVs. The logical extents within the LV correspond one-to-one with physical extents in the VG. You do not need to specify the PEs to create LVs. LVM will locate the available PEs and piece them together to create a LV of the requested size.

Within a VG, you can create multiple LVs, each acting like a traditional partition but with the ability to span across physical volumes and resize dynamically. VGs can manage the allocation of disk space automatically.

3.1. CREATING AN LVM VOLUME GROUP

You can use the **vgcreate** command to create a volume group (VG). You can adjust the extent size for very large or very small volumes to optimize performance and storage efficiency. You can specify the extent size when creating a VG. To change the extent size you must re-create the volume group.

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

- One or more physical volumes are created. For more information about creating physical volumes, see Creating an LVM physical volume .

Procedure

1. List and identify the PV that you want to include in the VG:

   # pvs

2. Create a VG:

   # vgcreate *VolumeGroupName PhysicalVolumeName1 PhysicalVolumeName2*

   Replace *VolumeGroupName* with the name of the volume group that you want to create. Replace *PhysicalVolumeName* with the name of the PV.

   To specify the extent size when creating a VG, use the **-s *ExtentSize*** option. Replace *ExtentSize* with the size of the extent. If you provide no size suffix, the command defaults to MB.

Verification

- Verify that the VG is created:

   # vgs

```
VG             #PV #LV #SN Attr   VSize  VFree
VolumeGroupName   2   0   0 wz--n- 28.87g 28.87g
```

3.2. CREATING VOLUME GROUPS IN THE WEB CONSOLE

Create volume groups from one or more physical drives or other storage devices.

Logical volumes are created from volume groups. Each volume group can include multiple logical volumes.

Prerequisites

- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .

- The **cockpit-storaged** package is installed on your system.

- Physical drives or other types of storage devices from which you want to create volume groups.

Procedure

1. Log in to the RHEL 10 web console.

2. Click **Storage**.

3. In the **Storage** table, click the menu button.

4. From the drop-down menu, select **Create LVM2 volume group**.

5. In the **Name** field, enter a name for the volume group. The name must not include spaces.

6. Select the drives you want to combine to create the volume group.
   The RHEL web console displays only unused block devices. If you do not see your device in the list, make sure that it is not being used by your system, or format it to be empty and unused. Used devices include, for example:

   - Devices formatted with a file system

   - Physical volumes in another volume group

   - Physical volumes being a member of another software RAID device

7. Click **Create**.
   The volume group is created.

Verification

- On the **Storage** page, check whether the new volume group is listed in the   **Storage** table.

3.3. RENAMING AN LVM VOLUME GROUP

You can use the **vgrename** command to rename a volume group (VG).

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

- One or more physical volumes are created. For more information about creating physical volumes, see Creating an LVM physical volume .

- The volume group is created. For more information about creating volume groups, see Creating an LVM volume group.

Procedure

1. List and identify the VG that you want to rename:

   ```
   # vgs
   ```

2. Rename the VG:

   ```
   # vgrename OldVolumeGroupName NewVolumeGroupName
   ```

   Replace *OldVolumeGroupName* with the name of the VG. Replace *NewVolumeGroupName* with the new name for the VG.

Verification

- Verify that the VG has a new name:

   ```
   # vgs

   VG                 #PV #LV #SN Attr   VSize  VFree
   NewVolumeGroupName   1   0   0 wz--n- 28.87g 28.87g
   ```

3.4. EXTENDING AN LVM VOLUME GROUP

You can use the **vgextend** command to add physical volumes (PVs) to a volume group (VG).

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

- One or more physical volumes are created. For more information about creating physical volumes, see Creating an LVM physical volume .

- The volume group is created. For more information about creating volume groups, see Creating an LVM volume group.

Procedure

1. List and identify the VG that you want to extend:

   ```
   # vgs
   ```

2. List and identify the PVs that you want to add to the VG:

   ```
   # pvs
   ```

3. Extend the VG:

   ```
   # vgextend VolumeGroupName PhysicalVolumeName
   ```

   Replace *VolumeGroupName* with the name of the VG. Replace *PhysicalVolumeName* with the name of the PV.

Verification

- Verify that the VG now includes the new PV:

  ```
  # pvs

  PV        VG             Fmt  Attr PSize  PFree
  /dev/sda   VolumeGroupName lvm2 a--  28.87g 28.87g
  /dev/sdd   VolumeGroupName lvm2 a--   1.88g  1.88g
  ```

3.5. COMBINING LVM VOLUME GROUPS

You can combine two existing volume groups (VGs) with the **vgmerge** command. The source volume will be merged into the destination volume.

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

- One or more physical volumes are created. For more information about creating physical volumes, see Creating an LVM physical volume .

- Two or more volume groups are created. For more information about creating volume groups, see Creating an LVM volume group .

Procedure

1. List and identify the VG that you want to merge:

   ```
   # vgs

   VG              #PV #LV #SN Attr   VSize  VFree
   VolumeGroupName1  1   0   0 wz--n- 28.87g 28.87g
   VolumeGroupName2  1   0   0 wz--n-  1.88g  1.88g
   ```

2. Merge the source VG into the destination VG:

   ```
   # vgmerge VolumeGroupName2 VolumeGroupName1
   ```

Replace *VolumeGroupName2* with the name of the destination VG. Replace *VolumeGroupName1* with the name of the source VG.

Verification

- Verify that the VG now includes the new PV:

  ```
  # vgs

  VG              #PV #LV #SN Attr   VSize  VFree
  VolumeGroupName2   2   0   0 wz--n- 30.75g 30.75g
  ```

3.6. REMOVING PHYSICAL VOLUMES FROM A VOLUME GROUP

To remove unused physical volumes (PVs) from a volume group (VG), use the **vgreduce** command. The **vgreduce** command shrinks a volume group's capacity by removing one or more empty physical volumes. This frees those physical volumes to be used in different volume groups or to be removed from the system.

Procedure

1. If the physical volume is still being used, migrate the data to another physical volume from the same volume group:

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

2. If there are not enough free extents on the other physical volumes in the existing volume group:

   a. Create a new physical volume from */dev/vdb4*:

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

   b. Add the newly created physical volume to the volume group:

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

   c. Move the data from */dev/vdb3* to */dev/vdb4*:

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

3. Remove the physical volume */dev/vdb3* from the volume group:

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

Verification

- Verify that the */dev/vdb3* physical volume is removed from the *VolumeGroupName* volume group:

```
# pvs
  PV         VG              Fmt  Attr PSize     PFree     Used
  /dev/vdb1 VolumeGroupName lvm2 a--   1020.00m  0         1020.00m
  /dev/vdb2 VolumeGroupName lvm2 a--   1020.00m  0         1020.00m
  /dev/vdb3                 lvm2 a--   1020.00m  1008.00m  12.00m
  /dev/vdb4 VolumeGroupName lvm2 a--   1020.00m  0         1020.00m
```

3.7. SPLITTING AN LVM VOLUME GROUP

If there is enough unused space on the physical volumes, a new volume group can be created without adding new disks.

In the initial setup, the volume group *VolumeGroupName1* consists of */dev/vdb1*, */dev/vdb2*, and */dev/vdb3*. After completing this procedure, the volume group *VolumeGroupName1* will consist of */dev/vdb1* and */dev/vdb2*, and the second volume group, *VolumeGroupName2*, will consist of */dev/vdb3*.

Prerequisites

- You have sufficient space in the volume group. Use the **vgscan** command to determine how much free space is currently available in the volume group.

- Depending on the free capacity in the existing physical volume, move all the used physical extents to other physical volume using the **pvmove** command. For more information, see Removing physical volumes from a volume group .

Procedure

1. Split the existing volume group *VolumeGroupName1* to the new volume group *VolumeGroupName2*:

   ```
   # vgsplit VolumeGroupName1 VolumeGroupName2 /dev/vdb3
     Volume group "VolumeGroupName2" successfully split from "VolumeGroupName1"
   ```

   > **NOTE**
   >
   > If you have created a logical volume using the existing volume group, use the following command to deactivate the logical volume:
   >
   > ```
   > # lvchange -a n /dev/VolumeGroupName1/LogicalVolumeName
   > ```

2. View the attributes of the two volume groups:

   ```
   # vgs
     VG              #PV #LV #SN Attr   VSize  VFree
     VolumeGroupName1   2   1   0 wz--n- 34.30G 10.80G
     VolumeGroupName2   1   0   0 wz--n- 17.15G 17.15G
   ```

Verification

Verification

- Verify that the newly created volume group *VolumeGroupName2* consists of */dev/vdb3* physical volume:

```
# pvs
PV          VG                Fmt    Attr   PSize      PFree      Used
/dev/vdb1  VolumeGroupName1   lvm2   a--    1020.00m   0          1020.00m
/dev/vdb2  VolumeGroupName1   lvm2   a--    1020.00m   0          1020.00m
/dev/vdb3  VolumeGroupName2   lvm2   a--    1020.00m   1008.00m   12.00m
```

3.8. MOVING A VOLUME GROUP TO ANOTHER SYSTEM

You can move an entire LVM volume group (VG) to another system by using the following commands:

vgexport

Use this command on an existing system to make an inactive VG inaccessible to the system. Once the VG is inaccessible, you can detach its physical volumes (PV).

vgimport

Use this command on the other system to make the VG, which was inactive in the old system, accessible in the new system.

Prerequisites

- No users are accessing files on the active volumes in the volume group that you are moving.

Procedure

1. Unmount the *LogicalVolumeName* logical volume:

   ```
   # umount /mnt/LogicalVolumeName
   ```

2. Deactivate all logical volumes in the volume group, which prevents any further activity on the volume group:

   ```
   # vgchange -an VolumeGroupName
     0 logical volume(s) in volume group "VolumeGroupName" now active
   ```

3. Export the volume group to prevent it from being accessed by the system from which you are removing it:

   ```
   # vgexport VolumeGroupName
     Volume group "VolumeGroupName" successfully exported
   ```

4. View the exported volume group:

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

5. Shut down your system and unplug the disks that make up the volume group and connect them to the new system.

6. Plug the disks into the new system and import the volume group to make it accessible to the new system:

   ```
   # vgimport VolumeGroupName
   ```

   > **NOTE**
   >
   > You can use the **--force** argument of the **vgimport** command to import volume groups that are missing physical volumes and subsequently run the **vgreduce --removemissing** command.

7. Activate the volume group:

   ```
   # vgchange -ay VolumeGroupName
   ```

8. Mount the file system to make it available for use:

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

3.9. REMOVING LVM VOLUME GROUPS

You can remove an existing volume group by using the **vgremove** command. Only volume groups that do not contain logical volumes can be removed.

Prerequisites

- Administrative access.

Procedure

1. Ensure the volume group does not contain logical volumes:

   ```
   # vgs -o vg_name,lv_count VolumeGroupName

     VG              #LV
     VolumeGroupName    0
   ```

   Replace *VolumeGroupName* with the name of the volume group.

2. Remove the volume group:

   ```
   # vgremove VolumeGroupName
   ```

   Replace *VolumeGroupName* with the name of the volume group.

3.10. REMOVING LVM VOLUME GROUPS IN A CLUSTER ENVIRONMENT

In a cluster environment, LVM uses the **lockspace** <qualifier> to coordinate access to volume groups shared among multiple machines. You must stop the **lockspace** before removing a volume group to make sure no other node is trying to access or modify it during the removal process.

Prerequisites

- Administrative access.

- The volume group contains no logical volumes.

Procedure

1. Ensure the volume group does not contain logical volumes:

   ```
   # vgs -o vg_name,lv_count VolumeGroupName

   VG              #LV
   VolumeGroupName   0
   ```

   Replace *VolumeGroupName* with the name of the volume group.

2. Stop the **lockspace** on all nodes except the node where you are removing the volume group:

   ```
   # vgchange --lockstop VolumeGroupName
   ```

   Replace *VolumeGroupName* with the name of the volume group and wait for the lock to stop.

3. Remove the volume group:

   ```
   # vgremove VolumeGroupName
   ```

   Replace *VolumeGroupName* with the name of the volume group.

CHAPTER 4. BASIC LOGICAL VOLUME MANAGEMENT

Logical Volume Manager (LVM) creates a layer of abstraction over physical storage, which helps you to create logical storage volumes. This offers more flexibility compared to direct physical storage usage.

With LVM, you can do the following tasks:

- Create new logical volumes to extend storage capabilities of your system

- Extend existing volumes and thin pools to accommodate growing data

- Rename volumes for better organization

- Reduce volumes to free up unused space

- Safely remove volumes when they are no longer needed

- Activate or deactivate volumes to control the system's access to its data :leveloffset: +1

CHAPTER 5. OVERVIEW OF LOGICAL VOLUME FEATURES

With the Logical Volume Manager (LVM), you can manage disk storage in a flexible and efficient way that traditional partitioning schemes cannot offer. Below is a summary of key LVM features that are used for storage management and optimization.

Concatenation

Concatenation involves combining space from one or more physical volumes into a singular logical volume, effectively merging the physical storage.

Striping

Striping optimizes data I/O efficiency by distributing data across multiple physical volumes. This method enhances performance for sequential reads and writes by allowing parallel I/O operations.

RAID

LVM supports RAID levels 0, 1, 4, 5, 6, and 10. When you create a RAID logical volume, LVM creates a metadata subvolume that is one extent in size for every data or parity subvolume in the array.

Thin provisioning

Thin provisioning enables the creation of logical volumes that are larger than the available physical storage. With thin provisioning, the system dynamically allocates storage based on actual usage instead of allocating a predetermined amount upfront.

Snapshots

With LVM snapshots, you can create point-in-time copies of logical volumes. A snapshot starts empty. As changes occur on the original logical volume, the snapshot captures the pre-change states through copy-on-write (CoW), growing only with changes to preserve the state of the original logical volume.

Caching

LVM supports the use of fast block devices, such as SSD drives as write-back or write-through caches for larger slower block devices. Users can create cache logical volumes to improve the performance of their existing logical volumes or create new cache logical volumes composed of a small and fast device coupled with a large and slow device.

5.1. CREATING LOGICAL VOLUMES

LVM provides a flexible approach to handling disk storage by abstracting the physical layer into logical volumes that can be created and adjusted based on your needs.

5.1.1. Creating a linear (thick) logical volume

With linear logical volumes (LVs), you can merge multiple physical storage units into one virtual storage space. You can easily expand or reduce linear LVs to accommodate the data requirements.

Prerequisites

- Administrative access.

- The **lvm2** package is installed.

- The volume group is created. For more information, see Creating an LVM volume group .

Procedure

1. List the names of volume groups and their size:

```
# vgs -o vg_name,vg_size

VG             VSize
VolumeGroupName 30.75g
```

2. Create a linear LV:

```
# lvcreate --name LogicalVolumeName --size VolumeSize VolumeGroupName
```

Replace *LogicalVolumeName* with the name of the LV. Replace *VolumeSize* with the size for the LV. If no size suffix is provided the command defaults to MB. Replace *VolumeGroupName* with the name of the volume group.

Verification

- Verify that the linear LV is created:

```
# lvs -o lv_name,seg_type

LV              Type
LogicalVolumeName    linear
```

5.1.2. Creating or resizing a logical volume by using the storage RHEL system role

You can use the **storage** RHEL system role to create and resize LVM logical volumes. The role automatically creates volume groups if they do not exist.

Use the **storage** role to perform the following tasks:

- To create an LVM logical volume in a volume group consisting of many disks

- To resize an existing file system on LVM

- To express an LVM volume size in percentage of the pool's total size

If the volume group does not exist, the role creates it. If a logical volume exists in the volume group, it is resized if the size does not match what is specified in the playbook.

If you are reducing a logical volume, to prevent data loss you must ensure that the file system on that logical volume is not using the space in the logical volume that is being reduced.

Prerequisites

- You have prepared the control node and the managed nodes .

- You are logged in to the control node as a user who can run playbooks on the managed nodes.

- The account you use to connect to the managed nodes has **sudo** permissions for these nodes.

Procedure

1. Create a playbook file, for example, ~/**playbook.yml**, with the following content:

```
---
```

```
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create logical volume
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_safe_mode: false

        storage_pools:
          - name: myvg
            disks:
              - sda
              - sdb
              - sdc
            volumes:
              - name: mylv
                size: 2G
                fs_type: ext4
                mount_point: /mnt/data
```

The settings specified in the example playbook include the following:

size: *<size>*

> You must specify the size by using units (for example, GiB) or percentage (for example, 60%).

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

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

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

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

Verification

- Verify that specified volume has been created or resized to the requested size:

```
# ansible managed-node-01.example.com -m command -a 'lvs myvg'
```

5.1.3. Creating a striped logical volume

With striped logical volume (LV), you can distribute the data across multiple physical volumes (PVs), potentially increasing the read and write speed by utilizing the bandwidth of multiple disks simultaneously.

When creating a striped LV, it is important to consider the stripe number and size. The stripe number is

the count of PVs across which data is distributed. Increasing the stripe number can enhance performance by utilizing multiple disks concurrently. Stripe size is the size of the data chunk written to each disk in the stripe set before moving to the next disk and is specified in kilobytes (KB). The optimal stripe size depends on your workload and the filesystem block size. The default is 64KB and can be adjusted.

Prerequisites

- Administrative access.

Procedure

1. List the names of volume groups and their size:

   > # **vgs -o vg_name,vg_size**
   >
   > VG   VSize
   > VolumeGroupName 30.75g

2. Create a striped LV:

   > # lvcreate --stripes *NumberOfStripes* --stripesize *StripeSize* --size *LogicalVolumeSize* --name
   > *LogicalVolumeName VolumeGroupName*

   Replace *NumberOfStripes* with the number of stripes. Replace *StripeSize* with the stripe size in kilobytes. The **--stripesize** is not a required option. If you do not specify the stripe size it defaults to 64KB. Replace *LogicalVolumeName* with the name of the LV. Replace *VolumeGroupName* with the name of the volume group.

Verification

- Verify that the striped LV is created:

  > # lvs -o lv_name,seg_type
  >
  > LV   Type
  > LogicalVolumeName   striped

5.1.4. Creating a RAID logical volume

RAID logical volumes enable you to use multiple disks for redundancy and performance. LVM supports various RAID levels, including RAID0, RAID1, RAID4, RAID5, RAID6, and RAID10.

With LVM you can create striped RAIDs (RAID0, RAID4, RAID5, RAID6), mirrored RAID (RAID1), or a combination of both (RAID10).

RAID 4, RAID 5, and RAID 6 offer fault tolerance by storing parity data that can be used to reconstruct lost information in case of a disk failure.

When creating RAID LVs, place each stripe on a separate PV. The number of stripes equals to the number of PVs that should be in the volume group (VG).

Table 5.1. Minimal RAID configuration requirements

RAID level	Type	Parity	Minimum number of devices	Minimum stripe number
RAID0	Striping	None	2	2
RAID1	Mirroring	None	2	–
RAID4	Striping	Uses first device to store parity	3	2
RAID5	Striping	Uses an extra device to store parity	3	2
RAID6	Striping	Uses two extra devices to store parity	5	3
RAID10	Striping and mirroring	None	4	2

Prerequisites

- Administrative access.

Procedure

1. List the names of volume groups and their size:

   > # **vgs -o vg_name,vg_size**
   >
   > VG   VSize
   > VolumeGroupName 30.75g

2. Create a RAID LV:

   - To create a striped raid, use:

     > # lvcreate --type raid*level* --stripes *NumberOfStripes* --stripesize *StripeSize* --size *Size* --name *LogicalVolumeName VolumeGroupName*

     Replace *level* with the RAID level 0, 4, 5, or 6. Replace *NumberOfStripes* with the number of stripes. Replace *StripeSize* with the stripe size in kilobytes. Replace *Size* with the size of the LV. Replace *LogicalVolumeName* with the name of the LV.

   - To create a mirrored RAID, use:

     > # lvcreate --type raid1 --mirrors *MirrorsNumber* --size *Size* --name *LogicalVolumeName VolumeGroupName*

Replace *MirrorsNumber* with the number of mirrors. Replace *Size* with the size of the LV. Replace *LogicalVolumeName* with the name of the LV.

- To create a mirrored and striped RAID, use:

  ```
  # lvcreate --type raid10 --mirrors MirrorsNumber --stripes NumberOfStripes --stripesize StripeSize --size Size --name LogicalVolumeName VolumeGroupName
  ```

  Replace *MirrorsNumber* with the number of mirrors. Replace *NumberOfStripes* with the number of stripes. Replace *StripeSize* with the stripe size in kilobytes. Replace *Size* with the size of the LV. Replace *LogicalVolumeName* with the name of the LV.

Verification

- Verify that the RAID LV is created:

  ```
  # lvs -o lv_name,seg_type

  LV              Type
  LogicalVolumeName    raid0
  ```

5.1.5. Creating a thin logical volume

Under thin provisioning, physical extents (PEs) from a volume group (VG) are allocated to create a thin pool with a specific physical size. Logical volumes (LVs) are then allocated from this thin pool based on a virtual size, not limited by the pool's physical capacity. With this, the virtual size of each thin LV can exceed the actual size of the thin pool leading to over–provisioning, when the collective virtual sizes of all thin LVs surpasses the physical capacity of the thin pool. Therefore, it is essential to monitor both logical and physical usage closely to avoid running out of space and outages.

Thin provisioning optimizes storage efficiency by allocating space as needed, lowering initial costs and improving resource utilization. However, when using thin LVs, beware of the following drawbacks:

- Improper discard handling can block the release of unused storage space, causing full allocation of the space over time.

- Copy on Write (CoW) operation can be slower on file systems with snapshots.

- Data blocks can be intermixed between multiple file systems leading to random access limitations.

Prerequisites

- Administrative access.

- You have created a physical volume. For more information, see Creating an LVM physical volume.

- You have created a volume group. For more information, see Creating an LVM volume group .

- You have created a logical volume. For more information, see Creating logical volumes.

Procedure

1. List the names of volume groups and their size:

> # **vgs -o vg_name,vg_size**
>
> VG   VSize
> VolumeGroupName 30.75g

2. Create a thin pool:

   > # lvcreate --type thin-pool --size *PoolSize* --name *ThinPoolName VolumeGroupName*

   Replace *PoolSize* with the maximum amount of disk space the thin pool can use. Replace *ThinPoolName* with the name for the thin pool. Replace *VolumeGroupName* with the name of the volume group.

3. Create a thin LV:

   > # lvcreate --type thin --virtualsize *MaxVolumeSize* --name *ThinVolumeName* --thinpool *ThinPoolName VolumeGroupName*

   Replace *MaxVolumeSize* with the maximum size the volume can grow to within the thin pool. Replace *ThinPoolName* with the name for the thin pool. Replace *VolumeGroupName* with the name of the volume group.

   NOTE

   You can create other thin LVs within the same thin pool.

Verification

- Verify that the thin LV is created:

  > # lvs -o lv_name,seg_type
  > LV   Type
  > ThinPoolName   thin-pool
  > ThinVolumeName   thin

5.1.6. Creating a VDO logical volume

VDO logical volumes (LVs) use the Virtual Data Optimizer (VDO) technology to enhance storage efficiency. VDO LVs have both a virtual size and a physical size. The virtual size refers to the total amount of storage presented to users and applications. The physical size is the actual amount of physical storage allocated from the VG and consumed by the VDO pool.

The virtual size of a VDO LV is generally larger than the physical size of the VDO pool, making it over-provisioned. Due to over-provisioning the physical space in the VDO pool needs to be closely monitored and extended when needed.

A VDO LV and a VDO pool are created as a pair, and always exist as a pair.

Prerequisites

- Administrative access.

Procedure

1. List the names of volume groups and their size:

   # vgs -o vg_name,vg_size

   ```
   VG            VSize
   VolumeGroupName 30.75g
   ```

2. Create a VDO LV:

   ```
   # lvcreate --type vdo --virtualsize VolumeSize --size PhysicalPoolSize --name
   VDOVolumeName --vdopool VDOPoolName VolumeGroupName
   ```

   Replace the *VolumeSize* with the size for the volume. Replace the *PhysicalPoolSize* with the size for the pool. Replace the *VDOVolumeName* with the name for your VDO volume. Replace the *VDOPoolName* with the name for your VDO pool. Replace *VolumeGroupName* with the name of the volume group.

Verification

- Verify that the VDO LV is created:

  ```
  # lvs -o name,seg_type,size


  LV           Type     LSize
  VDOPoolName   vdo-pool  5.00g
  VDOVolumeName vdo       5.00g
  ```

5.2. RESIZING LOGICAL VOLUMES

With Logical Volume Manager (LVM), you can resize logical volumes (LVs) as needed without affecting the data stored on them.

5.2.1. Extending a linear logical volume

You can extend linear (thick) LVs and their snapshots with the **lvextend** command.

Prerequisites

- Administrative access.

- The LV that you want to extend has a filesystem on top.

Procedure

1. Ensure your volume group has enough space to extend your LV:

   ```
   # lvs -o lv_name,lv_size,vg_name,vg_size,vg_free
   LV              LSize  VG          VSize  VFree
   LogicalVolumeName    1.49g   VolumeGroupName 30.75g 29.11g
   ```

2. Extend the linear LV and resize the file system:

   ```
   # lvextend --size +AdditionalSize --resizefs VolumeGroupName/LogicalVolumeName
   ```

Replace *AdditionalSize* with how much space to add to the LV. The default unit of measurement is megabytes, but you can specify other units. Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName* with the name of the thin volume.

Verification

- Verify that the linear LV is extended:

```
# lvs -o lv_name,lv_size
 LV              LSize
 LogicalVolumeName    6.49g
```

5.2.2. Extending a thin logical volume

You can extend the thin logical volume (LV) with the **lvextend** command.

Prerequisites

- Administrative access.

Procedure

1. Ensure that the thin pool has enough space for any new data that will be written:

```
# lvs -o lv_name,lv_size,data_percent

 LV            LSize   Data%
 MyThinPool      20.10g 3.21
 ThinVolumeName    1.10g  4.88
```

2. Extend the thin LV and resize the file system:

```
# lvextend --size +AdditionalSize --resizefs VolumeGroupName/ThinVolumeName
```

Replace *AdditionalSize* with how much space to add to the LV. The default unit of measurement is megabytes, but you can specify other units. Replace *VolumeGroupName* with the name of the volume group. Replace *ThinVolumeName* with the name of the thin volume.

Verification

- Verify the thin LV is extended:

```
# lvs -o lv_name,lv_size,data_percent

 LV            LSize   Data%
 MyThinPool      20.10g 3.21
 ThinVolumeName    6.10g  0.43
```

5.2.3. Extending a thin pool

The virtual size of thin logical volumes can exceed the physical capacity of the thin pool resulting in over-provisioning. To prevent running out of space, you must monitor and periodically extend the capacity of the thin pool.

The **data_percent** metric indicates the percentage of the allocated data space that the thin pool currently uses. The **metadata_percent** metric reflects the percentage of space used for storing metadata, which is essential for managing the mappings within the thin pool.

Monitoring these metrics is vital to ensure efficient storage management and to avoid capacity issues.

LVM provides the option to manually extend the data or metadata capacity as needed. Alternatively, you can enable monitoring and automate the expansion of your thin pool.

5.2.3.1. Manually extending a thin pool

Logical Volume Manager (LVM) provides the option to manually extend the data segment, the metadata segment, or the thin pool.

5.2.3.1.1. Extending a thin pool

You can use the **lvextend** command to extend the thin pool.

Prerequisites

- Administrative access.

Procedure

1. Display the data and metadata space used:

   ```
   # lvs -o lv_name,seg_type,data_percent,metadata_percent

    LV             Type      Data%  Meta%
    ThinPoolName      thin-pool 97.66  26.86
    ThinVolumeName    thin      48.80
   ```

2. Extend the thin pool:

   ```
   # lvextend -L Size VolumeGroupName/ThinPoolName
   ```

   Replace *Size* with the new size for your thin pool. Replace *VolumeGroupName* with the name of the volume group. Replace *ThinPoolName* with the name of the thin pool.

   The data size will be extended. The metadata size will be extended if necessary.

Verification

- Verify that the thin pool is extended:

   ```
   # lvs -o lv_name,seg_type,data_percent,metadata_percent

    LV             Type      Data%  Meta%
    ThinPoolName      thin-pool 24.41  16.93
    ThinVolumeName    thin      24.41
   ```

5.2.3.1.2. Extending a thin pool data segment

You can use the **lvextend** command to extend the **data_percent** segment.

Prerequisites

- Administrative access.

Procedure

1. Display the **data_percent** segment:

   ```
   # lvs -o lv_name,seg_type,data_percent

   LV              Type      Data%
   ThinPoolName    thin-pool 93.87
   ```

2. Extend the **data_percent** segment:

   ```
   # lvextend -L Size VolumeGroupName/ThinPoolName_tdata
   ```

   Replace *Size* with the size for your data segment. Replace *VolumeGroupName* with name of the volume group. Replace *ThinPoolName* with the name of the thin pool.

Verification

- Verify that the **data_percent** segment is extended:

   ```
   # lvs -o lv_name,seg_type,data_percent

   LV              Type      Data%
   ThinPoolName    thin-pool 40.23
   ```

5.2.3.1.3. Extending a thin pool metadata segment

You can use the **lvextend** command to extend the **metadata_percent** segment.

Prerequisites

- Administrative access.

Procedure

1. Display the **metadata_percent** segment:

   ```
   # lvs -o lv_name,seg_type,metadata_percent

   LV              Type      Meta%
   ThinPoolName    thin-pool 75.00
   ```

2. Extend the **metadata_percent** segment:

   ```
   # lvextend -L Size VolumeGroupName/ThinPoolName_tmeta
   ```

–

Replace *Size* with the size for your metadata segment. Replace *VolumeGroupName* with name of the volume group. Replace *ThinPoolName* with the name of the thin pool.

Verification

- Verify that the **metadata_percent** segment is extended:

  ```
  # lvs -o lv_name,seg_type,metadata_percent

  LV            Type     Meta%
  ThinPoolName     thin-pool 0.19
  ```

5.2.3.2. Automatically extending a thin pool

You can automate the expansion of your thin pool by enabling monitoring and setting the **thin_pool_autoextend_threshold** and the **thin_pool_autoextend_percent** configuration parameters.

Prerequisites

- Administrative access.

Procedure

1. Check if the thin pool is monitored:

   ```
   # lvs -o lv_name,vg_name,seg_monitor

   LV            VG            Monitor
   ThinPoolName     VolumeGroupName not monitored
   ```

2. Enable thin pool monitoring with the **dmeventd** daemon:

   ```
   # lvchange --monitor y VolumeGroupName/ThinPoolName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *ThinPoolName* with the name of the thin pool.

3. Open the /**etc/lvm/lvm.conf** file in an editor of your choice.

4. Uncomment the **thin_pool_autoextend_threshold** and **thin_pool_autoextend_percent** lines and set each parameter to a required value:

   ```
   thin_pool_autoextend_threshold = 70
   thin_pool_autoextend_percent = 20
   ```

   thin_pool_autoextend_threshold determines the percentage at which LVM starts to auto-extend the thin pool. For example, setting it to 70 means LVM will try to extend the thin pool when it reaches 70% capacity.

   thin_pool_autoextend_percent specifies by what percentage the thin pool should be extended when it reaches threshold. For example, setting it to 20 means the thin pool will be increased by 20% of its current size.

5. Save the changes and exit the editor.

6. Restart the **lvm2-monitor**:

```
# systemctl restart lvm2-monitor
```

5.2.4. Extending a VDO Pool

It is crucial to monitor and periodically extend the capacity of the VDO pool to prevent running out of space.

Logical Volume Manager (LVM) provides the option to manually extend the VDO pool capacity as needed. Alternatively, you can enable monitoring and automate the extension of your VDO pool.

5.2.4.1. Manually extending a VDO Pool

Use the **lvextend** command to extend a VDO pool.

Prerequisites

- Administrative access.

Procedure

1. Display the current VDO usage:

   ```
   # lvs -o lv_name,vg_name,lv_size,data_percent VolumeGroupName/VDOPoolName

   LV          VG              LSize Data%
   VDOPoolName VolumeGroupName 5.00g 60.03
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace the *VDOPoolName* with the name of the VDO pool.

2. Extend the VDO Pool:

   ```
   # lvextend --size PhysicalSize VolumeGroupName/VDOPoolName
   ```

   Replace *PhysicalSize* with the new physical size. Replace *VolumeGroupName* with the name of the volume group. Replace the *VDOPoolName* with the name of the VDO pool.

Verification

1. Verify that the VDO pool is extended:

   ```
   # lvs -o lv_name,vg_name,lv_size,data_percent VolumeGroupName/VDOPoolName

   LV          VG              LSize  Data%
   VDOPoolName VolumeGroupName 10.00g 30.02
   ```

5.2.4.2. Automatically extending a VDO Pool

You can automate the expansion of your Virtual Data Optimizer (VDO) pool by enabling monitoring and setting the **vdo_pool_autoextend_threshold** and the **vdo_pool_autoextend_percent** parameters.

Prerequisites

- Administrative access.

Procedure

1. Check if the VDO pool is monitored:

   ```
   # lvs -o name,seg_monitor VolumeGroupName/VDOPoolName

   LV            VG            Monitor
   VDOPoolName        VolumeGroupName not monitored
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *VDOPoolName* with the name of the VDO pool.

2. Enable VDO pool monitoring with the **dmeventd** daemon:

   ```
   # lvchange --monitor y VolumeGroupName/VDOPoolName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *VDOPoolName* with the name of the VDO pool.

3. Open the **/etc/lvm/lvm.conf** file in an editor of your choice.

4. Uncomment the **vdo_pool_autoextend_percent** and **vdo_pool_autoextend_threshold** lines and set each parameter to a required value:

   ```
   vdo_pool_autoextend_threshold = 70
   vdo_pool_autoextend_percent = 20
   ```

   vdo_pool_autoextend_threshold determines the percentage at which LVM starts to auto-extend the VDO pool. For example, setting it to 70 means LVM tries to extend the VDO pool when it reaches 70% capacity.

   vdo_pool_autoextend_percent specifies by what percentage the VDO pool should be extended when it reaches the threshold. For example, setting it to 20 means the VDO pool will be increased by 20% of its current size.

5. Save the changes and exit the editor.

6. Restart the **lvm2-monitor**:

   ```
   # systemctl restart lvm2-monitor
   ```

5.2.5. Shrinking logical volumes

When the size of the LV is reduced, the freed up logical extents are returned to the volume group and then can be used by other LVs.

> **WARNING**
>
> Data stored in the reduced area is lost. Always back up the data and resize the file system before proceeding.

Prerequisites

- Administrative access.

Procedure

1. List the logical volumes and their volume groups:

   ```
   # lvs -o lv_name,vg_name,lv_size

   LV              VG              LSize
   LogicalVolumeName    VolumeGroupName 6.49g
   ```

2. Check where the logical volume is mounted:

   ```
   # findmnt -o SOURCE,TARGET /dev/VolumeGroupName/LogicalVolumeName

   SOURCE                          TARGET
   /dev/mapper/VolumeGroupName-LogicalVolumeName /MountPoint
   ```

   Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume.

3. Unmount the logical volume:

   ```
   # umount /MountPoint
   ```

   Replace */MountPoint* with the mounting point for your logical volume.

4. Check and repair any file system errors:

   ```
   # e2fsck -f /dev/VolumeGroupName/LogicalVolumeName
   ```

5. Resize the LV and the file system:

   ```
   # lvreduce --size TargetSize --resizefs VolumeGroupName/LogicalVolumeName
   ```

   Replace *TargetSize* with the new size of the LV. Replace *VolumeGroupName/LogicalVolumeName* with the path to your logical volume.

6. Remount the file system:

   ```
   # mount /dev/VolumeGroupName/LogicalVolumeName /MountPoint
   ```

   Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume. Replace */MountPoint* with the mounting point for your file system.

Verification

1. Verify the space usage of the file system:

   ```
   # df -hT /MountPoint/

   Filesystem                        Type  Size  Used Avail Use% Mounted on
   /dev/mapper/VolumeGroupName-NewLogicalVolumeName ext4  2.9G  139K  2.7G   1%
   /MountPoint
   ```

   Replace */MountPoint* with the mounting point for your logical volume.

2. Verify the size of the LV:

   ```
   # lvs -o lv_name,lv_size

   LV              LSize
   NewLogicalVolumeName 4.00g
   ```

5.3. RENAMING LOGICAL VOLUMES

You can rename an existing logical volume, including snapshots, by using the **lvrename** command.

Prerequisites

- Administrative access.

Procedure

1. List the logical volumes and their volume groups:

   ```
   # lvs -o lv_name,vg_name

   LV              VG
   LogicalVolumeName VolumeGroupName
   ```

2. Rename the logical volume:

   ```
   # lvrename VolumeGroupName/LogicalVolumeName
   VolumeGroupName/NewLogicalVolumeName
   ```

   Replace *VolumeGroupName* with name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume. Replace *NewLogicalVolumeName* with the new logical volume name.

Verification

- Verify that the logical volume is renamed:

  ```
  # lvs -o lv_name
   LV
   NewLogicalVolumeName
  ```

5.4. REMOVING LOGICAL VOLUMES

You can remove an existing logical volume, including snapshots, by using the **lvremove** command.

Prerequisites

- Administrative access.

Procedure

1. List the logical volumes and their paths:

   > # **lvs -o lv_name,lv_path**
   >
   > LV   Path
   > LogicalVolumeName /dev/VolumeGroupName/LogicalVolumeName

2. Check where the logical volume is mounted:

   > # findmnt -o SOURCE,TARGET */dev/VolumeGroupName/LogicalVolumeName*
   >
   > SOURCE   TARGET
   > /dev/mapper/VolumeGroupName-LogicalVolumeName /MountPoint

   Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume.

3. Unmount the logical volume:

   > # **umount */MountPoint***

   Replace */MountPoint* with the mounting point for your logical volume.

4. Remove the logical volume:

   > # **lvremove *VolumeGroupName/LogicalVolumeName***

   Replace *VolumeGroupName/LogicalVolumeName* with the path to your logical volume.

5.5. ACTIVATING LOGICAL VOLUMES

You can activate the logical volume with the **lvchange** command.

Prerequisites

- Administrative access.

Procedure

1. List the logical volumes, their volume groups, and their paths:

   > # **lvs -o lv_name,vg_name,lv_path**
   >
   > LV   VG   Path

> LogicalVolumeName VolumeGroupName VolumeGroupName/LogicalVolumeName

2. Activate the logical volume:

   > # **lvchange --activate y** *VolumeGroupName*/*LogicalVolumeName*

   Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

   > **NOTE**
   >
   > When activating a thin LV that was created as a snapshot of another LV, you might need to use the **--ignoreactivationskip** option to activate it.

Verification

- Verify that the LV is active:

  ```
  # lvdisplay VolumeGroupName/LogicalVolumeName

  ...
  LV Status              available
  ```

  Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

5.6. DEACTIVATING LOGICAL VOLUMES

By default, when you create a logical volume, it is in an active state. You can deactivate the logical volume with the **lvchange** command.

> **WARNING**
>
> Deactivating a logical volume with active mounts or in use can lead to data inconsistencies and system errors.

Prerequisites

- Administrative access.

Procedure

1. List the logical volumes, their volume groups, and their paths:

   > # **lvs -o lv_name,vg_name,lv_path**
   >
   > LV   VG   Path
   > LogicalVolumeName VolumeGroupName /dev/VolumeGroupName/LogicalVolumeName

2. Check where the logical volume is mounted:

```
# findmnt -o SOURCE,TARGET /dev/VolumeGroupName/LogicalVolumeName

SOURCE                          TARGET
/dev/mapper/VolumeGroupName-LogicalVolumeName /MountPoint
```

Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume.

3. Unmount the logical volume:

```
# umount /MountPoint
```

Replace */MountPoint* with the mounting point for your logical volume.

4. Deactivate the logical volume:

```
# lvchange --activate n VolumeGroupName/LogicalVolumeName
```

Replace *VolumeGroupName* with name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

Verification

- Verify that the LV is not active:

```
# lvdisplay VolumeGroupName/LogicalVolumeName

...
  LV Status              NOT available
```

Replace *VolumeGroupName* with name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

CHAPTER 6. ADVANCED LOGICAL VOLUME MANAGEMENT

LVM includes advanced features such as, Snapshots, which are point-in-time copies of logical volumes (LVs), Caching, with which you can use faster storage as a cache for slower storage, Creating custom thin pools, and Creating custom VDO LVs.

6.1. MANAGING LOGICAL VOLUME SNAPSHOTS

A snapshot is a logical volume (LV) that mirrors the content of another LV at a specific point in time.

6.1.1. Understanding logical volume snapshots

When you create a snapshot, you are creating a new LV that serves as a point-in-time copy of another LV. Initially, the snapshot LV contains no actual data. Instead, it references the data blocks of the original LV at the moment of snapshot creation.

> **WARNING**
>
> It is important to regularly monitor the snapshot's storage usage. If a snapshot reaches 100% of its allocated space, it will become invalid.
>
> It is essential to extend the snapshot before it gets completely filled. This can be done manually by using the **lvextend** command or automatically via the **/etc/lvm/lvm.conf** file.

Thick LV snapshots

When data on the original LV changes, the copy-on-write (CoW) system copies the original, unchanged data to the snapshot before the change is made. This way, the snapshot grows in size only as changes occur, storing the state of the original volume at the time of the snapshot's creation. Thick snapshots are a type of LV that requires you to allocate some amount of storage space upfront. This amount can later be extended or reduced, however, you should consider what type of changes you intend to make to the original LV. This helps you to avoid either wasting resources by allocating too much space or needing to frequently increase the snapshot size if you allocate too little.

Thin LV snapshots

Thin snapshots are a type of LV created from an existing thin-provisioned LV. Thin snapshots do not require allocating extra space upfront. Initially, both the original LV and its snapshot share the same data blocks. When changes are made to the original LV, it writes new data to different blocks, while the snapshot continues to reference the original blocks, preserving a point-in-time view of the LV's data at the snapshot creation.

Thin provisioning is a method of optimizing and managing storage efficiently by allocating disk space on an as-needed basis. This means that you can create multiple LVs without needing to allocate a large amount of storage upfront for each LV. The storage is shared among all LVs in a thin pool, making it a more efficient use of resources. A thin pool allocates space on-demand to its LVs.

Choosing between thick and thin LV snapshots

The choice between thick or thin LV snapshots is directly determined by the type of LV you are taking a snapshot of. If your original LV is a thick LV, your snapshots will be thick. If your original LV is thin, your snapshots will be thin.

6.1.2. Managing thick logical volume snapshots

When you create a thick LV snapshot, it is important to consider the storage requirements and the intended lifespan of your snapshot. You need to allocate enough storage for it based on the expected changes to the original volume.

The snapshot must have a sufficient size to capture changes during its intended lifespan, but it cannot exceed the size of the original LV. If you expect a low rate of change, a smaller snapshot size of 10%–15% might be sufficient. For LVs with a high rate of change, you might need to allocate 30% or more.

> **IMPORTANT**
>
> It is essential to extend the snapshot before it gets completely filled. If a snapshot reaches 100% of its allocated space, it becomes invalid. You can monitor the snapshot capacity with the **lvs -o lv_name,data_percent,origin** command.

6.1.2.1. Creating thick logical volume snapshots

You can create a thick LV snapshot with the **lvcreate** command.

Prerequisites

- Administrative access.

- You have created a physical volume. For more information, see Creating an LVM physical volume.

- You have created a volume group. For more information, see Creating an LVM volume group .

- You have created a logical volume. For more information, see Creating logical volumes.

Procedure

1. Identify the LV of which you want to create a snapshot:

   ```
   # lvs -o vg_name,lv_name,lv_size

   VG              LV              LSize
   VolumeGroupName LogicalVolumeName 10.00g
   ```

   The size of the snapshot cannot exceed the size of the LV.

2. Create a thick LV snapshot:

   ```
   # lvcreate --snapshot --size SnapshotSize --name SnapshotName
   VolumeGroupName/LogicalVolumeName
   ```

   Replace *SnapshotSize* with the size you want to allocate for the snapshot (e.g. 10G). Replace *SnapshotName* with the name you want to give to the snapshot logical volume. Replace *VolumeGroupName* with the name of the volume group that contains the original logical volume. Replace *LogicalVolumeName* with the name of the logical volume that you want to create a snapshot of.

Verification

- Verify that the snapshot is created:

```
# lvs -o lv_name,origin

LV              Origin
LogicalVolumeName
SnapshotName        LogicalVolumeName
```

6.1.2.2. Manually extending logical volume snapshots

If a snapshot reaches 100% of its allocated space, it becomes invalid. It is essential to extend the snapshot before it gets completely filled. This can be done manually by using the **lvextend** command.

Prerequisites

- Administrative access.

Procedure

1. List the names of volume groups, logical volumes, source volumes for snapshots, their usage percentages, and sizes:

   ```
   # lvs -o vg_name,lv_name,origin,data_percent,lv_size
   VG            LV            Origin          Data%  LSize
   VolumeGroupName LogicalVolumeName                    10.00g
   VolumeGroupName SnapshotName     LogicalVolumeName 82.00   5.00g
   ```

2. Extend the thick-provisioned snapshot:

   ```
   # lvextend --size +AdditionalSize VolumeGroupName/SnapshotName
   ```

   Replace *AdditionalSize* with how much space to add to the snapshot (for example, +1G). Replace *VolumeGroupName* with the name of the volume group. Replace *SnapshotName* with the name of the snapshot.

Verification

- Verify that the LV is extended:

   ```
   # lvs -o vg_name,lv_name,origin,data_percent,lv_size
   VG            LV            Origin          Data%  LSize
   VolumeGroupName LogicalVolumeName                    10.00g
   VolumeGroupName SnapshotName     LogicalVolumeName 68.33   6.00g
   ```

6.1.2.3. Automatically extending thick logical volume snapshots

If a snapshot reaches 100% of its allocated space, it becomes invalid. It is essential to extend the snapshot before it gets completely filled. This can be done automatically.

Prerequisites

- Administrative access.

Procedure

1. Open the **/etc/lvm/lvm.conf** file in an editor of your choice.

2. Uncomment the **snapshot_autoextend_threshold** and **snapshot_autoextend_percent** lines and set each parameter to a required value:

   ```
   snapshot_autoextend_threshold = 70
   snapshot_autoextend_percent = 20
   ```

   snapshot_autoextend_threshold determines the percentage at which LVM starts to auto-extend the snapshot. For example, setting the parameter to 70 means that LVM will try to extend the snapshot when it reaches 70% capacity.

   snapshot_autoextend_percent specifies by what percentage the snapshot should be extended when it reaches the threshold. For example, setting the parameter to 20 means the snapshot will be increased by 20% of its current size.

3. Save the changes and exit the editor.

4. Restart the **lvm2-monitor**:

   ```
   # systemctl restart lvm2-monitor
   ```

6.1.2.4. Merging thick logical volume snapshots

You can merge a thick LV snapshot into the original logical volume from which the snapshot was created. The process of merging means that the original LV is reverted to the state it was in when the snapshot was created. Once the merge is complete, the snapshot is removed.

> **NOTE**
>
> The merge between the original and snapshot LV is postponed if either is active. It only proceeds once the LVs are reactivated and not in use.

Prerequisites

- Administrative access.

Procedure

1. List the LVs, their volume groups, and their paths:

   ```
   # lvs -o lv_name,vg_name,lv_path

     LV                 VG               Path
     LogicalVolumeName    VolumeGroupName /dev/VolumeGroupName/LogicalVolumeName
     SnapshotName         VolumeGroupName /dev/VolumeGroupName/SnapshotName
   ```

2. Check where the LVs are mounted:

   ```
   # findmnt -o SOURCE,TARGET /dev/VolumeGroupName/LogicalVolumeName
   # findmnt -o SOURCE,TARGET /dev/VolumeGroupName/SnapshotName
   ```

Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume.
Replace */dev/VolumeGroupName/SnapshotName* with the path to your snapshot.

3. Unmount the LVs:

   ```
   # umount /LogicalVolume/MountPoint
   # umount /Snapshot/MountPoint
   ```

   Replace */LogicalVolume/MountPoint* with the mounting point for your logical volume. Replace
   /Snapshot/MountPoint with the mounting point for your snapshot.

4. Deactivate the LVs:

   ```
   # lvchange --activate n VolumeGroupName/LogicalVolumeName
   # lvchange --activate n VolumeGroupName/SnapshotName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName*
   with the name of the logical volume. Replace *SnapshotName* with the name of your snapshot.

5. Merge the thick LV snapshot into the origin:

   ```
   # lvconvert --merge VolumeGroupName/SnapshotName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *SnapshotName* with
   the name of the snapshot.

6. Activate the LV:

   ```
   # lvchange --activate y VolumeGroupName/LogicalVolumeName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName*
   with the name of the logical volume.

7. Mount the LV:

   ```
   # mount /dev/VolumeGroupName/LogicalVolumeName /LogicalVolume/MountPoint
   ```

   Replace */dev/VolumeGroupName/LogicalVolumeName* with the path to your logical volume.
   Replace */LogicalVolume/MountPoint* with the mounting point for your logical volume.

Verification

- Verify that the snapshot is removed:

  ```
  # lvs -o lv_name
  ```

6.1.3. Managing thin logical volume snapshots

Thin provisioning is appropriate where storage efficiency is a priority. Storage space dynamic allocation
reduces initial storage costs and maximizes the use of available storage resources.

In environments with dynamic workloads or where storage grows over time, thin provisioning allows for
flexibility. It enables the storage system to adapt to changing needs without requiring large upfront
allocations of the storage space. With dynamic allocation, over-provisioning is possible, where the total

size of all LVs can exceed the physical size of the thin pool, under the assumption that not all space will be utilized at the same time.

6.1.3.1. Creating thin logical volume snapshots

You can create a thin LV snapshot with the **lvcreate** command. When creating a thin LV snapshot, avoid specifying the snapshot size. Including a size parameter results in the creation of a thick snapshot instead.

Prerequisites

- Administrative access.

- You have created a physical volume. For more information, see [Creating an LVM physical volume](#).

- You have created a volume group. For more information, see [Creating an LVM volume group](#) .

- You have created a logical volume. For more information, see [Creating logical volumes](#).

Procedure

1. Identify the LV of which you want to create a snapshot:

   ```
   # lvs -o lv_name,vg_name,pool_lv,lv_size

   LV              VG            Pool      LSize
   PoolName        VolumeGroupName           152.00m
   ThinVolumeName    VolumeGroupName PoolName   100.00m
   ```

2. Create a thin LV snapshot:

   ```
   # lvcreate --snapshot --name ThinSnapshotName VolumeGroupName/ThinVolumeName
   ```

   Replace *ThinSnapshotName* with the name you want to give to the snapshot logical volume. Replace *VolumeGroupName* with the name of the volume group that contains the original logical volume. Replace *ThinVolumeName* with the name of the thin logical volume that you want to create a snapshot of.

Verification

- Verify that the snapshot is created:

  ```
  # lvs -o lv_name,origin

  LV              Origin
  PoolName
  ThinSnapshotName  ThinVolumeName
  ThinVolumeName
  ```

6.1.3.2. Merging thin logical volume snapshots

You can merge thin LV snapshot into the original logical volume from which the snapshot was created. The process of merging means that the original LV is reverted to the state it was in when the snapshot was created. Once the merge is complete, the snapshot is removed.

Prerequisites

- Administrative access.

Procedure

1. List the LVs, their volume groups, and their paths:

   ```
   # lvs -o lv_name,vg_name,lv_path

   LV              VG          Path
   ThinPoolName      VolumeGroupName
   ThinSnapshotName  VolumeGroupName /dev/VolumeGroupName/ThinSnapshotName
   ThinVolumeName    VolumeGroupName /dev/VolumeGroupName/ThinVolumeName
   ```

2. Check where the original LV is mounted:

   ```
   # findmnt -o SOURCE,TARGET /dev/VolumeGroupName/ThinVolumeName
   ```

   Replace *VolumeGroupName/ThinVolumeName* with the path to your logical volume.

3. Unmount the LV:

   ```
   # umount /ThinVolumeName/MountPoint
   ```

   Replace */ThinVolumeName/MountPoint* with the mounting point for your logical volume. Replace */ThinSnapshot/MountPoint* with the mounting point for your snapshot.

4. Deactivate the LV:

   ```
   # lvchange --activate n VolumeGroupName/ThinVolumeName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *ThinVolumeName* with the name of the logical volume.

5. Merge the thin LV snapshot into the origin:

   ```
   # lvconvert --mergethin VolumeGroupName/ThinSnapshotName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *ThinSnapshotName* with the name of the snapshot.

6. Mount the LV:

   ```
   # mount /ThinVolumeName/MountPoint
   ```

   Replace */ThinVolumeName/MountPoint* with the mounting point for your logical volume.

Verification

- Verify that the original LV is merged:

  ```
  # lvs -o lv_name
  ```

6.2. CACHING LOGICAL VOLUMES

You can cache logical volumes by using the **dm-cache** or **dm-writecache** targets.

dm-cache utilizes faster storage device (SSD) as cache for a slower storage device (HDD). It caches read and write data, optimizing access times for frequently used data. It is beneficial in mixed workload environments where enhancing read and write operations can lead to significant performance improvements.

dm-writecache optimizes write operations by using a faster storage medium (SSD) to temporarily hold write data before it is committed to the primary storage device (HDD). It is beneficial for write-intensive applications where write performance can slow down the data transfer process.

6.2.1. Caching logical volumes with dm-cache

When caching LV with **dm-cache**, a cache pool is created. A cache pool is a LV that combines both the cache data, which stores the actual cached content, and cache metadata, which tracks what content is stored in the cache. This pool is then associated with a specific LV to cache its data.

dm-cache targets two types of blocks: frequently accessed (hot) blocks are moved to the cache, while less frequently accessed (cold) blocks remain on the slower device.

Prerequisites

- Administrative access.

Procedure

1. Display the LV you want to cache and its volume group:

   ```
   # lvs -o lv_name,vg_name
    LV                VG
    LogicalVolumeName    VolumeGroupName
   ```

2. Create the cache pool:

   ```
   # lvcreate --type cache-pool --name CachePoolName --size Size VolumeGroupName
   /FastDevicePath
   ```

   Replace *CachePoolName* with the name of the cache pool. Replace *Size* with the size for your cache pool. Replace *VolumeGroupName* with the name of the volume group. Replace */FastDevicePath* with the path to your fast device, for example SSD or NVME.

3. Attach the cache pool to the LV:

   ```
   # lvconvert --type cache --cachepool VolumeGroupName/CachePoolName
   VolumeGroupName/LogicalVolumeName
   ```

Verification

- Verify that the LV is now cached:

  ```
  # lvs -o lv_name,pool_lv

    LV                Pool
    LogicalVolumeName    [CachePoolName_cpool]
  ```

6.2.2. Caching logical volumes with dm-writecache

When caching LVs with **dm-writecache**, a caching layer between the logical volume and the physical storage device is created. **dm-writecache** operates by temporarily storing write operations in a faster storage medium, such as an SSD, before eventually writing them back to the primary storage device, optimizing write-intensive workloads.

Prerequisites

- Administrative access.

Procedure

1. Display the logical volume you want to cache and its volume group:

   ```
   # lvs -o lv_name,vg_name
     LV              VG
     LogicalVolumeName    VolumeGroupName
   ```

2. Create a cache volume:

   ```
   # lvcreate --name CacheVolumeName --size Size VolumeGroupName /FastDevicePath
   ```

   Replace *CacheVolumeName* with the name of the cache volume. Replace *Size* with the size for your cache pool. Replace *VolumeGroupName* with the name of the volume group. Replace */FastDevicePath* with the path to your fast device, for example SSD or NVME.

3. Deactivate the cache volume:

   ```
   # lvchange -an VolumeGroupName/CacheVolumeName
   ```

   Replace *VolumeGroupName* with the name of the volume group. Replace *CacheVolumeName* with the name of the cache volume.

4. Attach the cache volume to the LV:

   ```
   # lvconvert --type writecache --cachevol CacheVolumeName
   VolumeGroupName/LogicalVolumeName
   ```

   Replace *CacheVolumeName* with the name of the cache volume. Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

Verification

- Verify that the LV is now cached:

```
# lvs -o lv_name,pool_lv

LV              Pool
LogicalVolumeName    [CacheVolumeName_cvol]
```

6.2.3. Uncaching a logical volume

You can remove caching from a logical volume, either by detaching the cache volume while preserving its data, or by completely uncache the logical volume and remove the cache, freeing up storage space.

Use two main ways to remove caching from a LV.

Splitting

You can detach the cache from the LV but preserve the cache volume itself. In this case the LV will no longer benefit from the caching mechanism but the cache volume and its data will remain intact. While the cache volume is preserved, the data within the cache cannot be reused and will be erased the next time it is used in a caching setup.

Uncaching

You can detaches the cache from the LV and remove the cache volume entirely. This action effectively destroys the cache, freeing up the space.

Prerequisites

- Administrative access.

Procedure

1. Display the cached LV:

   ```
   # lvs -o lv_name,pool_lv,vg_name

   LV              Pool              VG
   LogicalVolumeName    [CacheVolumeName_cvol] VolumeGroupName
   ```

2. Detach or remove the cached volume:

   - To detach the cached volume, use:

     ```
     # lvconvert --splitcache VolumeGroupName/LogicalVolumeName
     ```

   - To detach and remove the cached volume, use:

     ```
     # lvconvert --uncache VolumeGroupName/LogicalVolumeName
     ```

     Replace *VolumeGroupName* with the name of the volume group. Replace *LogicalVolumeName* with the name of the logical volume.

Verification

- Verify that the LV is not cached:

  ```
  # lvs -o lv_name,pool_lv
  ```

6.3. CREATING A CUSTOM THIN POOL

You can create custom thin pools to have a better control over the storage.

Prerequisites

- Administrative access.

Procedure

1. Display available volume groups:

   ```
   # vgs -o vg_name

     VG
     VolumeGroupName
   ```

2. List available devices:

   ```
   # lsblk
   ```

3. Create a LV to hold the thin pool data:

   ```
   # lvcreate --name ThinPoolDataName --size Size VolumeGroupName /DevicePath
   ```

   Replace *ThinPoolDataName* with the name for your thin pool data LV. Replace *Size* with the size for your LV. Replace *VolumeGroupName* with the name of your volume group. Replace */DevicePath* with your device path.

4. Create a LV to hold the thin pool metadata:

   ```
   # lvcreate --name ThinPoolMetadataName --size Size VolumeGroupName /DevicePath
   ```

5. Combine the LVs into a thin pool:

   ```
   # lvconvert --type thin-pool --poolmetadata ThinPoolMetadataName
   VolumeGroupName/ThinPoolDataName
   ```

Verification

1. Verify that the custom thin pool is created:

   ```
   # lvs -o lv_name,seg_type

     LV               Type
     ThinPoolDataName  thin-pool
   ```

6.4. CREATING A CUSTOM VDO LOGICAL VOLUME

With Logical Volume Manager (LVM), you can create a custom LV that uses Virtual Data Optimizer (VDO) pool for data storage.

Prerequisites

Prerequisites

- Administrative access.

Procedure

1. Display the VGs:

   ```
   # vgs

   VG              #PV #LV #SN Attr   VSize   VFree
   VolumeGroupName   1   0   0 wz--n-  28.87g 28.87g
   ```

2. Create a LV to be converted to a VDO pool:

   ```
   # lvcreate --name VDOPoolName --size Size VolumeGroupName /DevicePath
   ```

   Replace *VDOPoolName* with the name for your VDO pool. Replace *Size* with the size for your VDO pool. Replace *VolumeGroupName* with the name of the VG. Replace */DevicePath* with your device path.

3. Convert this LV to a VDO pool. In this conversion, you are creating a new VDO LV that uses the VDO pool. Because **lvcreate** is creating a new VDO LV, you must specify parameters for the new VDO LV. Use **--name|-n** to specify the name of the new VDO LV, and **--virtualsize|-V** to specify the size of the new VDO LV.

   ```
   # lvconvert --type vdo-pool --name VDOVolumeName --virtualsize VDOVolumeSize
   VolumeGroupName/VDOPoolName
   ```

   Replace *VDOVolumeName* with the name for your VDO volume. Replace *VDOVolumeSize* with the size for your VDO volume. Replace *VolumeGroupName/VDOPoolName* with the names for your VG and your VDO pool.

Verification

1. Verify that the LV is converted to the VDO pool:

   ```
   # lvs -o lv_name,vg_name,seg_type

   LV                VG            Type
   VDOPoolName           VolumeGroupName vdo-pool
   VDOVolumeName          VolumeGroupName vdo
   ```

CHAPTER 7. MANAGING SYSTEM UPGRADES WITH SNAPSHOTS

Perform revert capable upgrades of Red Hat Enterprise Linux systems to return to an earlier version of the operating system. You can use *Snapshot Manager* (**snapm**), *Boom Boot Manager* (**boom**), and the *Leapp* operating system modernization framework.

Snapshot Manager provides an easy-to-use front end to storage snapshots using LVM2 or Stratis. Snapshot Manager uses a plugin design to automatically find snapshot providers for a list of mount points or block device paths.

Snapshots are created for each given mount point or device (*sources*) and bound together into a *snapshot set*. Snapshot Manager integrates with Boom Boot Manager and automatically manages boot entries for system snapshots when you use the **--boot** or **--revert** options.

Before performing the operating system upgrades, consider the following aspects:

- System upgrades with snapshots support revert-protected upgrades if the following conditions are met:

  - The system (and all file systems to be included in the snapshot process) is installed to LVM2 logical volumes or Stratis storage file systems. LVM2 support includes both linear (thick) volumes and thin-provisioned volumes.

  - Enough free space is available to maintain the snapshots for the duration of the upgrade process.

  - You are running at least Red Hat Enterprise Linux 9.6.

- Revert occurs at the granularity of file system mounts. If the following directories are part of the /**var** or root file systems, reverting an upgrade will also revert the contents of these directories:

  - /**var/log**

  - /**var/lib/libvirt/images**

  - /**var/lib/containers**
    If this is not desired, consider installing the system to independent volumes to hold the content of the log, images, and containers directories as required. For details on partitions on AMD and Intel 64-bit, and 64-bit ARM architectures, see Recommended partitioning scheme.

- Ensuring consistency:

  - Snapshot Manager takes snapshots of the running system.

  - File systems are frozen to ensure *crash consistency*.

  - To ensure *application consistency*, consider shutting down running services or isolating to a specific systemd unit (for example, **rescue.target**).

- System upgrades with snapshots do not work for Red Hat Update Infrastructure (RHUI) systems. Instead of using Snapshot Manager, consider creating snapshots of your virtual machines (VMs).

- System upgrades with Snapshot Manager are currently unsupported for systems deployed with Image Mode.

7.1. OVERVIEW OF THE SNAPSHOT MANAGER PROCESS

Create a *snapshot set* by using the **snapm** command so that you can access, use, and revert to an earlier version of the operating system.

Snapshot Manager can automatically create boot entries that you can select and access from the GRUB boot loader menu. With a *snapshot entry*, you can boot and access the earlier version of the operating system. With a *revert entry* in conjunction with the **snapm snapset revert** command, you can restore the state of the system as it existed before the update attempt.

The following boot entries are part of the upgrade and revert processes:

Upgrade boot entry

Boots the Leapp upgrade environment when using the **leapp** utility to perform a major version upgrade. Use the **leapp** utility to create and manage this boot entry. The **leapp** upgrade process automatically removes this entry.

Red Hat Enterprise Linux 10 boot entry

Boots the upgrade system environment. The **leapp** utility automatically creates this boot entry after a successful upgrade process.

Snapshot boot entry

Boots the snapshot of the original system. Use it to review and test the earlier operating system state, following a successful or unsuccessful upgrade attempt. Before upgrading the operating system, use the **snapm snapset create** command with **-b** (**--boot**) to create this boot entry.

Revert boot entry

Boots the original system environment after initiating a revert with the **snapm snapset revert** command. When booted in this way, reverts any upgrade to the earlier system state. Use the **snapm snapset create** command with **-r** (**--revert**) to create this boot entry when initiating a revert of the upgrade procedure.
A separate revert entry protects against a possible removal or modification of the images during the upgrade process, because Boom Boot Manager automatically backs up any boot images required to start the system (**vmlinuz** and **initramfs**).

7.2. SIZE POLICIES

With Snapshot Manager you can provide a *size policy* when creating snapshot sets. The size policy is a hint that specifies how much space is needed for the snapshot content. Different plugins that use different snapshot backends apply size policies in different ways, according to the requirements of individual snapshot providers.

For the snapshot data to remain valid during the whole upgrade process, ensure that enough space is available for all types of snapshot.

- **lvm2cow**: The size policy determines the size of the exception store allocated for the snapshot volume. This is a fixed size that can be extended after creation and reflects the maximum amount of change that the snapshot can store. If the space available to the snapshot is exhausted, it is invalidated by the kernel and can no longer be used. For more information, see **snapm(8)** and **lvcreate(8)** man pages on your system.

- **lvm2thin** and **stratis**: The size policies are checked against the available space in the corresponding thin pool at the time the snapshot is created. The space is consumed dynamically from the thin pool as the content of the original volume changes. For more information, see **lvmthin(7)** and **stratis(8)** man pages on your system.

In either case, an error occurs if not enough space is available for the specified size policy.

Snapshot Manager provides four types of size policy:

- **FIXED**: A fixed size with an optional unit suffix, such as MiB, GiB, TiB, and others. For example, **10GiB**.

- **%FREE**: A percentage of the free space available from 0 to 100%. For example, **50%FREE**.

- **%USED**: A percentage of the space currently consumed on the mount point, as reported by the **df** command. You can use values greater than 100% to allow the existing content to be completely overwritten without running out of space. This policy can only be applied to snapshot sources that correspond to mounted file systems. For example, **200%USED**.

- **%SIZE**: A percentage of the size of the origin volume from 0 to 100%. For example, **75%SIZE**.

You can specify size policies as a global default by using the **--size-policy** argument, or for individual sources by appending a size policy string separated by a colon (**:**):

- **--size-policy=100%SIZE**

- **/:4GiB /var:75%SIZE**

Determine the appropriate size policy by examining the space that each file system currently uses, and the space available in the volume groups or thin pools that will be used. If enough space is available, the size policy **100%SIZE** ensures that a snapshot origin volume can be completely overwritten without risking the snapshot becoming invalid.

7.3. UPGRADING TO ANOTHER VERSION BY USING SNAPSHOT MANAGER

Perform an upgrade of your Red Hat Enterprise Linux operating system by using Snapshot Manager.

Prerequisites

- You are running Red Hat Enterprise Linux 9.6.

- You have installed the current version of the **snapm** and **boom-boot** packages. Use **dnf -y install snapm** to install Snapshot Manager and its dependencies.

- You have enough space available for the snapshot set. Make a size estimate based on the size of the original installation. List all the mounted logical volumes and ensure that enough space is available for each volume.

- You have installed the **leapp** and **leapp-upgrade-el9toel10** packages.

Procedure

1. Create a named snapshot set of your system volumes, with boot and revert boot entries. For example, if your system consists of separate root and **/var** file systems, and you want to name the snapshot set **before-upgrade**:

   ```
   # snapm snapset create --boot --revert before-upgrade / /var
   SnapsetName:   before-upgrade
   Sources:       /, /var
   NrSnapshots:   2
   Time:          2025-04-23 20:41:53
   UUID:          5e56c455-6285-5f2d-a640-1595eac8381a
   Status:        Active
   Autoactivate:  yes
   Bootable:      yes
   BootEntries:
     SnapshotEntry:  178e20f
     RevertEntry:    6703c58
   ```

   Where:

   - **create** creates the snapshot set.

   - **--boot** enables creation of a snapshot boot entry.

   - **--revert** enables creation of a revert boot entry.

   - **before-upgrade** specifies the name of the snapshot set.

   - **/ /var** is the space–separated list of mount points to include in the snapshot set.

   This command automatically creates a *boom operating system profile* for the running system if one does not already exist. To have more control over the configured operating system profile (for example, to set custom boot options that the running system does not already use), see the **boom(8)** man page on your system and the **boom profile create** subcommand.

2. Upgrade to Red Hat Enterprise Linux 10 using the Leapp utility:

   ```
   # leapp upgrade
   ==> Processing phase `configuration_phase`
   ====> * ipu_workflow_config
         IPU workflow config actor
   ==> Processing phase `FactsCollection`
   …

   ====> * add_upgrade_boot_entry
       Add new boot entry for Leapp provided initramfs.
   A reboot is required to continue. Please reboot your system.
   Debug output written to /var/log/leapp/leapp-upgrade.log
   ============================================================
              REPORT OVERVIEW
   ============================================================
   HIGH and MEDIUM severity reports:
    1. Detected modified configuration files in leapp configuration directories.
    2. Leapp detected loaded kernel drivers which are no longer maintained in RHEL 10.
    3. GRUB2 core will be automatically updated during the upgrade
    4. Detected customized configuration for dynamic linker.
    5. Packages not signed by Red Hat found on the system
   ```

```
Reports summary:
 Errors:                 0
 Inhibitors:             0
 HIGH severity reports:  5
 MEDIUM severity reports: 0
 LOW severity reports:   3
 INFO severity reports:  2

Before continuing, review the full report below for details about discovered problems and
possible remediation instructions:
 A report has been generated at /var/log/leapp/leapp-report.txt
 A report has been generated at /var/log/leapp/leapp-report.json


==============================================================
              END OF REPORT OVERVIEW
==============================================================

Answerfile has been generated at /var/log/leapp/answerfile
Reboot the system to continue with the upgrade. This might take a while depending on the
system configuration.
Make sure you have console access to view the actual upgrade process.
```

Review and resolve any blockers indicated by the **leapp upgrade** command report.

3. Reboot into the **upgrade** boot entry:

```
# leapp upgrade --reboot
==> Processing phase `configuration_phase`
====> * ipu_workflow_config
       IPU workflow config actor
==> Processing phase `FactsCollection`
...
```

The **Red Hat Enterprise Linux Upgrade Initramfs** is marked as the default and will boot automatically.

Verification

- After completing the upgrade, the system automatically reboots. The GRUB screen shows the upgraded (Red Hat Enterprise Linux 10) and the earlier version of the available operating system. The upgraded system version is the default selection.

Additional resources

- Reviewing the pre-upgrade report

7.4. SWITCHING BETWEEN RED HAT ENTERPRISE LINUX VERSIONS

Access the current and earlier Red Hat Enterprise Linux versions on your machine. Using Snapshot Manager and the entries created in the GRUB boot loader menu to access different operating system versions reduces the risk associated with upgrading an operating system, and also helps reduce downtime. With this ability to switch between environments, you can:

- Quickly compare both environments.

- Switch between environments while evaluating the result of the upgrade.

- Recover older content of the file system.

- Continue accessing the old system, while the upgraded host is running.

- Halt and revert the update process at any time, even if the upgrade itself fails.

Procedure

1. Reboot the system:

   ```
   # reboot
   ```

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

Verification

- Check the kernel version:

  - For the snapshot environment:

    ```
    # uname -r
    5.14.0-558.el9.x86_64
    ```

  - For the upgraded environment:

    ```
    # uname -r
    6.12.0-83.el10.x86_64
    ```

- Check the root device:

  - For the snapshot environment:

    ```
    # findmnt /
    TARGET SOURCE                                     FSTYPE OPTIONS
    /     /dev/mapper/rhel-root-snapset_before-upgrade_1745441237_- xfs
    rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota
    ```

  - For the upgraded environment:

    ```
    # findmnt /
    TARGET SOURCE          FSTYPE OPTIONS
    /     /dev/mapper/rhel-root xfs
    rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota
    ```

7.5. DELETING THE SNAPSHOT SET AFTER A SUCCESSFUL UPGRADE

If you have successfully upgraded your system by using Snapshot Manager, you can remove the snapshot set. Deleting the snapshot set automatically deletes any corresponding boot entries.

> **IMPORTANT**
>
> You cannot perform any further operations with the snapshot set after you delete it.

Prerequisites

- You have successfully upgraded your Red Hat Enterprise Linux to a later version by using Snapshot Manager and Leapp.

Procedure

1. Boot into Red Hat Enterprise Linux 10 from the GRUB boot loader screen.

2. After the system loads, view the available snapshot sets. The following output shows the **before-upgrade** snapshot set in the list of snapshot sets:

   ```
   # snapm snapset list
   SnapsetName      Time                 NrSnapshots Status  Sources
   before-upgrade   2025-04-23 21:47:17          2 Active  /, /var
   ```

3. Delete the snapshot set by name:

   ```
   # snapm snapset delete before-upgrade
   ```

4. Complete the remaining post-upgrade tasks.

Additional resources

- Upgrading from RHEL 9 to RHEL 10

7.6. REVERTING AFTER AN UNSUCCESSFUL UPDATE

To revert the operating system upgrade back to the previous state of the system after an unsuccessful upgrade, use the **snapm snapset revert** command. This can also be helpful if you find a problem with the upgraded environment, for example, incompatibility with in-house software. To start the revert, use the snapshot boot environment.

Prerequisites

- You have a snapshot set. For instructions for creating a snapshot, see Upgrading to another version by using Snapshot Manager.

Procedure

1. Start the revert operation:

   ```
   # snapm snapset revert before-upgrade
     Delaying merge since origin is open.
     Merging of snapshot rhel/root-snapset_before-upgrade_1745441237_- will occur on next
   activation of rhel/root.
     Delaying merge since origin is open.
     Merging of snapshot rhel/var-snapset_before-upgrade_1745441237_-var will occur on next
   activation of rhel/var.
   ```

> WARNING - Snaphot set before-upgrade is in use: reboot required to complete revert
> WARNING - Boot into 'Revert before-upgrade 2025-04-23 21:47:17 (5.14.0-558.el9.x86_64)'
> to continue

> ⚠ **WARNING**
>
> After you revert the snapshot set, you must continue with all the remaining
> steps in this procedure to prevent data loss.

2. Reboot your machine to restore the operating system state:

   ```
   # reboot
   ```

3. Select the **Revert** boot entry from the GRUB screen. When the system volumes are active, the system automatically starts the snapshot revert operation.
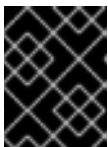
   > **IMPORTANT**
   >
   > When the merge operation starts, the snapshot volume is no longer available. Starting the revert operation automatically deletes the snapshot boot entry.

4. After you complete the merge operation, remove the unused entries:

   ```
   # rm -f /boot/loader/entries/*.el10*
   ```

   ```
   # rm -f /boot/*.el10*
   ```

5. After a successful revert to the system, delete the revert boot entry by using the **boom** command:

   ```
   # boom list -o+title
   BootID  Version              Name               RootDevice         Title
   34d576e 5.14.0-546.el9.x86_64    Red Hat Enterprise Linux /dev/mapper/rhel-root Red Hat
   Enterprise Linux (5.14.0-546.el9.x86_64) 9.6 (Plow)
   b737685 5.14.0-558.el9.x86_64    Red Hat Enterprise Linux /dev/mapper/rhel-root Red Hat
   Enterprise Linux (5.14.0-558.el9.x86_64) 9.6 (Plow)
   6703c58 5.14.0-558.el9.x86_64    Red Hat Enterprise Linux /dev/rhel/root       Revert before-
   upgrade 2025-04-23 21:47:17 (5.14.0-558.el9.x86_64)
   ```

   ```
   # boom delete 6703c58
   Deleted 1 entry
   ```

CHAPTER 8. CUSTOMIZING THE LVM REPORT

LVM provides a wide range of configuration and command line options to produce customized reports. You can sort the output, specify units, use selection criteria, and update the **lvm.conf** file to customize the LVM report.

8.1. CONTROLLING THE FORMAT OF THE LVM DISPLAY

LVM command output can be customized by controlling displayed fields, sorting options, and formatting for **pvs**, **lvs**, and **vgs** commands by using specific arguments.

When you use the **pvs**, **lvs**, or **vgs** command without additional options, you see the default set of fields displayed in the default sort order. The default fields for the **pvs** command include the following information sorted by the name of physical volumes:

```
# pvs
  PV        VG             Fmt   Attr PSize   PFree
  /dev/vdb1 VolumeGroupName lvm2  a--  17.14G  17.14G
  /dev/vdb2 VolumeGroupName lvm2  a--  17.14G  17.09G
  /dev/vdb3 VolumeGroupName lvm2  a--  17.14G  17.14G
```

- **PV**: Physical volume name.

- **VG**: Volume group name.

- **Fmt**: Metadata format of the physical volume: **lvm2** or **lvm1**.

- **Attr**: Status of the physical volume: (a) – allocatable or (x) – exported.

- **PSize**: Size of the physical volume.

- **PFree**: Free space remaining on the physical volume.

  ### Displaying custom fields

  To display a different set of fields than the default, use the **-o** option. The following example displays only the name, size and free space of the physical volumes:

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

  ### Sorting the LVM display

  To sort the results by specific criteria, use the **-O** option. The following example sorts the entries by the free space of their physical volumes in ascending order:

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

To sort the results by descending order, use the **-O** option along with the **-** character:

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

Additional resources

- Specifying the units for an LVM display

- Customizing the LVM configuration file

8.2. SPECIFYING THE UNITS FOR AN LVM DISPLAY

LVM display commands support customization of size unit output by using base 2, base 10, or custom units with the **--units** argument for flexible formatting.

See the following table for all arguments:

Units type	Description	Available options	Default

Units type	Description	Available options	Default
Base 2 units	Units are displayed in powers of 2 (multiples of 1024).	**b**: Bytes. **s**: Sectors, 512 bytes each. **k**: Kibibytes. **m**: Mebibytes. **g**: Gibibytes. **t**: Tebibytes. **p**: Pebibytes. **e**: Exbibytes. **h**: Human-readable, the most suitable unit is used. **r**: Human-readable with rounding indicator, works similarly to **h** with rounding prefix **<** or **>** to indicate how LVM rounds the displayed size to the nearest unit.	**r** (when **--units** is not specified). You can override the default by setting the units parameter in the global section of the **/etc/lvm/lvm.conf** file.

Units type	Description	Available options	Default
Base 10 units	Units are displayed in multiples of 1000.	**B**: Bytes. **S**: Sectors, 512 bytes each. **K**: Kilobytes. **M**: Megabytes. **G**: Gigabytes. **T**: Terabytes. **P**: Petabytes. **E**: Exabytes. **H**: Human-readable, the most suitable unit is used. **R**: Human-readable with rounding indicator, works similarly to **H** with rounding prefix **<** or **>** to indicate how LVM rounds the displayed size to the nearest unit.	N/A
Custom units	Combination of a quantity with a base 2 or base 10 unit. For example, to display the results in 4 mebibytes, use **4m**.	N/A	N/A

- If you do not specify a value for the units, human-readable format (**r**) is used by default. The following **vgs** command displays the size of VGs in human-readable format. The most suitable unit is used and the rounding indicator **<** shows that the actual size is an approximation and it is less than 931 gibibytes.

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

- The following **pvs** command displays the output in base 2 gibibyte units for the   **/dev/vdb** physical volume:

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

- The following **pvs** command displays the output in base 10 gigabyte units for the **/dev/vdb** physical volume:

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

- The following **pvs** command displays the output in 512-byte sectors:

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

- You can specify custom units for an LVM display command. The following example displays the output of the **pvs** command in units of 4 mebibytes:

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

8.3. CUSTOMIZING THE LVM CONFIGURATION FILE

You can customize your LVM configuration according to your specific storage and system requirements by editing the **lvm.conf** file. For example, you can edit the **lvm.conf** file to modify filter settings, configure volume group auto activation, manage a thin pool, or automatically extend a snapshot.

Procedure

1. Open the **lvm.conf** file in an editor of your choice.

2. Customize the **lvm.conf** file by uncommenting and modifying the setting for which you want to modify the default display values.

   - To customize what fields you see in the **lvs** output, uncomment the **lvs_cols** parameter and modify it:

     ```
     lvs_cols="lv_name,vg_name,lv_attr"
     ```

   - To hide empty fields for the **pvs**, **vgs**, and **lvs** commands, uncomment the **compact_output=1** setting:

     ```
     compact_output = 1
     ```

   - To set gigabytes as the default unit for the **pvs**, **vgs**, and **lvs** commands, replace the **units = "r"** setting with **units = "G"**:

     ```
     units = "G"
     ```

3. Ensure that the corresponding section of the **lvm.conf** file is uncommented. For example, to modify the **lvs_cols** parameter, the **report** section must be uncommented:

```
report {
...
}
```

Verification

- View the changed values after modifying the **lvm.conf** file:

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

8.4. DEFINING LVM SELECTION CRITERIA

LVM selection criteria use the **-S** option to filter and process physical volumes, volume groups, and logical volumes based on specific field values and operators.

Selection criteria are a set of statements in the form of **<field> <operator> <value>**, which use comparison operators to define values for specific fields. Objects that match the selection criteria are then processed or displayed. Objects can be physical volumes (PVs), volume groups (VGs), or logical volumes (LVs). Statements are combined by logical and grouping operators.

To define selection criteria use the **-S** or **--select** option followed by one or multiple statements.

The **-S** option works by describing the objects to process, rather than naming each object. This is helpful when processing many objects and it would be difficult to find and name each object separately or when searching objects that have a complex set of characteristics. The **-S** option can also be used as a shortcut to avoid typing many names.

To see full sets of fields and possible operators, use the **lvs -S help** command. Replace **lvs** with any reporting or processing command to see the details of that command:

- Reporting commands include **pvs**, **vgs**, **lvs**, **pvdisplay**, **vgdisplay**, **lvdisplay**, and **dmsetup info -c**.

- Processing commands include **pvchange**, **vgchange**, **lvchange**, **vgimport**, **vgexport**, **vgremove**, and **lvremove**.

  Examples of selection criteria using the pvs commands

  - The following example of the **pvs** command displays only physical volumes with a name that contains the string **nvme**:

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

  - The following example of the **pvs** command displays only physical devices in the **myvg** volume group:

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

```
/dev/vdb2   myvg lvm2 a--  1020.00m 896.00m
```

Examples of selection criteria using the lvs commands

- The following example of the **lvs** command displays only logical volumes with a size greater than 100m but less than 200m:

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

- The following example of the **lvs** command displays only logical volumes with a name that contains **lvol** and any number between 0 and 2:

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

- The following example of the **lvs** command displays only logical volumes with a **raid1** segment type:

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

Advanced examples

You can combine selection criteria with other options.

- The following example of the **lvchange** command adds a specific tag **mytag** to only active logical volumes:

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

- The following example of the **lvs** command displays all logical volumes whose name does not match **_pmspare** and changes the default headers to custom ones:

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

- The following example of the **lvchange** command flags a logical volume with **role=thinsnapshot** and **origin=thin1** to be skipped during normal activation commands:

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

- The following example of the **lvs** command displays only logical volumes that match all three conditions:

  - Name contains **_tmeta**.

  - Role is **metadata**.

  - Size is less or equal to 4m.

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

CHAPTER 9. CONFIGURING LVM ON SHARED STORAGE

Shared storage can be accessed by multiple nodes at the same time. You can use LVM to manage shared storage. It is commonly used in cluster and high-availability setups.

There are two common scenarios for how shared storage appears on the system:

- LVM devices are attached to a host and passed to a guest VM to use. In this case, the device is never intended to be used by the host, only by the guest VM.

- Machines are attached to a storage area network (SAN), for example using Fiber Channel, and the SAN LUNs are visible to multiple machines:

9.1. CONFIGURING LVM FOR VM DISKS

To prevent VM storage from being exposed to the host, you can configure LVM device access and LVM **system ID**. You can do this by excluding the devices in question from the host, which ensures that the LVM on the host does not see or use the devices passed to the guest VM. You can protect against accidental usage of the VM's VG on the host by setting the LVM **system ID** in the VG to match the guest VM.

Procedure

1. In the **lvm.conf** file, check if the **system.devices** file is enabled:

   ```
   use_devicesfile=1
   ```

2. Exclude the devices in question from the host's devices file:

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

3. Optional: You can further protect LVM devices:

   a. Set the LVM **system ID** feature in both the host and the VM in the **lvm.conf** file:

      ```
      system_id_source = "uname"
      ```

   b. Set the VG's **system ID** to match the VM **system ID**. This ensures that only the guest VM is capable of activating the VG:

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

9.2. CONFIGURING LVM TO USE SAN DISKS ON ONE MACHINE

To prevent the SAN LUNs from being used by the wrong machine, exclude the LUNs from the devices file on all machines except the one machine which is meant to use them.

Procedure

1. In the **lvm.conf** file, check if the **system.devices** file is enabled:

   ```
   use_devicesfile=1
   ```

2. Exclude the devices in question from the host's devices file:

> $ lvmdevices --deldev <device>

3. Set the LVM **system ID** feature in the **lvm.conf** file:

> system_id_source = "uname"

4. Set the VG's **system ID** to match the **system ID** of the machine using this VG:

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

9.3. CONFIGURING LVM TO USE SAN DISKS FOR FAILOVER

You can configure LUNs to be moved between machines, for example for failover purposes. You can set up the LVM by configuring the LVM devices file and including the LUNs in the devices file on all machines that may use the devices and by configuring the LVM **system ID** on each machine.

Procedure

1. In the **lvm.conf** file, check if the **system.devices** file is enabled:

> use_devicesfile=1

2. Include the devices in question in the host's devices file:

> $ lvmdevices --adddev *<device>*

3. Set the LVM **system ID** feature in all machines in the **lvm.conf** file:

> system_id_source = "uname"

Additional resources

- Configuring and managing high availability clusters

9.4. CONFIGURING LVM TO SHARE SAN DISKS AMONG MULTIPLE MACHINES

Using the **lvmlockd** daemon and a lock manager such as **sanlock**, you can enable access to a shared VG on the SAN disks from multiple machines. The specific commands may differ based on the lock manager and operating system used.

> **WARNING**
>
> When using **pacemaker**, the system must be configured and started using the pacemaker steps shown in Configuring and managing high availability clusters instead.

Procedure

1. In the **lvm.conf** file, check if the **system.devices** file is enabled:

   ```
   use_devicesfile=1
   ```

2. For each machine that will use the shared LUN, add the LUN in the machines devices file:

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

3. Configure the **lvm.conf** file to use the **lvmlockd** daemon on all machines:

   ```
   use_lvmlockd=1
   ```

4. Start the **lvmlockd** daemon file on all machines.

5. Start a lock manager to use with **lvmlockd**, such as **sanlock** on all machines.

6. Create a new shared VG using the command **vgcreate --shared**.

7. Start and stop access to existing shared VGs using the commands **vgchange --lockstart** and **vgchange --lockstop** on all machines.

9.5. CREATING SHARED LVM DEVICES USING THE ꜱᴛᴏʀᴀɢᴇ RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to create shared LVM devices if you want your multiple systems to access the same storage at the same time.

This can bring the following notable benefits:

- Resource sharing

- Flexibility in managing storage resources

- Simplification of storage management tasks

Prerequisites

- You have prepared the control node and the managed nodes .

- You are logged in to the control node as a user who can run playbooks on the managed nodes.

- The account you use to connect to the managed nodes has **sudo** permissions for these nodes.

- **lvmlockd** is configured on the managed node. For more information, see Configuring LVM to share SAN disks among multiple machines.

- You have a working cluster environment with shared storage and the storage RHEL system role enabled on each node.

Procedure

1. Create a playbook file, for example, ~/**playbook.yml**, with the following content:

   ```
   ---
   - name: Manage local storage
     hosts: managed-node-01.example.com
     become: true
     tasks:
       - name: Create shared LVM device
         ansible.builtin.include_role:
           name: redhat.rhel_system_roles.storage
         vars:
           storage_pools:
             - name: vg1
               disks: /dev/vdb
               type: lvm
               shared: true
               state: present
               volumes:
                 - name: lv1
                   size: 4g
                   mount_point: /opt/test1
                   fs_type: gfs2
           storage_safe_mode: false
           storage_use_partitions: true
   ```

   For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

2. Validate the playbook syntax:

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

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

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

CHAPTER 10. CONFIGURING RAID LOGICAL VOLUMES

You can create and manage Redundant Array of Independent Disks (RAID) volumes by using logical volume manager (LVM).

LVM supports RAID levels 0, 1, 4, 5, 6, and 10. An LVM RAID volume has the following characteristics:

- LVM creates and manages RAID logical volumes that leverage the Multiple Devices (MD) kernel drivers.

- You can temporarily split RAID1 images from the array and merge them back into the array later.

- LVM RAID volumes support snapshots.

- RAID logical volumes are not cluster-aware. Although you can create and activate RAID logical volumes exclusively on one machine, you cannot activate them simultaneously on more than one machine.

- When you create a RAID logical volume (LV), LVM creates a metadata subvolume that is one extent in size for every data or parity subvolume in the array. For example, creating a 2-way RAID1 array results in two metadata subvolumes (**lv_rmeta_0** and **lv_rmeta_1**) and two data subvolumes (**lv_rimage_0** and **lv_rimage_1**).

- Adding integrity to a RAID LV reduces or prevents soft corruption.

10.1. RAID LEVELS AND LINEAR SUPPORT

RAID and linear storage levels offer various data protection, performance, and capacity options. Compare RAID levels 0, 1, 4, 5, 6, 10, and linear arrays to determine the best configuration for reliable storage in Linux environments.

The following are the supported configurations by RAID, including levels 0, 1, 4, 5, 6, 10, and linear:

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. 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 the total capacity of all disks. If the member disks have a different size, then the RAID0 uses all the space of those disks using the available zones.

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 relatively high costs. RAID level 1 is costly because you write the same information to all of the disks in the array, which 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, which is parity-based RAID levels that 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.
Since 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 it is used in specific circumstances where the system administrator is intentionally designing the software RAID device with this bottleneck in mind such as an array that has 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 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. The performance of a RAID level 4 array is always asymmetrical, which means reads outperform writes. This is because write operations consume extra CPU resources and main memory bandwidth when generating parity, and then also consume extra bus bandwidth when writing the actual data to disks because you are not only writing the data, but also the parity. Read operations need only read the data and not the parity unless the array is in a degraded state. As a result, read operations 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. Modern CPUs can calculate parity very fast. However, if you have a large number of disks in a RAID 5 array such that the combined aggregate data transfer speed across all devices is high enough, parity calculation can be a bottleneck.
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 two devices instead of one 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 reduces some of the space wasted in level 1 arrays with more than two devices. With

level 10, it is possible, for example, to create a 3-drive array configured to store only two 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, similar to a 3-device, level 1 array. This avoids CPU process usage to calculate parity similar to 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 and data can be lost. The capacity is the total of all member disks.

10.2. LVM RAID SEGMENT TYPES

LVM supports various RAID segment types for the lvcreate **--type** argument, including RAID1, RAID4, RAID5, RAID6, and RAID10 configurations with distinct characteristics.

The following table describes the possible RAID segment types.

Table 10.1. LVM RAID segment types

Segment type	Description
raid1	RAID1 mirroring. This is the default value for the **--type** argument of the **lvcreate** command, when you specify the **-m** argument without specifying striping.
raid4	RAID4 dedicated parity disk.
raid5_la	- RAID5 left asymmetric. - Rotating parity 0 with data continuation.
raid5_ra	- RAID5 right asymmetric. - Rotating parity N with data continuation.
raid5_ls	- RAID5 left symmetric. - It is same as **raid5**. - Rotating parity 0 with data restart.

Segment type	Description
raid5_rs	- RAID5 right symmetric. - Rotating parity N with data restart.
raid6_zr	- RAID6 zero restart. - It is same as **raid6**. - Rotating parity zero (left-to-right) with data restart.
raid6_nr	- RAID6 N restart. - Rotating parity N (left-to-right) with data restart.
raid6_nc	- RAID6 N continue. - Rotating parity N (left-to-right) with data continuation.
raid10	- Striped mirrors. This is the default value for the **--type** argument of the **lvcreate** command if you specify the **-m** argument along with the number of stripes that is greater than 1. - Striping of mirror sets.
raid0/raid0_meta	Striping. RAID0 spreads logical volume data across multiple data subvolumes in units of stripe size. This is used to increase performance. Logical volume data is lost if any of the data subvolumes fail.

10.3. PARAMETERS FOR CREATING A RAID0

You can create a RAID0 striped logical volume using the **lvcreate --type raid0[*meta] --stripes _Stripes --stripesize *StripeSize VolumeGroup [PhysicalVolumePath*]** command.

The following table describes different parameters, which you can use while creating a RAID0 striped logical volume.

Table 10.2. Parameters for creating a RAID0 striped logical volume

Parameter	Description

Parameter	Description
--type raid0[_meta]	Specifying **raid0** creates a RAID0 volume without metadata volumes. Specifying **raid0_meta** creates a RAID0 volume with metadata volumes. Since RAID0 is non-resilient, it does not store any mirrored data blocks as RAID1/10 or calculate and store any parity blocks as RAID4/5/6 do. Therefore it does not need metadata volumes to keep state about resynchronization progress of mirrored or parity blocks. Metadata volumes become mandatory on a conversion from RAID0 to RAID4/5/6/10. Specifying **raid0_meta** preallocates those metadata volumes to prevent a respective allocation failure.
--stripes *Stripes*	Specifies the number of devices to spread the logical volume across.
--stripesize *StripeSize*	Specifies the size of each stripe in kilobytes. This is the amount of data that is written to one device before moving to the next device.
VolumeGroup	Specifies the volume group to use.
PhysicalVolumePath	Specifies the devices to use. If this is not specified, LVM will choose the number of devices specified by the *Stripes* option, one for each stripe.

10.4. CREATING RAID LOGICAL VOLUMES

You can create RAID1 arrays with multiple numbers of copies, according to the value you specify for the **-m** argument. Similarly, you can specify the number of stripes for a RAID 0, 4, 5, 6, and 10 logical volume with the **-i** argument. You can also specify the stripe size with the **-I** argument. The following procedure describes different ways to create different types of RAID logical volume.

Procedure

- Create a 2-way RAID. The following command creates a 2-way RAID1 array, named *my_lv*, in the volume group *my_vg*, that is *1G* in size:

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

- Create a RAID5 array with stripes. The following command creates a RAID5 array with three stripes and one implicit parity drive, named *my_lv*, in the volume group *my_vg*, that is *1G* in size. Note that you can specify the number of stripes similar to an LVM striped volume. The correct number of parity drives is added automatically.

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

- Create a RAID6 array with stripes. The following command creates a RAID6 array with three 3 stripes and two implicit parity drives, named *my_lv*, in the volume group *my_vg*, that is *1G* one gigabyte in size:

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

Verification

- Display the LVM device *my_vg/my_lv*, which is a 2-way RAID1 array:

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

10.5. CONFIGURING AN LVM VOLUME GROUP ON RAID BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to configure LVM volume groups on RAID arrays.

Prerequisites

- You have prepared the control node and the managed nodes .

- You are logged in to the control node as a user who can run playbooks on the managed nodes.

- The account you use to connect to the managed nodes has **sudo** permissions for these nodes.

Procedure

1. Create a playbook file, for example, ~/**playbook.yml**, with the following content:

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

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

> **NOTE**
>
> Setting **raid_level** at the **storage_pool** level creates an MD RAID array first, and then builds an LVM volume group on top of it.

2. Validate the playbook syntax:

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

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

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

Verification

- Verify that your pool is on RAID:

```
# ansible managed-node-01.example.com -m command -a 'lsblk'
```

Additional resources

- Managing RAID

10.6. CREATING A RAID0 STRIPED LOGICAL VOLUME

A RAID0 logical volume spreads logical volume data across multiple data subvolumes in units of stripe size. The following procedure creates an LVM RAID0 logical volume called *mylv* that stripes data across the disks.

Prerequisites

1. You have created three or more physical volumes. For more information about creating physical volumes, see Creating an LVM physical volume .

2. You have created the volume group. For more information, see Creating an LVM volume group .

Procedure

1. Create a RAID0 logical volume from the existing volume group. The following command creates the RAID0 volume *mylv* from the volume group *myvg*, which is *2G* in size, with three stripes and a stripe size of *4kB*:

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

2. Create a file system on the RAID0 logical volume. The following command creates an ext4 file system on the logical volume:

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

3. Mount the logical volume and report the file system disk space usage:

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

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

Verification

- View the created RAID0 stripped logical volume:

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

10.7. CONFIGURING A STRIPE SIZE FOR RAID LVM VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

You can use the **storage** RHEL system role to configure stripe sizes for RAID LVM volumes.

Prerequisites

- [You have prepared the control node and the managed nodes](#) .

- You are logged in to the control node as a user who can run playbooks on the managed nodes.

- The account you use to connect to the managed nodes has **sudo** permissions for these nodes.

Procedure

1. Create a playbook file, for example, **~/playbook.yml**, with the following content:

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure stripe size for RAID LVM volumes
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_safe_mode: false
        storage_pools:
          - name: my_pool
            type: lvm
            disks: [sdh, sdi]
            volumes:
              - name: my_volume
                size: "1 GiB"
```

```
                  mount_point: "/mnt/app/shared"
                  fs_type: xfs
                  raid_level: raid0
                  raid_stripe_size: "256 KiB"
                  state: present
```

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file on the control node.

> **NOTE**
>
> Setting **raid_level** at the **volumes** level creates LVM RAID logical volumes.

2. Validate the playbook syntax:

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

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

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

Verification

- Verify that stripe size is set to the required size:

  ```
  # ansible managed-node-01.example.com -m command -a 'lvs -o+stripesize /dev/my_pool/my_volume'
  ```

Additional resources

- [Managing RAID](#)

10.8. SOFT DATA CORRUPTION

Soft corruption in data storage implies that the data retrieved from a storage device is different from the data written to that device. The corrupted data can exist indefinitely on storage devices. You might not discover this corrupted data until you retrieve and attempt to use this data.

Depending on the type of configuration, a Redundant Array of Independent Disks (RAID) logical volume(LV) prevents data loss when a device fails. If a device consisting of a RAID array fails, the data can be recovered from other devices that are part of that RAID LV. However, a RAID configuration does not ensure the integrity of the data itself. Soft corruption, silent corruption, soft errors, and silent errors are terms that describe data that has become corrupted, even if the system design and software continues to function as expected.

When creating a new RAID LV with DM integrity or adding integrity to an existing RAID LV, consider the following points:

- The integrity metadata requires additional storage space. For each RAID image, every 500MB data requires 4MB of additional storage space because of the checksums that get added to the data.

- While some RAID configurations are impacted more than others, adding DM integrity impacts performance due to latency when accessing the data. A RAID1 configuration typically offers better performance than RAID5 or its variants.

- The RAID integrity block size also impacts performance. Configuring a larger RAID integrity block size offers better performance. However, a smaller RAID integrity block size offers greater backward compatibility.

- There are two integrity modes available: **bitmap** or **journal**. The **bitmap** integrity mode typically offers better performance than **journal** mode.

TIP

If you experience performance issues, either use RAID1 with integrity or test the performance of a particular RAID configuration to ensure that it meets your requirements.

10.9. CREATING A RAID LOGICAL VOLUME WITH DM INTEGRITY

When you create a RAID LV with device mapper (DM) integrity or add integrity to an existing RAID logical volume (LV), it mitigates the risk of losing data due to soft corruption. Wait for the integrity synchronization and the RAID metadata to complete before using the LV. Otherwise, the background initialization might impact the LV's performance.

Device mapper (DM) integrity is used with RAID levels 1, 4, 5, 6, and 10 to mitigate or prevent data loss due to soft corruption. The RAID layer ensures that a non-corrupted copy of the data can fix the soft corruption errors.

Procedure

1. Create a RAID LV with DM integrity. The following example creates a new RAID LV with integrity named *test-lv* in the *my_vg* volume group, with a usable size of *256M* and RAID level *1*:

   > # **lvcreate --type** *raid1* **--raidintegrity y -L** *256M* **-n** *test-lv my_vg*
   > Logical volume "test-lv" created.

   NOTE

   To add DM integrity to an existing RAID LV, use the following command:

   > # **lvconvert --raidintegrity y** *my_vg/test-lv*

   Adding integrity to a RAID LV limits the number of operations that you can perform on that RAID LV.

2. Optional: Remove the integrity before performing certain operations.

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

Verification

- View information about the added DM integrity:

  - View information about the test-lv RAID LV that was created in the *my_vg* volume group:

    ```
    # lvs -a my_vg
      LV                      VG     Attr       LSize   Origin                    Cpy%Sync
      test-lv                 my_vg rwi-a-r--- 256.00m                            2.10
      [test-lv_rimage_0]      my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 93.75
      [test-lv_rimage_0_imeta]  my_vg ewi-ao----   8.00m
      [test-lv_rimage_0_iorig]  my_vg -wi-ao---- 256.00m
      [test-lv_rimage_1]      my_vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 85.94
      [...]
    ```

    The following describes different options from this output:

    g attribute

    It is the list of attributes under the Attr column indicates that the RAID image is using integrity. The integrity stores the checksums in the **_imeta** RAID LV.

    Cpy%Sync column

    It indicates the synchronization progress for both the top level RAID LV and for each RAID image.

    RAID image

    It is is indicated in the LV column by **raid_image_N**.

    LV column

    It ensures that the synchronization progress displays 100% for the top level RAID LV and for each RAID image.

  - Display the type for each RAID LV:

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

  - There is an incremental counter that counts the number of mismatches detected on each RAID image. View the data mismatches detected by integrity from **rimage_0** under *my_vg/test-lv*:

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

    In this example, the integrity has not detected any data mismatches and thus the **IntegMismatches** counter shows zero (0).

○ View the data integrity information in the **/var/log/messages** log files.
For example, in case of dm–integrity mismatches you see the following output:

> device-mapper: integrity: dm-12: Checksum failed at sector 0x24e7

In case of dm–integrity data corrections you see the following:

> md/raid1:mdX: read error corrected (8 sectors at 9448 on dm-16)

10.10. CONVERTING A RAID LOGICAL VOLUME TO ANOTHER RAID LEVEL

LVM supports RAID takeover, which means converting a RAID logical volume from one RAID level to another, for example, from RAID 5 to RAID 6. You can change the RAID level to increase or decrease resilience to device failures.

Procedure

1. Create a RAID logical volume:

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

2. View the RAID logical volume:

   > # lvs -a -o +devices,segtype
   > LV   VG   Attr   LSize   Pool Origin Data% Meta% Move Log Cpy%Sync Convert Devices   Type
   >
   >   my_lv   my_vg rwi-a-r--- 504.00m   100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0) raid5
   >   [my_lv_rimage_0] my_vg iwi-aor--- 168.00m   /dev/sda(1) linear
   >   [my_lv_rimage_1] my_vg iwi-aor--- 168.00m   /dev/sdb(1) linear
   >   [my_lv_rimage_2] my_vg iwi-aor--- 168.00m   /dev/sdc(1) linear
   >   [my_lv_rimage_3] my_vg iwi-aor--- 168.00m   /dev/sdd(1) linear
   >   [my_lv_rmeta_0] my_vg ewi-aor---   4.00m   /dev/sda(0) linear
   >   [my_lv_rmeta_1] my_vg ewi-aor---   4.00m   /dev/sdb(0) linear
   >   [my_lv_rmeta_2] my_vg ewi-aor---   4.00m   /dev/sdc(0) linear
   >   [my_lv_rmeta_3] my_vg ewi-aor---   4.00m   /dev/sdd(0) linear

3. Convert the RAID logical volume to another RAID level:

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

4. Optional: If this command prompts to repeat the conversion, run:

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

Verification

1. View the RAID logical volume with the converted RAID level:

```
# lvs -a -o +devices,segtype
  LV              VG     Attr      LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices                                                    Type
  my_lv           my_vg  rwi-a-r--- 504.00m                     100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0),my_lv_rimage_
4(0) raid6
  [my_lv_rimage_0] my_vg  iwi-aor--- 168.00m                                        /dev/sda(2)
linear
  [my_lv_rimage_1] my_vg  iwi-aor--- 168.00m                                        /dev/sdb(2)
linear
  [my_lv_rimage_2] my_vg  iwi-aor--- 168.00m                                        /dev/sdc(2)
linear
  [my_lv_rimage_3] my_vg  iwi-aor--- 168.00m                                        /dev/sdd(2)
linear
  [my_lv_rimage_4] my_vg  iwi-aor--- 168.00m                                        /dev/sde(2)
linear
  [my_lv_rmeta_0]  my_vg  ewi-aor---   4.00m                                        /dev/sda(0)
linear
  [my_lv_rmeta_1]  my_vg  ewi-aor---   4.00m                                        /dev/sdb(0)
linear
  [my_lv_rmeta_2]  my_vg  ewi-aor---   4.00m                                        /dev/sdc(0)
linear
  [my_lv_rmeta_3]  my_vg  ewi-aor---   4.00m                                        /dev/sdd(0)
linear
  [my_lv_rmeta_4]  my_vg  ewi-aor---   4.00m                                        /dev/sde(0)
linear
```

10.11. CONVERTING A LINEAR DEVICE TO A RAID LOGICAL VOLUME

You can convert an existing linear logical volume to a RAID logical volume. To perform this operation, use the **--type** argument of the **lvconvert** command.

RAID logical volumes are composed of metadata and data subvolume pairs. When you convert a linear device to a RAID1 array, it creates a new metadata subvolume and associates it with the original logical volume on one of the same physical volumes that the linear volume is on. The additional images are added in a metadata/data subvolume pair. If the metadata image that pairs with the original logical volume cannot be placed on the same physical volume, the **lvconvert** fails.

Procedure

1. View the logical volume device that needs to be converted:

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

2. Convert the linear logical volume to a RAID device. The following command converts the linear logical volume *my_lv* in volume group __my_vg, to a 2-way RAID1 array:

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

Verification

- Ensure if the logical volume is converted to a RAID device:

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

10.12. CONVERTING AN LVM RAID1 LOGICAL VOLUME TO AN LVM LINEAR LOGICAL VOLUME

You can convert an existing RAID1 LVM logical volume to an LVM linear logical volume. To perform this operation, use the **lvconvert** command and specify the **-m0** argument. This removes all the RAID data subvolumes and all the RAID metadata subvolumes that make up the RAID array, leaving the top-level RAID1 image as the linear logical volume.

Procedure

1. Display an existing LVM RAID1 logical volume:

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

2. Convert an existing RAID1 LVM logical volume to an LVM linear logical volume. The following command converts the LVM RAID1 logical volume *my_vg/my_lv* to an LVM linear device:

   ```
   # lvconvert -m0 my_vg/my_lv
   Are you sure you want to convert raid1 LV my_vg/my_lv to type linear losing all resilience?
   ```

```
[y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

When you convert an LVM RAID1 logical volume to an LVM linear volume, you can also specify which physical volumes to remove. In the following example, the **lvconvert** command specifies that you want to remove */dev/sde1*, leaving */dev/sdf1* as the physical volume that makes up the linear device:

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

Verification

- Verify if the RAID1 logical volume was converted to an LVM linear device:

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

10.13. CONVERTING A MIRRORED LVM DEVICE TO A RAID1 LOGICAL VOLUME

You can convert an existing mirrored LVM device with a segment type mirror to a RAID1 LVM device. To perform this operation, use the **lvconvert** command with the **--type** raid1 argument. This renames the mirror subvolumes named **mimage** to RAID subvolumes named **rimage**.

In addition, it also removes the mirror log and creates metadata subvolumes named **rmeta** for the data subvolumes on the same physical volumes as the corresponding data subvolumes.

Procedure

1. View the layout of a mirrored logical volume *my_vg/my_lv*:

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

2. Convert the mirrored logical volume *my_vg/my_lv* to a RAID1 logical volume:

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

Verification

- Verify if the mirrored logical volume is converted to a RAID1 logical volume:

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
```

```
[my_lv_rimage_0]        /dev/sde1(0)
[my_lv_rimage_1]         /dev/sdf1(0)
[my_lv_rmeta_0]         /dev/sde1(125)
[my_lv_rmeta_1]         /dev/sdf1(125)
```

10.14. CHANGING THE NUMBER OF IMAGES IN AN EXISTING RAID1 DEVICE

You can change the number of images in an existing RAID1 array, similar to the way you can change the number of images in the implementation of LVM mirroring.

When you add images to a RAID1 logical volume with the **lvconvert** command, you can perform the following operations:

- specify the total number of images for the resulting device,

- how many images to add to the device, and

- can optionally specify on which physical volumes the new metadata/data image pairs reside.

Procedure

1. Display the LVM device *my_vg/my_lv*, which is a 2-way RAID1 array:

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

    Metadata subvolumes named **rmeta** always exist on the same physical devices as their data subvolume counterparts **rimage**. The metadata/data subvolume pairs will not be created on the same physical volumes as those from another metadata/data subvolume pair in the RAID array unless you specify **--alloc** anywhere.

2. Convert the 2-way RAID1 logical volume *my_vg/my_lv* to a 3-way RAID1 logical volume:

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

    The following are a few examples of changing the number of images in an existing RAID1 device:

    - You can also specify which physical volumes to use while adding an image to RAID. The following command converts the 2-way RAID1 logical volume *my_vg/my_lv* to a 3-way RAID1 logical volume by specifying the physical volume */dev/sdd1* to use for the array:

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

    - Convert the 3-way RAID1 logical volume into a 2-way RAID1 logical volume:

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

- Convert the 3-way RAID1 logical volume into a 2-way RAID1 logical volume by specifying the physical volume */dev/sde1*, which contains the image to remove:

  > **# lvconvert *-m1 my_vg/my_lv /dev/sde1***

  Additionally, when you remove an image and its associated metadata subvolume volume, any higher-numbered images will be shifted down to fill the slot. Removing **lv_rimage_1** from a 3-way RAID1 array that consists of **lv_rimage_0**, **lv_rimage_1**, and **lv_rimage_2** results in a RAID1 array that consists of **lv_rimage_0** and **lv_rimage_1**. The subvolume **lv_rimage_2** will be renamed and take over the empty slot, becoming **lv_rimage_1**.

Verification

- View the RAID1 device after changing the number of images in an existing RAID1 device:

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

10.15. SPLITTING OFF A RAID IMAGE AS A SEPARATE LOGICAL VOLUME

You can split off an image of a RAID logical volume to form a new logical volume. When you are removing a RAID image from an existing RAID1 logical volume or removing a RAID data subvolume and its associated metadata subvolume from the middle of the device, any higher numbered images will be shifted down to fill the slot. The index numbers on the logical volumes that make up a RAID array will thus be an unbroken sequence of integers.
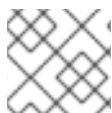
> **NOTE**
>
> You cannot split off a RAID image if the RAID1 array is not yet in sync.

Procedure

1. Display the LVM device *my_vg/my_lv*, which is a 2-way RAID1 array:

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

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

2. Split the RAID image into a separate logical volume:

   - The following example splits a 2-way RAID1 logical volume, *my_lv*, into two linear logical volumes, *my_lv* and *new*:

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

   - The following example splits a 3-way RAID1 logical volume, *my_lv*, into a 2-way RAID1 logical volume, *my_lv*, and a linear logical volume, *new*:

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

Verification

- View the logical volume after you split off an image of a RAID logical volume:

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

10.16. SPLITTING AND MERGING A RAID IMAGE

You can temporarily split off an image of a RAID1 array for read-only use while tracking any changes by using the **--trackchanges** argument with the **--splitmirrors** argument of the **lvconvert** command. Using this feature, you can merge the image into an array at a later time while resyncing only those portions of the array that have changed since the image was split.

When you split off a RAID image with the **--trackchanges** argument, you can specify which image to split but you cannot change the name of the volume being split. In addition, the resulting volumes have the following constraints:

- The new volume you create is read-only.

- You cannot resize the new volume.

- You cannot rename the remaining array.

- You cannot resize the remaining array.

- You can activate the new volume and the remaining array independently.

You can merge an image that was split off. When you merge the image, only the portions of the array that have changed since the image was split are resynced.

Procedure

1. Create a RAID logical volume:

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

2. Optional: View the created RAID logical volume:

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

3. Split an image from the created RAID logical volume and track the changes to the remaining array:

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

4. Optional: View the logical volume after splitting the image:

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

5. Merge the volume back into the array:

```
# lvconvert --merge my_vg/my_lv_rimage_2
my_vg/my_lv_rimage_2 successfully merged back into my_vg/my_lv
```

Verification

- View the merged logical volume:

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

10.17. SETTING THE RAID FAULT POLICY TO ALLOCATE

You can set the **raid_fault_policy** field to the **allocate** parameter in the **/etc/lvm/lvm.conf** file. With this preference, the system attempts to replace the failed device with a spare device from the volume group. If there is no spare device, the system log includes this information.

Procedure

1. View the RAID logical volume:

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

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

2. View the RAID logical volume if the */dev/sdb* device fails:

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

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

   You can also view the system log for the error messages if the */dev/sdb* device fails.

3. Set the **raid_fault_policy** field to **allocate** in the **lvm.conf** file:

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

   > **NOTE**
   >
   > If you set **raid_fault_policy** to **allocate** but there are no spare devices, the allocation fails, leaving the logical volume as it is. If the allocation fails, you can fix and replace the failed device by using the **lvconvert --repair** command.

Verification

- Verify if the failed device is now replaced with a new device from the volume group:

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

> **NOTE**
>
> Even though the failed device is now replaced, the display still indicates that LVM could not find the failed device because the device is not yet removed from the volume group. You can remove the failed device from the volume group by executing the **vgreduce --removemissing** *my_vg* command.

Additional resources

- [Replacing a failed RAID device in a logical volume](#)

10.18. SETTING THE RAID FAULT POLICY TO WARN

You can set the **raid_fault_policy** field to the **warn** parameter in the **lvm.conf** file. With this preference, the system adds a warning to the system log that indicates a failed device. Based on the warning, you can determine the further steps.

By default, the value of the **raid_fault_policy** field is **warn** in **lvm.conf**.

Procedure

1. View the RAID logical volume:

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

2. Set the raid_fault_policy field to warn in the lvm.conf file:

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

3. View the system log to display error messages if the */dev/sdb* device fails:

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

Alternatively, you can also open the **lvm.conf** file to display the error messages.

If the _/dev/sdb_ device fails, the system log displays error messages. In this case, however, LVM will not automatically attempt to repair the RAID device by replacing one of the images. Instead, if the device has failed you can replace the device with the **--repair** argument of the **lvconvert** command.

Additional resources

- Replacing a failed RAID device in a logical volume

10.19. REPLACING A WORKING RAID DEVICE

You can replace a working RAID device in a logical volume by using the **--replace** argument of the **lvconvert** command.

> ⚠️ **WARNING**
>
> In the case of RAID device failure, the following commands do not work.

Prerequisites

- The RAID device has not failed.

Procedure

1. Create a RAID1 array:

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

2. Examine the created RAID1 array:

   ```
   # lvs -a -o name,copy_percent,devices my_vg
   LV              Copy%  Devices
   my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
   [my_lv_rimage_0]      /dev/sdb1(1)
   [my_lv_rimage_1]       /dev/sdb2(1)
   [my_lv_rimage_2]       /dev/sdc1(1)
   [my_lv_rmeta_0]       /dev/sdb1(0)
   [my_lv_rmeta_1]       /dev/sdb2(0)
   [my_lv_rmeta_2]       /dev/sdc1(0)
   ```

3. Replace the RAID device with any of the following methods depending on your requirements:

   a. Replace a RAID1 device by specifying the physical volume that you want to replace:

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

b. Replace a RAID1 device by specifying the physical volume to use for the replacement:

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

c. Replace multiple RAID devices at a time by specifying multiple replace arguments:

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

Verification

1. Examine the RAID1 array after specifying the physical volume that you wanted to replace:

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

2. Examine the RAID1 array after specifying the physical volume to use for the replacement:

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

3. Examine the RAID1 array after replacing multiple RAID devices at a time:

```
# lvs -a -o name,copy_percent,devices my_vg
LV              Copy%  Devices
my_lv            60.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]     /dev/sda1(1)
[my_lv_rimage_1]      /dev/sdd1(1)
[my_lv_rimage_2]      /dev/sde1(1)
[my_lv_rmeta_0]      /dev/sda1(0)
[my_lv_rmeta_1]      /dev/sdd1(0)
[my_lv_rmeta_2]      /dev/sde1(0)
```

10.20. REPLACING A FAILED RAID DEVICE IN A LOGICAL VOLUME

RAID is not similar to traditional LVM mirroring. In case of LVM mirroring, remove the failed devices. Otherwise, the mirrored logical volume would hang while RAID arrays continue running with failed devices. For RAID levels other than RAID1, removing a device would mean converting to a lower RAID level, for example, from RAID6 to RAID5, or from RAID4 or RAID5 to RAID0.

Instead of removing a failed device and allocating a replacement, with LVM, you can replace a failed device that serves as a physical volume in a RAID logical volume by using the **--repair** argument of the **lvconvert** command.

Prerequisites

- The volume group includes a physical volume that provides enough free capacity to replace the failed device.
  If no physical volume with enough free extents is available on the volume group, add a new, sufficiently large physical volume by using the **vgextend** utility.

Procedure

1. View the RAID logical volume:

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

2. View the RAID logical volume after the */dev/sdc* device fails:

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

3. Replace the failed device:

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

4. Optional: Manually specify the physical volume that replaces the failed device:

   # lvconvert --repair *my_vg/my_lv replacement_pv*

5. Examine the logical volume with the replacement:

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

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

   Until you remove the failed device from the volume group, LVM utilities still indicate that LVM cannot find the failed device.

6. Remove the failed device from the volume group:

   # vgreduce --removemissing *my_vg*

Verification

1. View the available physical volumes after removing the failed device:

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

2. Examine the logical volume after the replacing the failed device:

   # lvs --all --options name,copy_percent,devices *my_vg*
   ```
   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
   [my_lv_rimage_0]       /dev/sde1(1)
   [my_lv_rimage_1]        /dev/sdb1(1)
   [my_lv_rimage_2]        /dev/sdd1(1)
   [my_lv_rmeta_0]        /dev/sde1(0)
   [my_lv_rmeta_1]        /dev/sdb1(0)
   [my_lv_rmeta_2]        /dev/sdd1(0)
   ```

10.21. CHECKING DATA COHERENCY IN A RAID LOGICAL VOLUME

LVM provides scrubbing support for RAID logical volumes. RAID scrubbing is the process of reading all the data and parity blocks in an array and checking to see whether they are coherent. The **lvchange --syncaction repair** command initiates a background synchronization action on the array.

Procedure

1. Optional: Control the rate at which a RAID logical volume is initialized by setting any one of the following options:

   - **--maxrecoveryrate Rate[bBsSkKmMgG]** sets the maximum recovery rate for a RAID logical volume so that it will not expel nominal I/O operations.

   - **--minrecoveryrate Rate[bBsSkKmMgG]** sets the minimum recovery rate for a RAID logical volume to ensure that I/O for sync operations achieves a minimum throughput, even when heavy nominal I/O is present

     > # lvchange --maxrecoveryrate 4K *my_vg/my_lv*
     > Logical volume *my_vg/my_lv_changed*

     Replace *4K* with the recovery rate value, which is an amount per second for each device in the array. If you provide no suffix, the options assume kiB per second per device.

     > # lvchange --syncaction repair *my_vg*/my_lv_

     When you perform a RAID scrubbing operation, the background I/O required by the **sync** actions can crowd out other I/O to LVM devices, such as updates to volume group metadata. This might cause the other LVM operations to slow down.

     > **NOTE**
     >
     > You can also use these maximum and minimum I/O rate while creating a RAID device. For example, **lvcreate --type raid10 -i 2 -m 1 -L 10G --maxrecoveryrate 128 -n my_lv my_vg** creates a 2-way RAID10 array my_lv, which is in the volume group my_vg with 3 stripes that is 10G in size with a maximum recovery rate of 128 kiB/sec/device.

2. Display the number of discrepancies in the array, without repairing them:

   > # lvchange --syncaction check *my_vg/my_lv*

   This command initiates a background synchronization action on the array.

3. Optional: View the **var/log/syslog** file for the kernel messages.

4. Correct the discrepancies in the array:

   > # lvchange --syncaction repair *my_vg/my_lv*

   This command repairs or replaces failed devices in a RAID logical volume. You can view the **var/log/syslog** file for the kernel messages after executing this command.

Verification

1. Display information about the scrubbing operation:

   > # **lvs -o +raid_sync_action,raid_mismatch_count** *my_vg/my_lv*
   > LV   VG   Attr   LSize   Pool Origin Data% Meta% Move Log Cpy%Sync Convert
   > SyncAction Mismatches

```
my_lv my_vg rwi-a-r--- 500.00m                    100.00        idle    0
```

10.22. I/O OPERATIONS ON A RAID1 LOGICAL VOLUME

You can control the I/O operations for a device in a RAID1 logical volume by using the **--writemostly** and **--writebehind** parameters of the **lvchange** command.

The following is the format for using these parameters:

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

Marks a device in a RAID1 logical volume as **write-mostly** and avoids all read actions to these drives unless necessary. Setting this parameter keeps the number of I/O operations to the drive to a minimum.
Use the **lvchange --writemostly** */dev/sdb my_vg/my_lv* command to set this parameter.

You can set the **writemostly** attribute in the following ways:

:y

By default, the value of the **writemostly** attribute is yes for the specified physical volume in the logical volume.

:n

To remove the **writemostly** flag, append **:n** to the physical volume.

:t

To toggle the value of the **writemostly** attribute, specify the **--writemostly** argument.
You can use this argument more than one time in a single command, for example, **lvchange --writemostly /dev/sdd1:n --writemostly /dev/sdb1:t --writemostly /dev/sdc1:y** *my_vg/my_lv*.
With this, it is possible to toggle the **writemostly** attributes for all the physical volumes in a logical volume at once.

--[raid]writebehind IOCount

Specifies the maximum number of pending writes marked as **writemostly**. These are the number of write operations applicable to devices in a RAID1 logical volume. After the value of this parameter exceeds, all write actions to the constituent devices complete synchronously before the RAID array notifies for completion of all write actions.
You can set this parameter by using the **lvchange --writebehind** *100 my_vg/my_lv* command. Setting the **writemostly** attribute's value to zero clears the preference. With this setting, the system chooses the value arbitrarily.

10.23. RESHAPING A RAID VOLUME

RAID reshaping means changing attributes of a RAID logical volume without changing the RAID level. Some attributes that you can change include RAID layout, stripe size, and number of stripes.

Procedure

1. Create a RAID logical volume:

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

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

2. View the RAID logical volume:

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

3. Optional: View the **stripes** images and **stripesize** of the RAID logical volume:

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

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

4. Modify the attributes of the RAID logical volume by using the following ways depending on your requirement:

    a. Modify the **stripes** images of the RAID logical volume:

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

    b. Modify the **stripesize** of the RAID logical volume:

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

    c. Modify the **maxrecoveryrate** and **minrecoveryrate** attributes:

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

■

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

d. Modify the **syncaction** attribute:

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

e. Modify the **writemostly** and **writebehind** attributes:

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

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

> **NOTE**
>
> All operations are not supported for all volume types. Check the specific volume type requirements and limitations before proceeding.

Verification

1. View the **stripes** images and **stripesize** of the RAID logical volume:

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

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

2. View the RAID logical volume after modifying the **maxrecoveryrate** attribute:

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

3. View the RAID logical volume after modifying the **minrecoveryrate** attribute:

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

4. View the RAID logical volume after modifying the **syncaction** attribute:

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

10.24. CHANGING THE REGION SIZE ON A RAID LOGICAL VOLUME

When you create a RAID logical volume, the **raid_region_size** parameter from the **/etc/lvm/lvm.conf** file represents the region size for the RAID logical volume. After you created a RAID logical volume, you can change the region size of the volume. This parameter defines the granularity to keep track of the dirty or clean state. Dirty bits in the bitmap define the work set to synchronize after a dirty shutdown of a RAID volume, for example, a system failure.

If you set **raid_region_size** to a higher value, it reduces the size of bitmap as well as the congestion. But it impacts the **write** operation during resynchronizing the region because writes to RAID are postponed until synchronizing the region finishes.

Procedure

1. Create a RAID logical volume:

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

2. View the RAID logical volume:

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

LV              VG     Attr     LSize Pool Origin Data% Meta% Move Log    Cpy%Sync Convert
Devices                     Region
lvol0           test rwi-a-r--- 10.00g                       100.00
lvol0_rimage_0(0),lvol0_rimage_1(0)  2.00m
[lvol0_rimage_0]  test iwi-aor--- 10.00g                                  /dev/sde1(1)
0
[lvol0_rimage_1]  test iwi-aor--- 10.00g                                  /dev/sdf1(1)
0
[lvol0_rmeta_0]   test ewi-aor---  4.00m                                   /dev/sde1(0)
0
[lvol0_rmeta_1]   test ewi-aor---  4.00m
```

The **Region** column indicates the raid_region_size parameter's value.

3. Optional: View the **raid_region_size** parameter's value:

```
# grep raid_region_size /etc/lvm/lvm.conf
# Configuration option activation/raid_region_size.
# raid_region_size = 2048
```

4. Change the region size of a RAID logical volume:

```
# lvconvert -R 4096K test/lvol0

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

5. Resynchronize the RAID logical volume:

```
# lvchange --resync test/lvol0

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

Verification

1. View the RAID logical volume:

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

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

   The **Region** column indicates the changed value of the **raid_region_size** parameter.

2. View the **raid_region_size** parameter's value in the **lvm.conf** file:

   ```
   # grep raid_region_size /etc/lvm/lvm.conf
   # Configuration option activation/raid_region_size.
   # raid_region_size = 4096
   ```

CHAPTER 11. LIMITING LVM DEVICE VISIBILITY AND USAGE

You can limit the devices that are visible and usable to Logical Volume Manager (LVM) by controlling the devices that LVM can scan.

Use LVM commands to control LVM device scanning. LVM commands interact with a file called the **system.devices** file, which lists the visible and usable devices. This feature is enabled by default in Red Hat Enterprise Linux 10.

If you disable the devices file feature, the LVM device filter is enabled automatically.

To adjust the configuration of LVM device scanning, edit the LVM device filter settings in the **/etc/lvm/lvm.conf** file. The filters in the **lvm.conf** file consist of a series of simple regular expressions. The system applies these expressions to each device name in the **/dev** directory to decide whether to accept or reject each detected block device.

11.1. THE LVM DEVICES FILE

The Logical Volume Manager (LVM) **system.devices** file controls device visibility and usability to LVM. You can find the devices file in the **/etc/lvm/devices/** directory. Use LVM commands to manage the devices file. Do not directly edit the **system.devices** file.

By default, the **system.devices** file feature is enabled in Red Hat Enterprise Linux 10. When active, it replaces the LVM device filter. To enable the LVM device filter, disable the **system.devices** file. For more information see Disabling the system.devices file.

11.1.1. Adding devices to the system.devices file

To use devices with the Logical Volume Manager (LVM), the **system.devices** file must contain a list of the device IDs, otherwise LVM ignores them. The operating system (OS) installer adds devices to the **system.devices** file during installation. A newly installed system includes the root device into the devices file automatically. Any Physical Volumes (PV) attached to the system during OS installation are also included into the devices file. You can also specifically add devices to the devices file. LVM detects and uses only the list of devices stored in the devices file.

Procedure

- Add devices to the **system.devices** file by using one of the following methods:

  - Add devices by including their names to the devices file:

    ```
    $ lvmdevices --adddev <device_name>
    ```

  - Add all devices in a Volume Group (VG) to the devices file:

    ```
    $ vgimportdevices <vg_name>
    ```

  - Add all devices in all visible VGs to the devices file:

    ```
    $ vgimportdevices --all
    ```

- To implicitly include new devices into the **system.devices** file, use one of the following commands:

- Use the **pvcreate** command to initialize a new device:

    ```
    $ pvcreate <device_name>
    ```

    This action automatically adds the new Physical Volume (PV) to the **system.devices** file.

- Initialize new devices and add the new device arguments to the devices file automatically:

    ```
    $ vgcreate <vg_name> <device_names>
    ```

    Replace *<vg_name>* with the name of the VG, from which you want to add devices. Replace *<device_names>* with a space-separated list of the devices you want to add.

- Use the **vgextend** command to initialize new devices:

    ```
    $ vgextend <vg_name> <device_names>
    ```

    Replace *<vg_name>* with the name of the VG, from which you want to add devices. Replace *<device_names>* with the names of the devices you want to add. This adds the new device arguments to the devices file automatically.

Verification

Use the following Verification only in case you need to explicitly add new devices to the **system.devices** file.

- Display the **system.devices** file, to check the list of devices:

    ```
    $ cat /etc/lvm/devices/system.devices
    ```

- Update the **system.devices** file to match most recent device information:

    ```
    $ lvmdevices --update
    ```

11.1.2. Removing devices from the system.devices file

Remove a device to prevent the Logical Volume Manager (LVM) from detecting or using that device.

Procedure

- Remove a device by using one of the following methods depending on the information you have about that device:

    - Remove a device by name:

        ```
        $ lvmdevices --deldev <device_name>
        ```

    - Remove a device by the Physical Volume ID (PVID) of the device:

        ```
        $ lvmdevices --delpvid <PV_UUID>
        ```

Verification

- Display the **system.devices** file to verify, that the deleted device no longer present:

  $ cat /etc/lvm/devices/system.devices

- Update the **system.devices** file to match most recent device information:

  $ lvmdevices --update

11.1.3. Creating custom devices files

Logical Volume Manager (LVM) commands use the default **system.devices** file of the system. You can also create and use custom devices files by specifying the new file name in the LVM commands. Custom devices files are useful in cases when only certain applications need to use certain devices.

Procedure

1. Create a custom devices file in the **/etc/lvm/devices/** directory.

2. Include the new devices file name in the LVM command:

   $ lvmdevices --devicesfile *<devices_file_name>*

3. Optional: Display the new devices file to verify that the name of the new device is present:

   $ cat /etc/lvm/devices/<devices_file_name>

11.1.4. Accessing all devices on the system

You can enable Logical Volume Manager (LVM) to access and use all devices on the system, which overrides the restrictions caused by the devices listed in the **system.devices** file.

Procedure

- Specify an empty devices file:

  $ lvmdevices --devicesfile ""

11.1.5. Disabling the system.devices file

You can disable the **system.devices** file functionality. This action automatically enables the Logical Volume Manager (LVM) device filter.

> **IMPORTANT**
>
> If you remove the **system.devices** file, this action effectively disables it. This applies even if you enable the **system.devices** file in the **lvm.conf** configuration file by setting **use_devicesfile=1** in the devices section. Disabling the devices file automatically enables the **lvm.conf** device filter.

Procedure

1. Open the **lvm.conf** file.

2. Set the following value in the devices section:

   > use_devicesfile=0

11.2. PERSISTENT IDENTIFIERS FOR LVM FILTERING

Traditional Linux device names, such as **/dev/sda**, are subject to changes during system modifications and reboots. Persistent Naming Attributes (PNAs) like World Wide Identifier (WWID), Universally Unique Identifier (UUID), and path names are based on unique characteristics of the storage devices and are resilient to changes in hardware configurations. This makes them more stable and predictable across system reboots.

Implementation of persistent device identifiers in LVM filtering enhances the stability and reliability of LVM configurations. It also reduces the risk of system boot failures associated with the dynamic nature of device names.

Additional resources

- Persistent naming attributes

- How to configure lvm filter, when local disk name is not persistent? (Red Hat Knowledgebase)

11.3. THE LVM DEVICE FILTER

The Logical Volume Manager (LVM) device filter is a list of device name patterns. You can use it to specify a set of mandatory criteria by which the system can evaluate devices and consider them as valid for use with LVM.

The LVM device filter enables you control over which devices LVM uses. This can help to prevent accidental data loss or unauthorized access to storage devices.

11.3.1. LVM device filter pattern characteristics

Device filter patterns in LVM use regular expressions preceded by **a** (accept) or **r** (reject). The first matching pattern for a device determines its acceptance or rejection by LVM.

LVM evaluates device path names against these regular expressions, using the earliest match to decide whether each device is accepted or ignored.

If a single device has multiple path names, LVM accesses these path names according to their order of listing. Before any **r** pattern, if at least one path name matches an **a** pattern, LVM approves the device. However, if all path names are consistent with an **r** pattern before an **a** pattern is found, the device is rejected.

Path names that do not match the pattern do not affect the approval status of the device. If no path names correspond to a pattern for a device, LVM still approves the device.

For each device on the system, the **udev** rules generate multiple symlinks. Directories contain symlinks, such as **/dev/disk/by-id/**, **/dev/disk/by-uuid/**, **/dev/disk/by-path/** to ensure that each device on the system is accessible through multiple path names.

To reject a device in the filter, all of the path names associated with that particular device must match

the corresponding reject **r** expressions. However, identifying all possible path names to reject can be challenging. This is why it is better to create filters that specifically accept certain paths and reject all others, using a series of specific **a** expressions followed by a single **r|.*|** expression that rejects everything else.

While defining a specific device in the filter, use a symlink name for that device instead of the kernel name. The kernel name for a device can change, such as **/dev/sda** while certain symlink names do not change such as **/dev/disk/by-id/wwn-***.

The default device filter accepts all devices connected to the system. An ideal user configured device filter accepts one or more patterns and rejects everything else. For example, the pattern list ending with **r|.*|**.

You can find the LVM devices filter configuration in the **devices/filter** and **devices/global_filter** configuration fields in the **lvm.conf** file. The **devices/filter** and **devices/global_filter** configuration fields are equivalent.

For more information, see the **lvm.conf(5)** man page on your system.
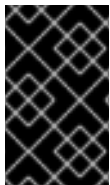
> **IMPORTANT**
>
> In Red Hat Enterprise Linux 10, the **/etc/lvm/devices/system.devices** file is enabled by default. The system automatically enables the LVM devices filter, when the **system.devices** file is disabled.

11.3.2. Examples of LVM device filter configurations

LVM device filter configurations control which devices are scanned and used by LVM, with specific patterns available for various device types and scenarios.

The following examples display the filter configurations to control the devices that LVM scans and uses later.

> **NOTE**
>
> You might encounter duplicate Physical Volume (PV) warnings when dealing with copied or cloned PVs. You can set up filters to resolve this.

- To scan all the devices, enter:

  filter = ["a|.*|"]

- To remove the **cdrom** device to avoid delays if the drive contains no media, enter:

  filter = ["r|^/dev/cdrom$|"]

- To add all loop devices and remove all other devices, enter:

  filter = ["a|loop|", "r|.*|"]

- To add all loop and SCSI devices and remove all other block devices, enter:

  filter = ["a|loop|", "a|/dev/sd.*|", "r|.*|"]

–

- To add only partition 8 on the first SCSI drive and remove all other block devices, enter:

  ```
  filter = [ "a|^/dev/sda8$|", "r|.*|" ]
  ```

- To add all partitions from a specific device identified by WWID along with all multipath devices, enter:

  ```
  filter = [ "a|/dev/disk/by-id/<disk-id>.|", "a|/dev/mapper/mpath.|", "r|.*|" ]
  ```

  The command also removes any other block devices.

Additional resources

- [Applying an LVM device filter configuration](#)

- [Example LVM device filters that prevent duplicate PV warnings](#)

11.3.3. Applying an LVM device filter configuration

You can control which devices LVM scans by setting up filters in the **lvm.conf** configuration file.

Prerequisites

- You have disabled the **system.devices** file feature.

- You have prepared the device filter pattern that you want to use.

Procedure

1. Use the following command to test the device filter pattern, without actually modifying the **/etc/lvm/lvm.conf** file. The following includes an example filter configuration.

   ```
   # lvs --config 'devices{ filter = [ "a|/dev/emcpower.|", "r|.|" ] }'
   ```

2. Add the device filter pattern in the configuration section **devices** of the **/etc/lvm/lvm.conf** file:

   ```
   filter = [ "a|/dev/emcpower.*|", "r|*.|" ]
   ```

3. Scan only necessary devices on reboot:

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

   This command rebuilds the **initramfs** file system so that LVM scans only the necessary devices at the time of reboot.

CHAPTER 12. CONTROLLING LVM ALLOCATION

By default, a volume group uses the **normal** allocation policy. This allocates physical extents according to common-sense rules like not placing parallel stripes on the same physical volume. You can change this to **contiguous**, **anywhere**, or **cling** by using the **--alloc** argument of the **vgcreate** command.

In general, allocation policies other than **normal** are required only in special cases where you need to specify unusual or nonstandard extent allocation.

12.1. ALLOCATING EXTENTS FROM SPECIFIED DEVICES

You can restrict the allocation from specific devices by using the device arguments at the end of the command line with the **lvcreate** and the **lvconvert** commands. You can specify the actual extent ranges for each device for more control. The command only allocates extents for the new logical volume (LV) by using the specified physical volume (PV) as arguments. It takes available extents from each PV until they run out and then takes extents from the next PV listed. If there is not enough space on all the listed PVs for the requested LV size, then command fails. Note that the command only allocates from the named PVs. Raid LVs use sequential PVs for separate raid images or separate stripes. If the PVs are not large enough for an entire raid image, then the resulting device use is not entirely predictable.

Procedure

1. Create a volume group (VG):

   # vgcreate *<vg_name>* *<PV>* ...

   Where:

   - ***<vg_name>*** is the name of the VG.

   - ***<PV>*** are the PVs.

2. You can allocate PV to create different volume types, such as linear or raid:

   a. Allocate extents to create a linear volume:

      # lvcreate -n *<lv_name>* -L *<lv_size>* *<vg_name>* [*<PV>* ...]

      Where:

      - ***<lv_name>*** is the name of the LV.

      - ***<lv_size>*** is the size of the LV. Default unit is megabytes.

      - ***<vg_name>*** is the name of the VG.

      - **[*<PV ...>*]** are the PVs.
        You can specify one of the PVs, all of them, or none on the command line:

        - If you specify one PV, extents for that LV will be allocated from it.

> **NOTE**
>
> If the PV does not have sufficient free extents for the entire LV, then the **lvcreate** fails.

- If you specify two PVs, extents for that LV will be allocated from one of them, or a combination of both.

- If you do not specify any PV, extents will be allocated from one of the PVs in the VG, or any combination of all PVs in the VG.

> **NOTE**
>
> In these cases, LVM might not use all of the named or available PVs. If the first PV has sufficient free extents for the entire LV, then the other PV will probably not be used. However, if the first PV does not have a set allocation size of free extents, then LV might be allocated partly from the first PV and partly from the second PV.

In this example, **lv1** extents will be allocated from **sda**:

```
# lvcreate -n lv1 -L1G vg /dev/sda
```

In this example, **lv2** extents will be allocated from either **sda**, or **sdb**, or a combination of both:

```
# lvcreate -n lv2 -L1G vg /dev/sda /dev/sdb
```

In this example, **lv3** extents will be allocated from one of the PVs in the VG, or any combination of all PVs in the VG:

```
# lvcreate -n lv3 -L1G vg
```

b. Allocate extents to create a raid volume:

```
# lvcreate --type <segment_type> -m <mirror_images> -n <lv_name> -L <lv_size> <vg_name> [ <PV> ... ]
```

Where:

- ** is the specified segment type (for example **raid5**, **mirror**, **snapshot**).

- *<mirror_images>* creates a **raid1** or a mirrored LV with the specified number of images. For example, **-m 1** would result in a **raid1** LV with two images.

- *<lv_name>* is the name of the LV.

- *<lv_size>* is the size of the LV. Default unit is megabytes.

- *<vg_name>* is the name of the VG.

- *<[PV ...]>* are the PVs.

The first raid image will be allocated from the first PV, the second raid image from the second PV, and so on.

In this example, **lv4** first raid image will be allocated from **sda** and second image will be allocated from **sdb**:

```
# lvcreate --type raid1 -m 1 -n lv4 -L1G vg /dev/sda /dev/sdb
```

In this example, **lv5** first raid image will be allocated from **sda**, second image will be allocated from **sdb**, and third image will be allocated from **sdc**:

```
# lvcreate --type raid1 -m 2 -n lv5 -L1G vg /dev/sda /dev/sdb /dev/sdc
```

12.2. LVM ALLOCATION POLICIES

LVM allocation policies control how physical extents are assigned to logical volumes using contiguous, cling, normal, and anywhere strategies.

When an LVM operation must allocate physical extents for one or more logical volumes (LVs), the allocation proceeds as follows:

- The complete set of unallocated physical extents in the volume group is generated for consideration. If you supply any ranges of physical extents at the end of the command line, only unallocated physical extents within those ranges on the specified physical volumes (PVs) are considered.

- Each allocation policy is tried in turn, starting with the strictest policy (**contiguous**) and ending with the allocation policy specified using the **--alloc** option or set as the default for the particular LV or volume group (VG). For each policy, working from the lowest-numbered logical extent of the empty LV space that needs to be filled, as much space as possible is allocated, according to the restrictions imposed by the allocation policy. If more space is needed, LVM moves on to the next policy.

The allocation policy restrictions are as follows:

- The **contiguous** policy requires that the physical location of any logical extent is adjacent to the physical location of the immediately preceding logical extent, with the exception of the first logical extent of a LV.
  When a LV is striped or mirrored, the **contiguous** allocation restriction is applied independently to each stripe or raid image that needs space.

- The **cling** allocation policy requires that the PV used for any logical extent be added to an existing LV that is already in use by at least one logical extent earlier in that LV.

- An allocation policy of **normal** will not choose a physical extent that shares the same PV as a logical extent already allocated to a parallel LV (that is, a different stripe or raid image) at the same offset within that parallel LV.

- If there are sufficient free extents to satisfy an allocation request but a **normal** allocation policy would not use them, the **anywhere** allocation policy will, even if that reduces performance by placing two stripes on the same PV.

You can change the allocation policy by using the **vgchange** command.

NOTE

Future updates can bring code changes in layout behavior according to the defined allocation policies. For example, if you supply on the command line two empty physical volumes that have an identical number of free physical extents available for allocation, LVM currently considers using each of them in the order they are listed; there is no guarantee that future releases will maintain that property. If you need a specific layout for a particular LV, build it up through a sequence of **lvcreate** and **lvconvert** steps such that the allocation policies applied to each step leave LVM no discretion over the layout.

12.3. PREVENTING ALLOCATION ON A PHYSICAL VOLUME

You can prevent allocation of physical extents on the free space of one or more physical volumes with the **pvchange** command. This might be necessary if there are disk errors, or if you will be removing the physical volume.

Procedure

- Use the following command to disallow the allocation of physical extents on *device_name*:

  ```
  # pvchange -x n /dev/sdk1
  ```

  You can also allow allocation where it had previously been disallowed by using the **-xy** arguments of the **pvchange** command.

CHAPTER 13. GROUPING LVM OBJECTS WITH TAGS

You can assign tags to Logical Volume Manager (LVM) objects to group them. This feature can help youautomate the control of LVM behavior, such as activation, by a group. You can also use tags instead of LVM objects arguments.

13.1. LVM OBJECT TAGS

A Logical Volume Manager (LVM) tag groups LVM objects of the same type. You can attach tags to objects such as physical volumes, volume groups, and logical volumes, and to hosts in a cluster configuration.

To avoid ambiguity, prefix each tag with **@**. Each tag is expanded by replacing it with all the objects that possess that tag and that are of the type expected by its position on the command line.

LVM tags are strings of up to 1024 characters. LVM tags cannot start with a hyphen.

A valid tag consists of a limited range of characters only. The allowed characters are **A-Z a-z 0-9 _ + . - / = ! : # &**.

Only objects in a volume group can be tagged. Physical volumes lose their tags if they are removed from a volume group; this is because tags are stored as part of the volume group metadata and that is deleted when a physical volume is removed.

You can apply some commands to all volume groups (VG), logical volumes (LV), or physical volumes (PV) that have the same tag. The man page of the given command shows the syntax, such as **VG|Tag**, **LV|Tag**, or **PV|Tag** when you can substitute a tag name for a VG, LV, or PV name.

13.2. ADDING TAGS TO LVM OBJECTS

You can add tags to LVM objects to group them by using the **--addtag** option with various volume management commands.

Prerequisites

- The **lvm2** package is installed.

Procedure

- To add a tag to an existing PV, use:

  ```
  # pvchange --addtag <@tag> <PV>
  ```

- To add a tag to an existing VG, use:

  ```
  # vgchange --addtag <@tag> <VG>
  ```

- To add a tag to a VG during creation, use:

  ```
  # vgcreate --addtag <@tag> <VG> ...
  ```

- To add a tag to an existing LV, use:

```
# lvchange --addtag <@tag> <LV>
```

- To add a tag to a LV during creation, use:

```
# lvcreate --addtag <@tag> ...
```

13.3. REMOVING TAGS FROM LVM OBJECTS

If you no longer want to keep your LVM objects grouped, you can remove tags from the objects by using the **--deltag** option with various volume management commands.

Prerequisites

- The **lvm2** package is installed.

- You have created tags on physical volumes (PV), volume groups (VG), or logical volumes (LV).

Procedure

- To remove a tag from an existing PV, use:

```
# pvchange --deltag @tag PV
```

- To remove a tag from an existing VG, use:

```
# vgchange --deltag @tag VG
```

- To remove a tag from an existing LV, use:

```
# lvchange --deltag @tag LV
```

13.4. DISPLAYING TAGS ON LVM OBJECTS

You can display tags on your LVM objects with the following commands.

Prerequisites

- The **lvm2** package is installed.

- You have created tags on physical volumes (PV), volume groups (VG), or logical volumes (LV).

Procedure

- To display all tags on an existing PV, use:

```
# pvs -o tags <PV>
```

- To display all tags on an existing VG, use:

```
# vgs -o tags <VG>
```

- To display all tags on an existing LV, use:

  # lvs -o tags *<LV>*

13.5. CONTROLLING LOGICAL VOLUME ACTIVATION WITH TAGS

Configuration file entries and volume group tags can selectively activate specific logical volumes on designated hosts in cluster environments by using metadata-based filtering.

You can specify in the configuration file that only certain logical volumes should be activated on that host.

For example, the following entry acts as a filter for activation requests (such as **vgchange -ay**) and only activates **vg1/lvol0** and any logical volumes or volume groups with the **database** tag in the metadata on that host:

  activation { volume_list = ["vg1/lvol0", "@database"] }

The special match **@*** that causes a match only if any metadata tag matches any host tag on that machine.

As another example, consider a situation where every machine in the cluster has the following entry in the configuration file:

  tags { hosttags = 1 }

If you want to activate **vg1/lvol2** only on host **db2**, do the following:

1. Run **lvchange --addtag @db2 vg1/lvol2** from any host in the cluster.

2. Run **lvchange -ay vg1/lvol2**.

This solution involves storing host names inside the volume group metadata.

CHAPTER 14. TROUBLESHOOTING LVM

You can use Logical Volume Manager (LVM) tools to troubleshoot a variety of issues in LVM volumes and groups.

14.1. GATHERING DIAGNOSTIC DATA ON LVM

If an LVM command is not working as expected, you can gather diagnostics in the following ways.

Procedure

- Use the following methods to gather different kinds of diagnostic data:

  - Add the **-v** argument to any LVM command to increase the verbosity level of the command output. Verbosity can be further increased by adding additional **v's**. A maximum of four such **v's** is allowed, for example, **-vvvv**.

  - In the **log** section of the **/etc/lvm/lvm.conf** configuration file, increase the value of the **level** option. This causes LVM to provide more details in the system log.

  - If the problem is related to the logical volume activation, enable LVM to log messages during the activation:

    i. Set the **activation = 1** option in the **log** section of the **/etc/lvm/lvm.conf** configuration file.

    ii. Execute the LVM command with the **-vvvv** option.

    iii. Examine the command output.

    iv. Reset the **activation** option to **0**.
        If you do not reset the option to **0**, the system might become unresponsive during low memory situations.

  - Display an information dump for diagnostic purposes:

    ```
    # lvmdump
    ```

  - Display additional system information:

    ```
    # lvs -v
    ```

    ```
    # pvs --all
    ```

    ```
    # dmsetup info --columns
    ```

  - Examine the last backup of the LVM metadata in the **/etc/lvm/backup/** directory and archived versions in the **/etc/lvm/archive/** directory.

  - Check the current configuration information:

    ```
    # lvmconfig
    ```

- Check the **/run/lvm/hints** cache file for a record of which devices have physical volumes on them.

14.2. DISPLAYING INFORMATION ABOUT FAILED LVM DEVICES

Troubleshooting information about a failed Logical Volume Manager (LVM) volume can help you determine the reason of the failure.

You can check the following examples of the most common LVM volume failures.

> **Example 14.1. Failed volume groups**
>
> In this example, one of the devices that made up the volume group *myvg* failed. The volume group usability then depends on the type of failure. For example, the volume group is still usable if RAID volumes are also involved. You can also see information about the failed device.
>
> ```
> # vgs --options +devices
>  /dev/vdb1: open failed: No such device or address
>  /dev/vdb1: open failed: No such device or address
>   WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
>   WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
> /dev/sdb1).
>   WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
> devices.
>
> VG    #PV #LV #SN Attr   VSize  VFree  Devices
> myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](0)
> myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](5120),/dev/vdb1(0)
> ```

> **Example 14.2. Failed logical volume**
>
> In this example, one of the devices failed. This can be a reason for the logical volume in the volume group to fail. The command output shows the failed logical volumes.
>
> ```
> # lvs --all --options +devices
>
>   /dev/vdb1: open failed: No such device or address
>   /dev/vdb1: open failed: No such device or address
>   WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
>   WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
> /dev/sdb1).
>   WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
> devices.
>
>   LV    VG  Attr      LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
>   mylv myvg -wi-a---p- 20.00g                                            [unknown](0)
> [unknown](5120),/dev/sdc1(0)
> ```

> **Example 14.3. Failed image of a RAID logical volume**
>
> The following examples show the command output from the **pvs** and **lvs** utilities when an image of a RAID logical volume has failed. The logical volume is still usable.

```
# pvs

  Error reading device /dev/sdc1 at 0 length 4.

  Error reading device /dev/sdc1 at 4096 length 4.

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and
assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and
assumed devices.

  PV          VG        Fmt  Attr PSize    PFree
  /dev/sda2   rhel_bp-01 lvm2 a--  <464.76g   4.00m
  /dev/sdb1   myvg       lvm2 a--  <836.69g 736.68g
  /dev/sdd1   myvg       lvm2 a--  <836.69g <836.69g
  /dev/sde1   myvg       lvm2 a--  <836.69g <836.69g
  [unknown]   myvg       lvm2 a-m  <836.69g 736.68g
```

```
# lvs -a --options name,vgname,attr,size,devices myvg

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and
assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and
assumed devices.

  LV                VG   Attr       LSize   Devices
  my_raid1          myvg rwi-a-r-p- 100.00g my_raid1_rimage_0(0),my_raid1_rimage_1(0)
  [my_raid1_rimage_0] myvg iwi-aor--- 100.00g /dev/sdb1(1)
  [my_raid1_rimage_1] myvg Iwi-aor-p- 100.00g [unknown](1)
  [my_raid1_rmeta_0]  myvg ewi-aor---   4.00m /dev/sdb1(0)
  [my_raid1_rmeta_1]  myvg ewi-aor-p-   4.00m [unknown](0)
```

14.3. REMOVING LOST LVM PHYSICAL VOLUMES FROM A VOLUME GROUP

If a physical volume fails, you can activate the remaining physical volumes in the volume group and remove all the logical volumes that used that physical volume from the volume group.

Procedure

1. Activate the remaining physical volumes in the volume group:

   ```
   # vgchange --activate y --partial myvg
   ```

2. Check which logical volumes will be removed:

> # vgreduce --removemissing --test *myvg*

3. Remove all the logical volumes that used the lost physical volume from the volume group:

> # vgreduce --removemissing --force *myvg*

4. Optional: If you accidentally removed logical volumes that you wanted to keep, you can reverse the **vgreduce** operation:

> # vgcfgrestore *myvg*

> ⚠ **WARNING**
>
> If you remove a thin pool, LVM cannot reverse the operation.

14.4. FINDING THE METADATA OF A MISSING LVM PHYSICAL VOLUME

If the volume group's metadata area of a physical volume is accidentally overwritten or otherwise destroyed, you get an error message indicating that the metadata area is incorrect, or that the system was unable to find a physical volume with a particular UUID.

This procedure finds the latest archived metadata of a physical volume that is missing or corrupted.

Procedure

1. Find the archived metadata file of the volume group that contains the physical volume. The archived metadata files are located at the **/etc/lvm/archive/*volume-group-name_backup-number*.vg** path:

> # cat /etc/lvm/archive/*myvg_00000-1248998876*.vg

Replace *00000-1248998876* with the backup-number. Select the last known valid metadata file, which has the highest number for the volume group.

2. Find the UUID of the physical volume. Use one of the following methods.

   - List the logical volumes:

     > # lvs --all --options +devices
     >
     > Couldn't find device with uuid '*FmGRh3-zhok-iVl8-7qTD-S5BI-MAEN-NYM5Sk*'.

   - Examine the archived metadata file. Find the UUID as the value labeled **id =** in the **physical_volumes** section of the volume group configuration.

   - Deactivate the volume group using the **--partial** option:

```
# vgchange --activate n --partial myvg

  PARTIAL MODE. Incomplete logical volumes will be processed.
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-
z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last
written to /dev/vdb1).
  0 logical volume(s) in volume group "myvg" now active
```

14.5. RESTORING METADATA ON AN LVM PHYSICAL VOLUME

This procedure restores metadata on a physical volume that is either corrupted or replaced with a new device. You might be able to recover the data from the physical volume by rewriting the metadata area on the physical volume.

> ⚠️ **WARNING**
>
> Do not attempt this procedure on a working LVM logical volume. You will lose your data if you specify the incorrect UUID.

Prerequisites

- You have identified the metadata of the missing physical volume. For details, see Finding the metadata of a missing LVM physical volume.

Procedure

1. Restore the metadata on the physical volume:

   ```
   # pvcreate --uuid physical-volume-uuid --restorefile /etc/lvm/archive/volume-group-
   name_backup-number.vg block-device
   ```

   > **NOTE**
   >
   > The command overwrites only the LVM metadata areas and does not affect the existing data areas.

   The following example labels the */dev/vdb1* device as a physical volume with the following properties:

   - The UUID of **FmGRh3-zhok-iVl8-7qTD-S5BI-MAEN-NYM5Sk**

   - The metadata information contained in **VG_00050.vg**, which is the most recent good archived metadata for the volume group

   ```
   # pvcreate --uuid "FmGRh3-zhok-iVl8-7qTD-S5BI-MAEN-NYM5Sk" --restorefile
   /etc/lvm/archive/VG_00050.vg /dev/vdb1
   ```

> ...
> Physical volume "/dev/vdb1" successfully created

2. Restore the metadata of the volume group:

   > # vgcfgrestore *myvg*
   >
   > Restored volume group *myvg*

3. Display the logical volumes on the volume group:

   > # lvs --all --options +devices *myvg*

   The logical volumes are currently inactive. For example:

   > LV   VG   Attr   LSize   Origin Snap% Move Log Copy% Devices
   > *mylv myvg*   -wi--- 300.00G   /dev/vdb1 (0),/dev/vdb1(0)
   > *mylv myvg*   -wi--- 300.00G   /dev/vdb1 (34728),/dev/vdb1(0)

4. If the segment type of the logical volumes is RAID, resynchronize the logical volumes:

   > # lvchange --resync *myvg/mylv*

5. Activate the logical volumes:

   > # lvchange --activate y *myvg/mylv*

6. If the on-disk LVM metadata takes at least as much space as what overrode it, this procedure can recover the physical volume. If what overrode the metadata went past the metadata area, the data on the volume may have been affected. You might be able to use the **fsck** command to recover that data.

Verification

- Display the active logical volumes:

  > # lvs --all --options +devices
  >
  > LV   VG   Attr   LSize   Origin Snap% Move Log Copy% Devices
  > *mylv myvg*   -wi--- 300.00G   /dev/vdb1 (0),/dev/vdb1(0)
  > *mylv myvg*   -wi--- 300.00G   /dev/vdb1 (34728),/dev/vdb1(0)

14.6. ROUNDING ERRORS IN LVM OUTPUT

LVM commands that report the space usage in volume groups round the reported number to **2** decimal places to provide human-readable output. This includes the **vgdisplay** and **vgs** utilities.

As a result of the rounding, the reported value of free space might be larger than what the physical extents on the volume group provide. If you attempt to create a logical volume the size of the reported free space, you might get the following error:

> Insufficient free extents

To work around the error, you must examine the number of free physical extents on the volume group, which is the accurate value of free space. You can then use the number of extents to create the logical volume successfully.

14.7. PREVENTING THE ROUNDING ERROR WHEN CREATING AN LVM VOLUME

When creating an LVM logical volume, you can specify the number of logical extents of the logical volume to avoid rounding error.

Procedure

1. Find the number of free physical extents in the volume group:

   ```
   # vgdisplay myvg
   ```

   For example, the following volume group has 8780 free physical extents:

   ```
   --- Volume group ---
    VG Name               myvg
    System ID
    Format                lvm2
    Metadata Areas        4
    Metadata Sequence No  6
    VG Access             read/write
    [...]
    Free  PE / Size       8780 / 34.30 GB
   ```

2. Create the logical volume. Enter the volume size in extents rather than bytes.
   For example, to create a logical volume by specifying the number of extents, run the following command:

   ```
   # lvcreate --extents 8780 --name mylv myvg
   ```

   Alternatively, you can extend the logical volume to use a percentage of the remaining free space in the volume group. For example:

   ```
   # lvcreate --extents 100%FREE --name mylv myvg
   ```

Verification

- Check the number of extents that the volume group now uses:

  ```
  # vgs --options +vg_free_count,vg_extent_count

    VG    #PV #LV #SN Attr   VSize  VFree  Free  #Ext
    myvg  2   1   0   wz--n- 34.30G 0      0     8780
  ```

14.8. LVM METADATA AND THEIR LOCATION ON DISK

LVM headers and metadata areas are available in different offsets and sizes.

The default LVM disk header:

- Is found in **label_header** and **pv_header** structures.

- Is in the second 512-byte sector of the disk. Note that if a non-default location was specified when creating the physical volume (PV), the header can also be in the first or third sector.

The standard LVM metadata area:

- Begins 4096 bytes from the start of the disk.

- Ends 1 MiB from the start of the disk.

- Begins with a 512 byte sector containing the **mda_header** structure.

A metadata text area begins after the **mda_header** sector and goes to the end of the metadata area. LVM VG metadata text is written in a circular fashion into the metadata text area. The **mda_header** points to the location of the latest VG metadata within the text area.

You can print LVM headers from a disk by using the **# pvck --dump headers /dev/sda** command. This command prints **label_header**, **pv_header**, **mda_header**, and the location of metadata text if found. Bad fields are printed with the **CHECK** prefix.

The LVM metadata area offset will match the page size of the machine that created the PV, so the metadata area can also begin 8K, 16K or 64K from the start of the disk.

Larger or smaller metadata areas can be specified when creating the PV, in which case the metadata area may end at locations other than 1 MiB. The **pv_header** specifies the size of the metadata area.

When creating a PV, a second metadata area can be optionally enabled at the end of the disk. The **pv_header** contains the locations of the metadata areas.

14.9. EXTRACTING VG METADATA FROM A DISK

Choose one of the following procedures to extract VG metadata from a disk, depending on your situation.

> **NOTE**
>
> For repair, you can use backup files in **/etc/lvm/backup/** without extracting metadata from disk.

Procedure

- Print current metadata text as referenced from valid **mda_header**:

  ```
  # pvck --dump metadata <disk>
  ```

  For example, to print metadata text from valid **mda_header**:

  ```
  # pvck --dump metadata /dev/sdb
    metadata text at 172032 crc Oxc627522f # vgname test segno 59
    ---
  ```

> *<raw metadata from disk>*
> ---

- Print the locations of all metadata copies found in the metadata area, based on finding a valid **mda_header**:

  > # pvck --dump metadata_all *<disk>*

  For example, to print the locations of metadata copies on the /**dev**/**sdb** disk:

  > # pvck --dump metadata_all /dev/sdb
  >   metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  >   metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  >   metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv

- Search for all copies of metadata in the metadata area without using an **mda_header**, for example, if headers are missing or damaged:

  > # pvck --dump metadata_search *<disk>*

  For example, to search for copies of metadata in the metadata area on the /**dev**/**sdb** disk without using an **mda_header**:

  > # pvck --dump metadata_search /dev/sdb
  >   Searching for metadata at offset 4096 size 1044480
  >   metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  >   metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  >   metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv

- Include the **-v** option in the **dump** command to show the description from each copy of metadata:

  > # pvck --dump metadata -v *<disk>*

  For example, to show description from each copy of metadata on the /**dev**/**sdb** disk:

  > # pvck --dump metadata -v /dev/sdb
  >   metadata text at 199680 crc 0x628cf243 # vgname my_vg seqno 40
  >   ---
  > my_vg {
  > id = "dmEbPi-gsgx-VbvS-Uaia-HczM-iu32-Rb7iOf"
  > seqno = 40
  > format = "lvm2"
  > status = ["RESIZEABLE", "READ", "WRITE"]
  > flags = []
  > extent_size = 8192
  > max_lv = 0

```
max_pv = 0
metadata_copies = 0

physical_volumes {

pv0 {
id = "8gn0is-Hj8p-njgs-NM19-wuL9-mcB3-kUDiOQ"
device = "/dev/sda"

device_id_type = "sys_wwid"
device_id = "naa.6001405e635dbaab125476d88030a196"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}

pv1 {
id = "E9qChJ-5ElL-HVEp-rc7d-U5Fg-fHxL-2QLyID"
device = "/dev/sdb"

device_id_type = "sys_wwid"
device_id = "naa.6001405f3f9396fddcd4012a50029a90"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}
```

This file can be used for repair. The first metadata area is used by default for dump metadata. If the disk has a second metadata area at the end of the disk, you can use the **--settings "mda_num=2"** option to use the second metadata area for dump metadata instead.

Additional resources

- Saving extracted metadata to a file

14.10. SAVING EXTRACTED METADATA TO A FILE

If you need to use dumped metadata for repair, it is required to save extracted metadata to a file with the **-f** option and the **--setings** option.

Procedure

- If **-f <filename>** is added to **--dump metadata**, the raw metadata is written to the named file. You can use this file for repair.

- If **-f <filename>** is added to **--dump metadata_all** or **--dump metadata_search**, then raw metadata from all locations is written to the named file.

- To save one instance of metadata text from **--dump metadata_all|metadata_search** add **--settings "metadata_offset=<offset>"** where **<offset>** is from the listing output "metadata at <offset>":

```
# pvck --dump metadata_search --settings metadata_offset=5632 -f meta.txt /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-
zb6G-BYat-u53Fxv
# head -2 meta.txt
test {
id = "FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv"
```

14.11. REPAIRING A DISK WITH DAMAGED LVM HEADERS AND METADATA USING THE PVCREATE AND THE VGCFGRESTORE COMMANDS

You can restore metadata and headers on a physical volume that is either corrupted or replaced with a new device. You might be able to recover the data from the physical volume by rewriting the metadata area on the physical volume.

> **WARNING**
>
> These instructions should be used with extreme caution, and only if you are familiar with the implications of each command, the current layout of the volumes, the layout that you need to achieve, and the contents of the backup metadata file. These commands have the potential to corrupt data, and as such, it is recommended that you contact Red Hat Global Support Services for assistance in troubleshooting.

Prerequisites

- You have identified the metadata of the missing physical volume. For details, see Finding the metadata of a missing LVM physical volume.

Procedure

1. Collect the following information needed for the **pvcreate** and **vgcfgrestore** commands. You can collect the information about your disk and UUID by running the **# pvs -o+uuid** command.

   - **metadata-file** is the path to the most recent metadata backup file for the VG, for example, **/etc/lvm/backup/<vg-name>**.

   - **vg-name** is the name of the VG that has the damaged or missing PV.

   - **UUID** of the PV that was damaged on this device is the value taken from the output of the **# pvs -i+uuid** command.

   - **disk** is the name of the disk where the PV is supposed to be, for example, **/dev/sdb**. Be certain this is the correct disk, or seek help, otherwise following these steps may lead to data loss.

2. Recreate LVM headers on the disk:

> # pvcreate --restorefile *<metadata-file>* --uuid *<UUID>* *<disk>*

Optionally, verify that the headers are valid:

> # pvck --dump headers *<disk>*

3. Restore the VG metadata on the disk:

> # vgcfgrestore --file *<metadata-file>* *<vg-name>*

Optionally, verify the metadata is restored:

> # pvck --dump metadata *<disk>*

If there is no metadata backup file for the VG, you can get one by saving extracted metadata to a file.

Verification

- To verify that the new physical volume is intact and the volume group is functioning correctly, check the output of the following command:

> # vgs

Additional resources

- Saving extracted metadata to a file

- Extracting LVM metadata backups from a physical volume (Red Hat Knowledgebase)

- How to repair metadata on physical volume online? (Red Hat Knowledgebase)

- How do I restore a volume group in Red Hat Enterprise Linux if one of the physical volumes that constitute the volume group has failed? (Red Hat Knowledgebase)

14.12. REPAIRING A DISK WITH DAMAGED LVM HEADERS AND METADATA USING THE PVCK COMMAND

This is an alternative to the Repairing a disk with damaged LVM headers and metadata using the pvcreate and the vgcfgrestore commands. There may be cases where the **pvcreate** and the **vgcfgrestore** commands do not work. This method is more targeted at the damaged disk.

This method uses a metadata input file that was extracted by **pvck --dump**, or a backup file from **/etc/lvm/backup**. When possible, use metadata saved by **pvck --dump** from another PV in the same VG, or from a second metadata area on the PV. For more information, see Saving extracted metadata to a file.

Procedure

- Repair the headers and metadata on the disk:

> # pvck --repair -f *<metadata-file>* *<disk>*

where

- *<metadata-file>* is a file containing the most recent metadata for the VG. This can be **/etc/lvm/backup/vg-name**, or it can be a file containing raw metadata text from the **pvck --dump metadata_search** command output.

- *<disk>* is the name of the disk where the PV is supposed to be, for example, **/dev/sdb**. To prevent data loss, verify that is the correct disk. If you are not certain the disk is correct, contact Red Hat Support.

> **NOTE**
>
> If the metadata file is a backup file, the **pvck --repair** should be run on each PV that holds metadata in VG. If the metadata file is raw metadata that has been extracted from another PV, the **pvck --repair** needs to be run only on the damaged PV.

Verification

- To check that the new physical volume is intact and the volume group is functioning correctly, check outputs of the following commands:

  ```
  # vgs <vgname>
  ```

  ```
  # pvs <pvname>
  ```

  ```
  # lvs <lvname>
  ```

Additional resources

- Extracting LVM metadata backups from a physical volume (Red Hat Knowledgebase)

- How to repair metadata on physical volume online? (Red Hat Knowledgebase)

- How do I restore a volume group in Red Hat Enterprise Linux if one of the physical volumes that constitute the volume group has failed? (Red Hat Knowledgebase)

14.13. TROUBLESHOOTING DUPLICATE PHYSICAL VOLUME WARNINGS FOR MULTIPATHED LVM DEVICES

When using LVM with multipathed storage, LVM commands that list a volume group or logical volume might display warning messages. You can troubleshoot these warnings to understand why LVM displays them, or to hide the warnings.

Following are some warning messages that you might see:

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/dm-5 not /dev/sdd
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowerb not /dev/sde
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sddlmab not /dev/sdf
```

You can troubleshoot these warnings to understand why LVM displays them, or to hide the warnings.

14.13.1. Root cause of duplicate PV warnings

When a multipath software such as Device Mapper Multipath (DM Multipath), EMC PowerPath, or Hitachi Dynamic Link Manager (HDLM) manages storage devices on the system, each path to a particular logical unit (LUN) is registered as a different SCSI device.

The multipath software then creates a new device that maps to those individual paths. Because each LUN has multiple device nodes in the **/dev** directory that point to the same underlying data, all the device nodes contain the same LVM metadata.

Table 14.1. Example device mappings in different multipath software

Multipath software	SCSI paths to a LUN	Multipath device mapping to paths
DM Multipath	**/dev**/**sdb** and **/dev**/**sdc**	**/dev**/**mapper**/**mpath1** or **/dev**/**mapper**/**mpatha**
EMC PowerPath		**/dev**/**emcpowera**
HDLM		**/dev**/**sddlmab**

As a result of the multiple device nodes, LVM tools find the same metadata multiple times and report them as duplicates.

14.13.2. Cases of duplicate PV warnings

Logical Volume Manager (LVM) displays duplicate PV warnings in two scenarios.

Single paths to the same device

The two devices displayed in the output are both single paths to the same device.
The following example shows a duplicate PV warning in which the duplicate devices are both single paths to the same device.

> Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sdd not /dev/sdf

If you list the current DM Multipath topology using the **multipath -ll** command, you can find both **/dev**/**sdd** and **/dev**/**sdf** under the same multipath map.

These duplicate messages are only warnings and do not mean that the LVM operation has failed. Rather, they are alerting you that LVM uses only one of the devices as a physical volume and ignores the others.

If the messages indicate that LVM chooses the incorrect device or if the warnings are disruptive to users, you can apply a filter. The filter configures LVM to search only the necessary devices for physical volumes, and to leave out any underlying paths to multipath devices. As a result, the warnings no longer appear.

Multipath maps

The two devices displayed in the output are both multipath maps.

The following examples show a duplicate PV warning for two devices that are both multipath maps. The duplicate physical volumes are located on two different devices rather than on two different paths to the same device.

> Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/mapper/mpatha not /dev/mapper/mpathc
>
> Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowera not /dev/emcpowerh

This situation is more serious than duplicate warnings for devices that are both single paths to the same device. These warnings often mean that the machine is accessing devices that it should not access: for example, LUN clones or mirrors.

Unless you clearly know which devices you should remove from the machine, this situation might be unrecoverable. Red Hat recommends that you contact Red Hat Technical Support to address this issue.

14.13.3. Example LVM device filters that prevent duplicate PV warnings

LVM device filters prevent duplicate physical volume warnings by using specific patterns to avoid multiple storage paths to the same logical unit.

The following examples show LVM device filters that avoid the duplicate physical volume warnings that are caused by multiple storage paths to a single logical unit (LUN).

You can configure the filter for logical volume manager (LVM) to check metadata for all devices. Metadata includes local hard disk drive with the root volume group on it and any multipath devices. By rejecting the underlying paths to a multipath device (such as **/dev/sdb**, **/dev/sdd**), you can avoid these duplicate PV warnings, because LVM finds each unique metadata area once on the multipath device itself.

- To accept the second partition on the first hard disk drive and any device mapper (DM) Multipath devices and reject everything else, enter:

  > filter = ["a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|"]

- To accept all HP SmartArray controllers and any EMC PowerPath devices, enter:

  > filter = ["a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|"]

- To accept any partitions on the first IDE drive and any multipath devices, enter:

  > filter = ["a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|"]

Additional resources

- Examples of LVM device filter configurations

Additional resources

- Limiting LVM device visibility and usage

- The LVM device filter