Red Hat Enterprise Linux 8

Configuring and managing logical volumes

Configuring and managing the LVM on RHEL

Last Updated: 2024-04-02

Red Hat Enterprise Linux 8 Configuring and managing logical volumes

Configuring and managing the LVM on RHEL

Legal Notice

Abstract

Logical volume management (LVM) creates a layer of abstraction over physical storage to create a logical storage volume, which is a virtual block storage device that a file system, database, or application can use. The physical volume (PV) is either a partition or a whole disk. By using these PVs, you can create a volume group (VG) to create a pool of disk space for the logical volumes (LV) from the available storage. You can create a logical volume (LV) by combining physical volumes into a volume group. LV provides more flexibility than using physical storage, and the created LVs can be extended or reduced without repartitioning or reformatting the physical device. You can also

perform several advanced operations with the LVM, such as creating thin-provisioned logical volumes, snapshots of the original volume, RAID volumes, cache volumes, and striped logical volumes.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your 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 management (LVM) creates a layer of abstraction over physical storage, which helps you to create logical storage volumes. This provides much greater flexibility in a number of ways than using physical storage directly.

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

1.1. LVM ARCHITECTURE

The following are the components of LVM:

Physical volume

A physical volume (PV) is a partition or whole disk designated for LVM use. For more information, see Managing LVM physical volumes.

Volume group

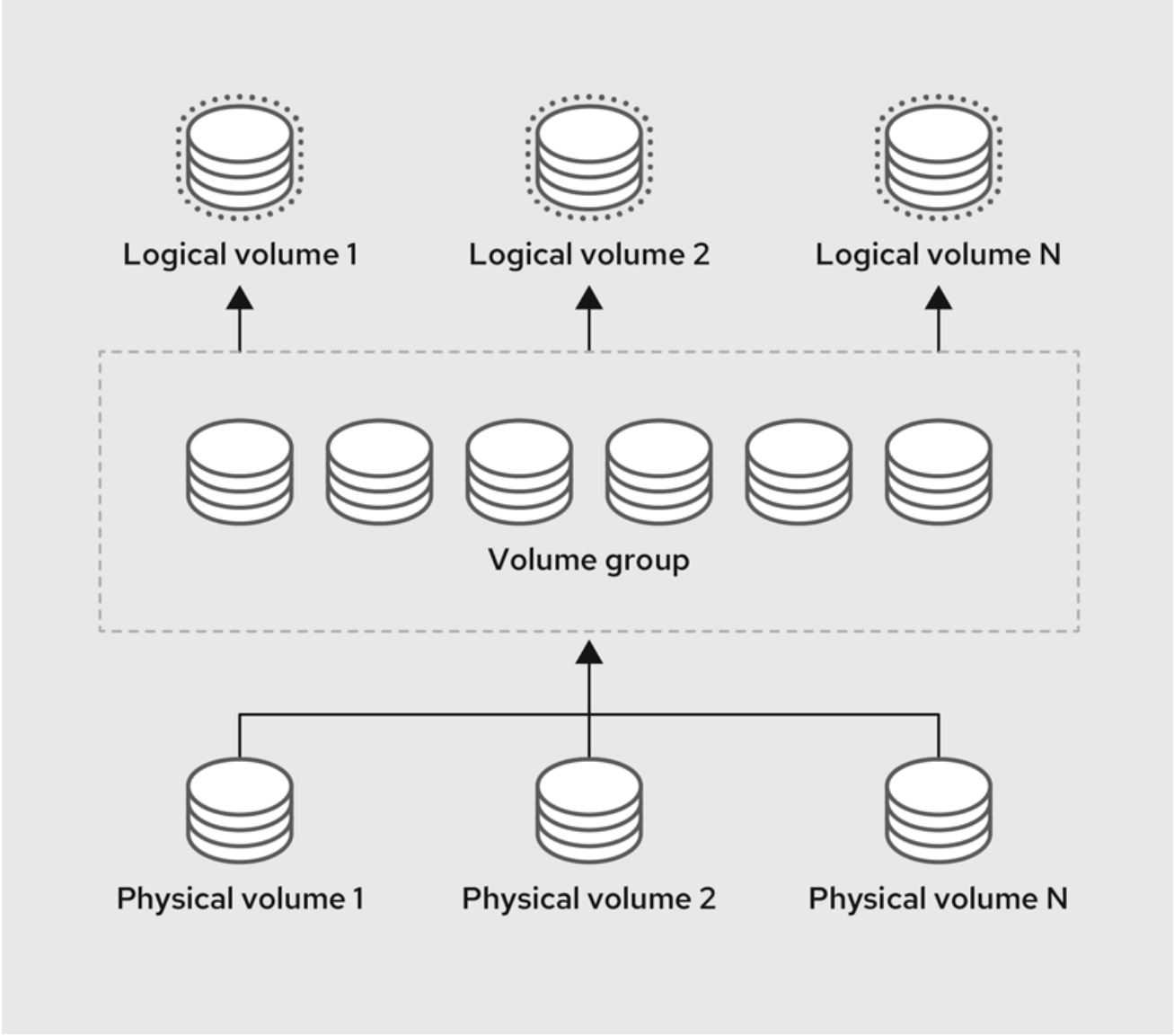
A volume group (VG) is a collection of physical volumes (PVs), which creates a pool of disk space out of which logical volumes can be allocated. For more information, see Managing LVM volume groups.

Logical volume

A logical volume represents a mountable storage device. For more information, see Managing LVM logical volumes.

The following diagram illustrates the components of LVM:

Figure 1.1. LVM logical volume components



1.2. ADVANTAGES OF LVM

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. For more information, see Modifying the size of a logical volume.

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** man page and Removing physical volumes from a volume group.

Striped Volumes

You can create a logical volume that stripes data across two or more devices. This can dramatically increase throughput. For more information, see Extending a striped logical volume.

RAID volumes

Logical volumes provide a convenient way to configure RAID for your data. This provides protection against device failure and improves performance. For more information, see Configuring RAID logical volumes.

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. For more information, see Snapshot of logical volumes.

Thin volumes

Logical volumes can be thinly provisioned. This allows you to create logical volumes that are larger than the available physical space. For more information, see Creating and managing thin provisioned volumes (thin volumes).

Cache volumes

A cache logical volume uses a fast block device, such as an SSD drive to improve the performance of a larger and slower block device. For more information, see Enabling caching to improve logical volume performance.

Additional resources

- Customizing the LVM report

CHAPTER 2. MANAGING LOCAL STORAGE BY USING RHEL SYSTEM ROLES

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

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

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

2.1. INTRODUCTION TO THE STORAGE RHEL SYSTEM ROLE

The **storage** role can manage:

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

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

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

Additional resources

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

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

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

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

storage_volumes

List of file systems on all unpartitioned disks to be managed.
storage_volumes can also include **raid** volumes.

Partitions are currently unsupported.

storage_pools

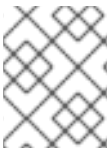
List of pools to be managed.
Currently the only supported pool type is LVM. With LVM, pools represent volume groups (VGs).
Under each pool there is a list of volumes to be managed by the role. With LVM, each volume corresponds to a logical volume (LV) with a file system.

Additional resources

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

2.3. CREATING AN XFS FILE SYSTEM ON A BLOCK DEVICE BY USING THE STORAGE RHEL SYSTEM ROLE

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



NOTE

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

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 on them.

Procedure

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

```
---
- hosts: managed-node-01.example.com
```

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

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

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
```

Additional resources

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

2.4. PERSISTENTLY MOUNTING A FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

The example Ansible applies the **storage** role to immediately and persistently mount an XFS file system.

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

2.5. MANAGING LOGICAL VOLUMES BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

2.6. ENABLING ONLINE BLOCK DISCARD BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

2. 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
```

Additional resources

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

2.7. CREATING AND MOUNTING AN EXT4 FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

2.8. CREATING AND MOUNTING AN EXT3 FILE SYSTEM BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

2.9. RESIZING AN EXISTING FILE SYSTEM ON LVM BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

   This playbook resizes the following existing file systems:

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

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
```

Additional resources

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

2.10. CREATING A SWAP VOLUME BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

2.11. CONFIGURING A RAID VOLUME BY USING THE STORAGE SYSTEM ROLE

With the **storage** System Role, you can configure a RAID volume on RHEL by using Red Hat Ansible Automation Platform and Ansible-Core. Create an Ansible playbook with the parameters to configure a RAID volume to suit your requirements.

WARNING

Device names might change in certain circumstances, for example, when you add a new disk to a system. Therefore, to prevent data loss, do not use specific disk names in the playbook.

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 on them.

Procedure

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

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

2. 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
```

Additional resources

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

2.12. CONFIGURING AN LVM POOL WITH RAID BY USING THE STORAGE RHEL SYSTEM ROLE

With the **storage** System Role, you can configure an LVM pool with RAID on RHEL by using Red Hat Ansible Automation Platform. You can set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

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

With the **storage** System Role, you can configure a stripe size for RAID LVM volumes on RHEL by using Red Hat Ansible Automation Platform. You can set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

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 on them.

Procedure

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

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

```
        raid_level: raid1
        raid_stripe_size: "256 KiB"
        state: present
```

2. 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:

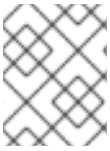
```
$ ansible-playbook ~/playbook.yml
```

Additional resources

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

2.14. COMPRESSING AND DEDUPLICATING A VDO VOLUME ON LVM BY USING THE STORAGE RHEL SYSTEM ROLE

The example Ansible playbook applies the **storage** RHEL System Role to enable compression and deduplication of Logical Volumes (LVM) by using Virtual Data Optimizer (VDO).



NOTE

Because of the **storage** System Role use of LVM VDO, only one volume per pool can use the compression and deduplication.

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 on them.

Procedure

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

```
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
```

```
    disks:
      - /dev/sdb
    volumes:
      - name: mylv1
        compression: true
        deduplication: true
        vdo_pool_size: 10 GiB
        size: 30 GiB
        mount_point: /mnt/app/shared
```

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

- The **deduplication** is used to deduplicate the duplicated data stored on the storage volume.
- The compression is used to compress the data stored on the storage volume, which results in more storage capacity.
- The vdo_pool_size specifies the actual size the volume takes on the device. The virtual size of VDO volume is set by the **size** parameter.

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
```

Additional resources

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

2.15. CREATING A LUKS2 ENCRYPTED VOLUME BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

You can also add other encryption parameters, such as **encryption_key**, **encryption_cipher**, **encryption_key_size**, and **encryption_luks**, to the playbook file.

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. View the encryption status:

```
# cryptsetup status sdb

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

2. Verify the created LUKS encrypted volume:

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
```

```
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards

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

Additional resources

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

2.16. EXPRESSING POOL VOLUME SIZES AS PERCENTAGE BY USING THE STORAGE RHEL SYSTEM ROLE

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

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 on them.

Procedure

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

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

```
    - name: web
      size: 30%
      mount_point: /opt/mount/web
    - name: cache
      size: 10%
      mount_point: /opt/cache/mount
```

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

2. 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
```

Additional resources

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

CHAPTER 3. MANAGING LVM PHYSICAL VOLUMES

The physical volume (PV) is a partition or whole disk designated for LVM use. To use the device for an LVM logical volume, the device must be initialized as a physical volume.

If you are using a whole disk device for your physical volume, the disk must have no partition table. For DOS disk partitions, the partition id should be set to 0x8e using the **fdisk** or **cfdisk** command or an equivalent. If you are using a whole disk device for your physical volume, the disk must have no partition table. Any existing partition table must be erased, which will effectively destroy all data on that disk. You can remove an existing partition table using the **wipefs -a <PhysicalVolume>`** command as root.

3.1. OVERVIEW OF PHYSICAL VOLUMES

Initializing a block device as a physical volume places a label near the start of the device. The following describes the LVM label:

- An LVM label provides correct identification and device ordering for a physical device. An unlabeled, non-LVM device can change names across reboots depending on the order they are discovered by the system during boot. An LVM label remains persistent across reboots and throughout a cluster.
- The LVM label identifies the device as an LVM physical volume. It contains a random unique identifier, the UUID for the physical volume. It also stores the size of the block device in bytes, and it records where the LVM metadata will be stored on the device.
- By default, the LVM label is placed in the second 512-byte sector. You can overwrite this default setting by placing the label on any of the first 4 sectors when you create the physical volume. This allows LVM volumes to co-exist with other users of these sectors, if necessary.

The following describes the LVM metadata:

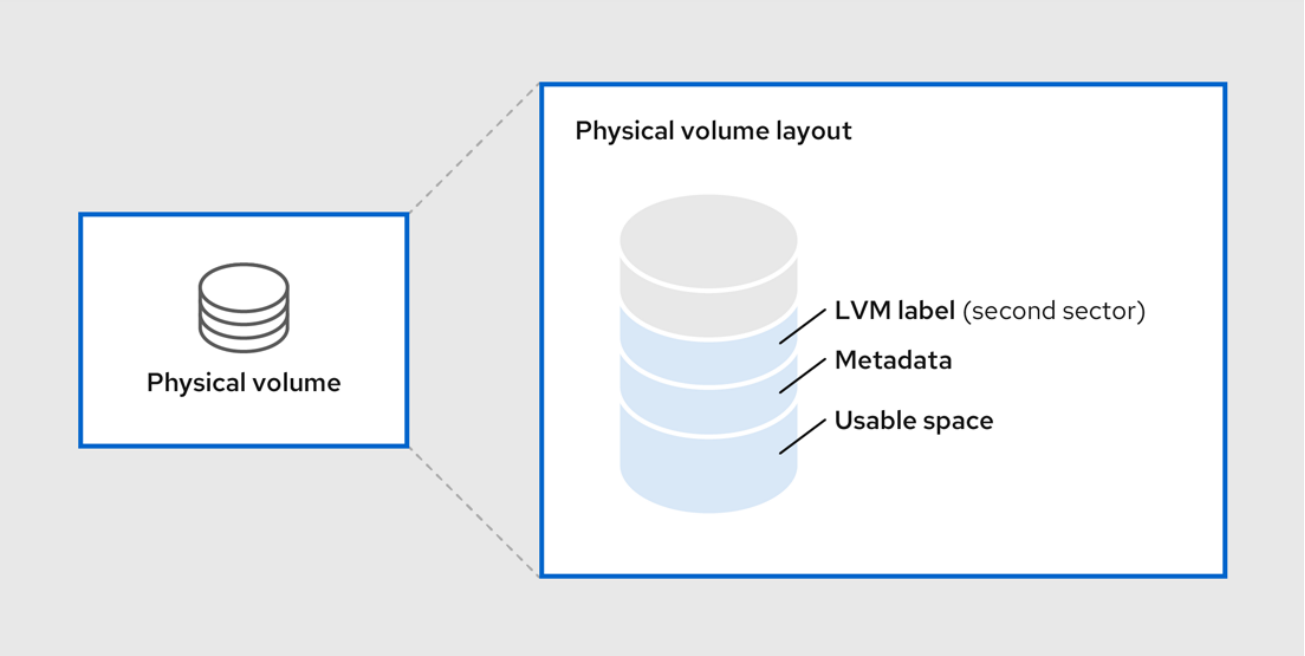
- The LVM metadata contains the configuration details of the LVM volume groups on your system. By default, an identical copy of the metadata is maintained in every metadata area in every physical volume within the volume group. LVM metadata is small and stored as ASCII.
- Currently LVM allows you to store 0, 1, or 2 identical copies of its metadata on each physical volume. The default is 1 copy. Once you configure the number of metadata copies on the physical volume, you cannot change that number at a later time. The first copy is stored at the start of the device, shortly after the label. If there is a second copy, it is placed at the end of the device. If you accidentally overwrite the area at the beginning of your disk by writing to a different disk than you intend, a second copy of the metadata at the end of the device will allow you to recover the metadata.

The following diagram illustrates the layout of an LVM physical volume. The LVM label is on the second sector, followed by the metadata area, followed by the usable space on the device.

NOTE

In the Linux kernel and throughout this document, sectors are considered to be 512 bytes in size.

Figure 3.1. Physical volume layout



Additional resources

- Multiple partitions on a disk

3.2. MULTIPLE PARTITIONS ON A DISK

You can create physical volumes (PV) out of disk partitions by using LVM.

Red Hat recommends that you create a single partition that covers the whole disk to label as an LVM physical volume for the following reasons:

Administrative convenience

It is easier to keep track of the hardware in a system if each real disk only appears once. This becomes particularly true if a disk fails.

Striping performance

LVM cannot tell that two physical volumes are on the same physical disk. If you create a striped logical volume when two physical volumes are on the same physical disk, the stripes could be on different partitions on the same disk. This would result in a decrease in performance rather than an increase.

RAID redundancy

LVM cannot determine that the two physical volumes are on the same device. If you create a RAID logical volume when two physical volumes are on the same device, performance and fault tolerance could be lost.

Although it is not recommended, there may be specific circumstances when you will need to divide a disk into separate LVM physical volumes. For example, on a system with few disks it may be necessary to move data around partitions when you are migrating an existing system to LVM volumes. Additionally, if you have a very large disk and want to have more than one volume group for administrative purposes then it is necessary to partition the disk. If you do have a disk with more than one partition and both of those partitions are in the same volume group, take care to specify which partitions are to be included in a logical volume when creating volumes.

Note that although LVM supports using a non-partitioned disk as physical volume, it is recommended to

create a single, whole-disk partition because creating a PV without a partition can be problematic in a mixed operating system environment. Other operating systems may interpret the device as free, and overwrite the PV label at the beginning of the drive.

3.3. CREATING LVM PHYSICAL VOLUME

This procedure describes how to create and label LVM physical volumes (PVs).

In this procedure, replace the */dev/vdb1*, */dev/vdb2*, and */dev/vdb3* with the available storage devices in your system.

Prerequisites

- The **lvm2** package is installed.

Procedure

1. Create multiple physical volumes by using the space-delimited device names as arguments to the **pvcreate** command:

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

   This places a label on */dev/vdb1*, */dev/vdb2*, and */dev/vdb3*, marking them as physical volumes belonging to LVM.

2. View the created physical volumes by using any one of the following commands as per your requirement:

   a. The **pvdisplay** command, which provides a verbose multi-line output for each physical volume. It displays physical properties, such as size, extents, volume group, and other options in a fixed format:

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

   b. The **pvs** command provides physical volume information in a configurable form, displaying one line per physical volume:

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

c. The **pvscan** command scans all supported LVM block devices in the system for physical volumes. You can define a filter in the **lvm.conf** file so that this command avoids scanning specific physical volumes:

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

Additional resources

- **pvcreate(8)**, **pvdisplay(8)**, **pvs(8)**, **pvscan(8)**, and **lvm(8)** man pages

3.4. REMOVING LVM PHYSICAL VOLUMES

If a device is no longer required for use by LVM, you can remove the LVM label by using the **pvremove** command. Executing the **pvremove** command zeroes the LVM metadata on an empty physical volume.

Procedure

1. Remove a physical volume:

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

2. View the existing physical volumes and verify if the required volume is removed:

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

If the physical volume you want to remove is currently part of a volume group, you must remove it from the volume group with the **vgreduce** command. For more information, see Removing physical volumes from a volume group

Additional resources

- **pvremove(8)** man page

3.5. ADDITIONAL RESOURCES

- Creating a partition table on a disk with parted .
- **parted(8)** man page.

CHAPTER 4. MANAGING LVM VOLUME GROUPS

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

Within a volume group, the disk space available for allocation is divided into units of a fixed-size called extents. An extent is the smallest unit of space that can be allocated. Within a physical volume, extents are referred to as physical extents.

A logical volume is allocated into logical extents of the same size as the physical extents. The extent size is therefore the same for all logical volumes in the volume group. The volume group maps the logical extents to physical extents.

4.1. CREATING LVM VOLUME GROUP

You can create an LVM volume group (VG) myvg using the */dev/vdb1* and */dev/vdb2* physical volumes (PVs). By default, when physical volumes are used to create a volume group, its disk space is divided into 4MB extents. This extent size is the minimum amount by which the logical volume can be increased or decreased in size. The extent size can be modified using the **-s** argument of the **vgcreate** command and large numbers of extents have no impact on I/O performance of the logical volume. You can put limits on the number of physical or logical volumes the volume group can have using the **-p** and **-l** arguments of the **vgcreate** command.

Prerequisites

- The **lvm2** package is installed.
- One or more physical volumes are created. For more information about creating physical volumes, see Creating LVM physical volume.

Procedure

1. Create a *myvg* VG using any of the following methods:
   - Without specifying any options:

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

   - By specifying the volume group extent size using the **-s** argument:

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

   - By limiting the number of physical or logical volumes the VG can have using the **-p** and **-l** arguments:

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

2. View the created volume groups by using any one of the following commands according to your requirement:

- The **vgs** command provides volume group information in a configurable form, displaying one line per volume group:

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

- The **vgdisplay** command displays volume group properties such as size, extents, number of physical volumes, and other options in a fixed form. The following example shows the output of the **vgdisplay** command for the volume group *myvg*. To display all existing volume groups, do not specify a volume group:

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

- The **vgscan** command scans all supported LVM block devices in the system for volume group:

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

3. Optional: Increase a volume group's capacity by adding one or more free physical volumes:

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

4. Optional: Rename an existing volume group:

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

Additional resources

- **vgcreate(8)**, **vgextend(8)**, **vgdisplay(8)**, **vgs(8)**, **vgscan(8)**, **vgrename(8)**, and **lvm(8)** man pages

4.2. COMBINING LVM VOLUME GROUPS

To combine two volume groups into a single volume group, use the **vgmerge** command. You can merge an inactive "source" volume with an active or an inactive "destination" volume if the physical extent sizes of the volume are equal and the physical and logical volume summaries of both volume groups fit into the destination volume groups limits.

Procedure

- Merge the inactive volume group *databases* into the active or inactive volume group *myvg* giving verbose runtime information:

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

Additional resources

- **vgmerge(8)** man page

4.3. 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 *myvg* volume group:

```
# vgextend myvg /dev/vdb4
  Volume group "myvg" 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 myvg /dev/vdb3
Removed "/dev/vdb3" from volume group "myvg"
```

Verification

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

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

Additional resources

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

4.4. SPLITTING A 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 *myvg* consists of */dev/vdb1*, */dev/vdb2*, and */dev/vdb3*. After completing this procedure, the volume group *myvg* will consist of */dev/vdb1* and */dev/vdb2*, and the second volume group, *yourvg*, 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 *myvg* to the new volume group *yourvg*:

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

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/myvg/mylv
```

For more information about creating logical volumes, see Managing LVM logical volumes.

2. View the attributes of the two volume groups:

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

-

Verification

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

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

Additional resources

- **vgsplit(8)**, **vgs(8)**, and **pvs(8)** man pages

4.5. MOVING A VOLUME GROUP TO ANOTHER SYSTEM

You can move an entire LVM volume group (VG) to another system 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 *mylv* logical volume:

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

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

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

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

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

4. View the exported volume group:

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

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

5. 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 myvg
```

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 myvg
```

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

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

Additional resources

- **vgimport(8)**, **vgexport(8)**, and **vgchange(8)** man pages

4.6. REMOVING LVM VOLUME GROUPS

You can remove an existing volume group using the **vgremove** command.

Prerequisites

- The volume group contains no logical volumes. To remove logical volumes from a volume group, see Removing LVM logical volumes.

Procedure

1. If the volume group exists in a clustered environment, stop the lockspace of the volume group on all other nodes. Use the following command on all nodes except the node where you are performing the removal:

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

   Wait for the lock to stop.

2. Remove the volume group:

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

Additional resources

- **vgremove(8)** man page

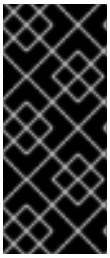
CHAPTER 5. MANAGING LVM LOGICAL VOLUMES

A logical volume is a virtual, block storage device that a file system, database, or application can use. To create an LVM logical volume, the physical volumes (PVs) are combined into a volume group (VG). This creates a pool of disk space out of which LVM logical volumes (LVs) can be allocated.

5.1. OVERVIEW OF LOGICAL VOLUMES

An administrator can grow or shrink logical volumes without destroying data, unlike standard disk partitions. If the physical volumes in a volume group are on separate drives or RAID arrays, then administrators can also spread a logical volume across the storage devices.

You can lose data if you shrink a logical volume to a smaller capacity than the data on the volume requires. Further, some file systems are not capable of shrinking. To ensure maximum flexibility, create logical volumes to meet your current needs, and leave excess storage capacity unallocated. You can safely extend logical volumes to use unallocated space, depending on your needs.



IMPORTANT

On AMD, Intel, ARM systems, and IBM Power Systems servers, the boot loader cannot read LVM volumes. You must make a standard, non-LVM disk partition for your **/boot** partition. On IBM Z, the **zipl** boot loader supports **/boot** on LVM logical volumes with linear mapping. By default, the installation process always creates the / and swap partitions within LVM volumes, with a separate **/boot** partition on a physical volume.

The following are the different types of logical volumes:

Linear volumes

A linear volume aggregates space from one or more physical volumes into one logical volume. For example, if you have two 60GB disks, you can create a 120GB logical volume. The physical storage is concatenated.

Striped logical volumes

When you write data to an LVM logical volume, the file system lays the data out across the underlying physical volumes. You can control the way the data is written to the physical volumes by creating a striped logical volume. For large sequential reads and writes, this can improve the efficiency of the data I/O.
Striping enhances performance by writing data to a predetermined number of physical volumes in round-robin fashion. With striping, I/O can be done in parallel. In some situations, this can result in near-linear performance gain for each additional physical volume in the stripe.

RAID logical volumes

LVM supports RAID levels 0, 1, 4, 5, 6, and 10. RAID logical volumes are not cluster-aware. 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-provisioned logical volumes (thin volumes)

Using thin-provisioned logical volumes, you can create logical volumes that are larger than the available physical storage. Creating a thinly provisioned set of volumes allows the system to allocate what you use instead of allocating the full amount of storage that is requested

Snapshot volumes

The LVM snapshot feature provides the ability to create virtual images of a device at a particular instant without causing a service interruption. When a change is made to the original device (the

origin) after a snapshot is taken, the snapshot feature makes a copy of the changed data area as it was prior to the change so that it can reconstruct the state of the device.

Thin-provisioned snapshot volumes

Using thin-provisioned snapshot volumes, you can have more virtual devices to be stored on the same data volume. Thinly provisioned snapshots are useful because you are not copying all of the data that you are looking to capture at a given time.

Cache volumes

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.

5.2. CREATING LVM LOGICAL VOLUME

Prerequisites

- The **lvm2** package is installed.
- The volume group is created. For more information, see Creating LVM volume group.

Procedure

1. Create a logical volume:

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

Use the **-n** option to set the LV name to *mylv*, and the **-L** option to set the size of LV in units of Mb, but it is possible to use any other units. The LV type is linear by default, but the user can specify the desired type by using the **--type** option.

IMPORTANT

The command fails if the VG does not have a sufficient number of free physical extents for the requested size and type.

2. View the created logical volumes by using any one of the following commands as per your requirement:

   a. The **lvs** command provides logical volume information in a configurable form, displaying one line per logical volume:

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

   b. The **lvdisplay** command displays logical volume properties, such as size, layout, and mapping in a fixed format:

```
# lvdisplay -v /dev/myvg/mylv
  --- Logical volume ---
  LV Path                /dev/myvg/mylv
```

```
  LV Name                mylv
  VG Name                myvg
  LV UUID                YTnAk6-kMIT-c4pG-HBFZ-Bx7t-ePMk-7YjhaM
  LV Write Access        read/write
[..]
```

c. The **lvscan** command scans for all logical volumes in the system and lists them:

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

3. Create a file system on the logical volume. The following command creates an **xfs** file system on the logical volume:

```
# mkfs.xfs /dev/myvg/mylv
meta-data=/dev/myvg/mylv       isize=512    agcount=4, agsize=32000 blks
         =                     sectsz=512   attr=2, projid32bit=1
         =                     crc=1        finobt=1, sparse=1, rmapbt=0
         =                     reflink=1
data     =                       bsize=4096   blocks=128000, imaxpct=25
         =                     sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                     sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
Discarding blocks...Done.
```

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

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

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

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

Additional resources

- **lvcreate(8)**, **lvdisplay(8)**, **lvs(8)**, **lvscan(8)**, **lvm(8)** and **mkfs.xfs(8)** man pages

5.3. 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 LVM physical volume.

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

5.4. RENAMING LVM LOGICAL VOLUMES

This procedure describes how to rename an existing logical volume *mylv* to *mylv1*.

Procedure

1. If the logical volume is currently mounted, unmount the volume:

```
# umount /mnt
```

   Replace */mnt* with the mount point.

2. Rename an existing logical volume:

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

   You can also rename the logical volume by specifying the full paths to the devices:

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

Additional resources

- **lvrename(8)** man page

5.5. REMOVING A DISK FROM A LOGICAL VOLUME

This procedure describes how to remove a disk from an existing logical volume, either to replace the disk or to use the disk as part of a different volume.

In order to remove a disk, you must first move the extents on the LVM physical volume to a different disk or set of disks.

Procedure

1. View the used and free space of physical volumes when using the LV:

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

2. Move the data to other physical volume:

   a. If there are enough free extents on the other physical volumes in the existing volume group, use the following command to move the data:

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

   b. If there are no enough free extents on the other physical volumes in the existing volume group, use the following commands to add a new physical volume, extend the volume group using the newly created physical volume, and move the data to this physical volume:

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

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

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

3. Remove the physical volume:

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

If a logical volume contains a physical volume that fails, you cannot use that logical volume. To remove missing physical volumes from a volume group, you can use the **--removemissing** parameter of the **vgreduce** command, if there are no logical volumes that are allocated on the missing physical volumes:

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

Additional resources

- **pvmove(8)**, **vgextend(8)**, **vereduce(8)**, and **pvs(8)** man pages

5.6. REMOVING LVM LOGICAL VOLUMES

This procedure describes how to remove an existing logical volume */dev/myvg/mylv1* from the volume group *myvg*.

Procedure

1. If the logical volume is currently mounted, unmount the volume:

```
# umount /mnt
```

2. If the logical volume exists in a clustered environment, deactivate the logical volume on all nodes where it is active. Use the following command on each such node:

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

3. Remove the logical volume using the **lvremove** utility:

```
# lvremove /dev/myvg/mylv1

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

NOTE

In this case, the logical volume has not been deactivated. If you explicitly deactivated the logical volume before removing it, you would not see the prompt verifying whether you want to remove an active logical volume.

Additional resources

- **lvremove(8)** man page

5.7. MANAGING LVM LOGICAL VOLUMES BY USING RHEL SYSTEM ROLES

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

- Create an LVM logical volume in a volume group consisting of multiple disks.
- Create an ext4 file system with a given label on the logical volume.
- Persistently mount the ext4 file system.

Prerequisites

- An Ansible playbook including the **storage** role

5.7.1. Managing logical volumes by using the storage RHEL System Role

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

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 on them.

Procedure

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

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

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

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
```

Additional resources

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

5.7.2. Additional resources

- For more information about the **storage** role, see Managing local storage by using RHEL System Roles.

5.8. REMOVING LVM VOLUME GROUPS

You can remove an existing volume group using the **vgremove** command.

Prerequisites

- The volume group contains no logical volumes. To remove logical volumes from a volume group, see Removing LVM logical volumes.

Procedure

1. If the volume group exists in a clustered environment, stop the lockspace of the volume group on all other nodes. Use the following command on all nodes except the node where you are performing the removal:

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

Wait for the lock to stop.

2. Remove the volume group:

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

Additional resources

- **vgremove(8)** man page

CHAPTER 6. MODIFYING THE SIZE OF A LOGICAL VOLUME

After you have created a logical volume, you can modify the size of the volume.

6.1. EXTENDING A LOGICAL VOLUME AND FILE SYSTEM

You can extend a logical volume (LV) using the **lvextend** command. You can specify by how much you want to extend the LV, or how large you want the LV to be after you extend it. Use the **-r** option of the **lvextend** command to grow the underlying file system along with the LV.

WARNING

You can also extend logical volumes using the **lvresize** command, but this command does not guarantee against accidental shrinkage.

Prerequisites

- You have an existing logical volume (LV) with a file system on it. Determine the file system type and size using the **df -Th** command. For more information about creating a logical volume and a file system, see Creating LVM logical volume.
- You have sufficient space in the volume group to grow your LV and file system. Use the **vgs -o name,vgfree** command to determine the available space. For more information about creating volume groups, see Creating LVM volume group.

Procedure

1. Optional: If the volume group has insufficient space to grow your LV, add a new physical volume to the volume group:

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

2. Extend the LV and the file system:

NOTE

Using the **lvextend** command without the **-r** argument extends the LV only. To extend an underlying XFS file system, see Increasing the size of an XFS file system, for a GFS2 file system, see Growing a GFS2 filesystem and for an ext4 file system, see Resizing an ext4 file system.

NOTE

Use the **-L** option to extend the LV to a new size and the **-l** option to specify the number of extents depending on the size of the logical volume that you want to increase.

```
# lvextend -r -L 3G /dev/myvg/mylv
fsck from util-linux 2.32.1
/dev/mapper/myvg-mylv: clean, 11/131072 files, 26156/524288 blocks
  Size of logical volume myvg/mylv changed from 2.00 GiB (512 extents) to 3.00 GiB (768
extents).
  Logical volume myvg/mylv successfully resized.
resize2fs 1.45.6 (20-Mar-2020)
Resizing the filesystem on /dev/mapper/myvg-mylv to 786432 (4k) blocks.
The filesystem on /dev/mapper/myvg-mylv is now 786432 (4k) blocks long.
```

You can also extend the *mylv* logical volume to fill all of the unallocated space in the *myvg* volume group:

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

Verification

- Verify that the file system and the LV has grown:

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

Additional resources

- **vgextend(8)**, **lvextend(8)**, and **xfs_growfs(8)** man pages

6.2. REDUCING A LOGICAL VOLUME AND FILE SYSTEM

You can reduce a logical volume and its file system by using the **lvreduce** command and the **resizefs** option.

If the logical volume you are reducing contains a file system, to prevent data loss you must ensure that the file system is not using the space in the logical volume that is being reduced. For this reason, use the **--resizefs** option of the **lvreduce** command when the logical volume contains a file system.

When you use **--resizefs**, **lvreduce** attempts to reduce the file system before shrinking the logical volume. If shrinking the file system fails because it is full or does not support shrinking, then the **lvreduce** command fails and does not attempt to reduce the logical volume.

WARNING

In most cases, the **lvreduce** command warns about possible data loss and asks for confirmation. However, you should not rely on these confirmation prompts to prevent data loss because in some cases you will not see these prompts, such as when the logical volume is inactive or the **--resizefs** option is not used.

Note that using the **--test** option of the **lvreduce** command does not indicate if the operation is safe because this option does not check the file system or test the file system resize.

Prerequisites

- File system of the logical volume supports shrinking. Determine the file system type and size using the **df -Th** command.

NOTE

For example, the GFS2 and XFS filesystems do not support shrinking.

- Underlying file system is not using the space in the LV that is being reduced.

Procedure

1. Shrink the *mylv* logical volume and its filesystem in the *myvg* volume group using one of the following options:
   - Reduce the LV and its file system to a desired value:

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

   - Reduce 64 megabytes from the logical volume and filesystem:

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

Additional resources

- **lvreduce(8)** man page

6.3. EXTENDING A STRIPED LOGICAL VOLUME

You can extend a striped logical volume (LV) by using the **lvextend** command with the required size.

Prerequisites

1. You have enough free space on the underlying physical volumes (PVs) that make up the volume group (VG) to support the stripe.

Procedure

1. **Optional:** Display your volume group:

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

2. **Optional:** Create a stripe using the entire amount of space in the volume group:

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

3. **Optional:** Extend the *myvg* volume group by adding new physical volumes:

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

   Repeat this step to add sufficient physical volumes depending on your stripe type and the amount of space used. For example, for a two-way stripe that uses up the entire volume group, you need to add at least two physical volumes.

4. Extend the striped logical volume *stripe1* that is a part of the *myvg* VG:

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

   You can also extend the *stripe1* logical volume to fill all of the unallocated space in the *myvg* volume group:

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

Verification

- Verify the new size of the extended striped LV:

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

CHAPTER 7. CUSTOMIZING THE LVM REPORT

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

7.1. CONTROLLING FORMAT OF THE LVM DISPLAY

Whether you use **pvs**, **lvs**, or **vgs**, these commands determine the default set of fields displayed and the sort order. You can control the output of these commands by executing the following commands.

Procedure

- Change the default fields in the LVM display using the **-o** option:

```
# 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
```

- Sort LVM display by using the **-O** option:

```
# 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
```

- Display a reverse sort by using the **-O** argument 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

- **lvmreport(7)**, **lvs(8)**, **vgs(8)**, and **pvs(8)** man page
- Specifying the units for an LVM report display
- Customizing the LVM configuration file

7.2. SPECIFYING THE UNITS FOR AN LVM REPORT DISPLAY

You can view the size of the LVM devices in base 2 or base 10 units by specifying the **--units** argument of the report command.

Base 2 units

The default units are displayed in powers of 2, which is multiples of 1024. You can specify it using human-readable (**r**) with **<** and **>** rounding indicator, bytes (**b**), sectors (**s**), kilobytes (**k**), megabytes (**m**), gigabytes (**g**), terabytes (**t**), petabytes (**p**), exabytes (**e**), and human-readable (**h**).
The default display is **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.

Base 10 units

You can specify the units to be displayed in multiples of 1000 by capitalizing the unit specification (**R**, **B**, **S**, **K**, **M**, **G**, **T**, **P**, **E**, **H**).

Procedure

- Specify the units for the LVM for base 2 gigabytes units:

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

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

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

- Indicate the actual size of LVM by using the **r** option with the **<** or **>** prefix in the output:

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

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

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

The **r** unit works similarly to **h** (human-readable), but in addition, the reported value gets a prefix of **<** or **>** to indicate that the actual size is slightly more or less than the displayed size. LVM rounds the decimal value, causing non-exact sizes to be reported.

It also shows how **--units g** or other **--units** do not always display exactly correct sizes. It also shows the primary purpose of **r**, which is the **<** to indicate that the displayed size is not exact. In this example, the value is not exact because the VG size is not an exact multiple of gigabytes, and .01 is also not an exact representation of the fraction.

- Specify the units for the LVM for base 10 gigabytes units:

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

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

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

- Specify sectors (**s**), defined as 512 bytes, or custom units. The following example displays the output of the **pvs** command as several sectors:

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

- Specify megabytes (**m**). The following example displays the output of the **pvs** command in units of 4 MB:

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

7.3. CUSTOMIZING THE LVM CONFIGURATION FILE

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

Procedure:

1. Display the default **lvm.conf** file:

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

   By default, the **lvm.conf** file contains only comments to display possible settings.

2. Customize the **lvm.conf** file according to your requirements by uncommenting the setting in **lvm.conf**. The following setting focuses on changing the default display of certain commands:

   - In the **lvm.conf** file, adjust the **lvs_cols** parameter to only print the specified fields:

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

     Use this option instead of the **lvs -o lv_name,vg_name,lv_attr** command to avoid unnecessary frequent use of the **-o** option.

   - In the **lvm.conf** file, use the **compact_output=1** setting to avoid printing empty fields for the **pvs**, **vgs**, and **lvs** commands:

```
{
```

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

3. View the default values after modifying the **lvm.conf** file:

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

Additional resources

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

7.4. DEFINING LVM SELECTION CRITERIA

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. Statements are combined by logical and grouping operators. To define selection criteria use the **-S** or **--select** option followed by one or multiple statements.

Some LVM commands support the **-S** option to select which objects to process based on certain attributes. These objects can be physical volumes (PVs), volume groups (VGs), or logical volumes (LVs).

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 select option can also be used as a shortcut to avoid typing many names.

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

Use selection criteria with LVM reporting and processing commands to only display or process the objects that satisfy chosen criteria:

- Reporting commands include **pvs**, **vgs**, **lvs**, **pvdisplay**, **vgdisplay**, **lvdisplay**, and **dmsetup info -c**.
- Processing commands include **pvchange**, **vgchange**, **lvchange**, **vgimport**, **vgexport**, **vgremove**, and **lvremove**.

Procedure

- Examples of selection criteria using the **pvs** command:

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

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

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

- Examples of selection criteria using the **lvs** commands:

```
# lvs
  LV   VG   Attr       LSize   Cpy%Sync
  mylv  myvg -wi-a----- 200.00m
  lvol0 myvg -wi-a----- 100.00m
  lvol1 myvg -wi-a----- 100.00m
  lvol2 myvg -wi------- 100.00m
  rr    myvg rwi-a-r--- 120.00m 100.00
```

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

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

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

- More advanced examples:

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

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

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

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

Additional resources

- **lvmreport(7)** man page

CHAPTER 8. CONFIGURING LVM ON SHARED STORAGE

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

8.1. 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 doesn't 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, filter the path to exclude the device:

```
filter = [ "r|^path_to_device$|" ]
```

2. 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>
```

8.2. CONFIGURING LVM TO USE SAN DISKS ON ONE MACHINE

To prevent the SAN LUNs from being used by the wrong machine, exclude those disks in the **lvm.conf** filter on all machines except the one machine which is meant to use them.

You can also protect the VG from being used by the wrong machine by configuring a **system ID** on all machines, and setting the **system ID** in the VG to match the machine using it.

Procedure

1. In the **lvm.conf** file filter the path to the device to exclude it:

```
filter = [ "r|^path_to_device$|" ]
```

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

```
system_id_source = "uname"
```

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

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

8.3. 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.conf** filter to include the LUNs on all machines that may use them and by configuring the LVM **system ID** on each machine.

The following procedure describes the initial LVM configuration, to finish setting up LVM for failover and move the VG between machines, you need to configure **pacemaker** and LVM-activate resource agent that will automatically modify the VG's system ID to match the system ID of the machine where the VG can be used. For more information see Configuring and managing high availability clusters.

Procedure

1. In the **lvm.conf** file, filter the path to exclude the device:

```
filter = [ "a|^path_to_device$|" ]
```

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

```
system_id_source = "uname"
```

8.4. CONFIGURING LVM TO SHARE SAN DISKS AMONG MULTIPLE MACHINES

Using the **lvmlockd** daemon and a lock manager such as **dlm** or **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. The following procedure describes the overview of the required steps to configure LVM to share SAN disks among multiple machines.

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. Configure the **lvm.conf** filter to include the LUNs of all machines that will use them:

```
filter = ["a|^path_to_device$|" ]
```

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

```
use_lvmlockd=1
```

3. Start the **lvmlockd** daemon file on all machines.
4. Start a lock manager to use with **lvmlockd**, such as **dlm** or **sanlock** on all machines.
5. Create a new shared VG using the command **vgcreate --shared**.
6. Start and stop access to existing shared VGs using the commands **vgchange --lockstart** and **vgchange --lockstop** on all machines.

Additional resources

- **lvmlockd(8)** man page

CHAPTER 9. CONFIGURING RAID LOGICAL VOLUMES

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

9.1. RAID LOGICAL VOLUMES

Logical volume manager (LVM) supports Redundant Array of Independent Disks (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.

Other characteristics include:

Clusters

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.

Subvolumes

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**). Similarly, creating a 3-way stripe and one implicit parity device, RAID4 results in four metadata subvolumes (**lv_rmeta_0**, **lv_rmeta_1**, **lv_rmeta_2**, and **lv_rmeta_3**) and four data subvolumes (**lv_rimage_0**, **lv_rimage_1**, **lv_rimage_2**, and **lv_rimage_3**).

Integrity

You can lose data when a RAID device fails or when soft corruption occurs. Soft corruption in data storage implies that the data retrieved from a storage device is different from the data written to that device. Adding integrity to a RAID LV reduces or prevent soft corruption. For more information, see Creating a RAID LV with DM integrity .

9.2. RAID LEVELS AND LINEAR SUPPORT

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.

9.3. LVM RAID SEGMENT TYPES

To create a RAID logical volume, you can specify a RAID type by using the **--type** argument of the **lvcreate** command. For most users, specifying one of the five available primary types, which are **raid1**, **raid4**, **raid5**, **raid6**, and **raid10**, should be sufficient.

The following table describes the possible RAID segment types.

Table 9.1. 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.

Segment type	Description
raid5_ls	• RAID5 left symmetric. • It is same as **raid5**. • Rotating parity 0 with data restart.
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.

9.4. 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)
```

Additional resources

- **lvcreate(8)** and **lvmraid(7)** man pages

9.5. 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 LVM physical volume.

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

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

With the **storage** System Role, you can configure a stripe size for RAID LVM volumes on RHEL by using Red Hat Ansible Automation Platform. You can set up an Ansible playbook with the available parameters to configure an LVM pool with RAID.

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 on them.

Procedure

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

```
---
- name: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
```

```
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            raid_level: raid1
            raid_stripe_size: "256 KiB"
            state: present
```

2. 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
```

Additional resources

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

9.7. 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 9.2. Parameters for creating a RAID0 striped logical volume

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. Hence, 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.

Parameter	Description
--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.

9.8. 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.

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. The integrity layer sits above each RAID image while an extra sub LV stores the integrity metadata or data checksums for each RAID image. When you retrieve data from an RAID LV with integrity, the integrity data checksums analyze the data for corruption. If corruption is detected, the integrity layer returns an error message, and the RAID layer retrieves a non-corrupted copy of the data from another RAID image. The RAID layer automatically rewrites non-corrupted data over the corrupted data to repair the soft corruption.

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.

9.9. CREATING A RAID LV WITH DM INTEGRITY

When you create a RAID LV with device mapper (DM) integrity or add integrity to an existing RAID 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.

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
Creating integrity metadata LV test-lv_rimage_0_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_0_imeta" created.
Creating integrity metadata LV test-lv_rimage_1_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_1_imeta" created.
Logical volume "test-lv" created.
```

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:
  a. View information about the test-lv RAID LV that was created in the my_vg volume group:

- 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, as shown in the following examples:

Example 9.1. Example of dm-integrity mismatches from the kernel message logs

device-mapper: integrity: dm-12: Checksum failed at sector 0x24e7

Example 9.2. Example of dm-integrity data corrections from the kernel message logs

md/raid1:mdX: read error corrected (8 sectors at 9448 on dm-16)

Additional resources

- **lvcreate(8)** and **lvmraid(7)** man pages

9.10. MINIMUM AND MAXIMUM I/O RATE OPTIONS

When you create a RAID logical volumes, the background I/O required to initialize the logical volumes with the sync operation can expel other I/O operations to LVM devices, such as updates to volume group metadata, particularly when you are creating many RAID logical volumes. This can cause the other LVM operations to slow down.

You can control the rate at which a RAID logical volume is initialized by implementing recovery throttling. To control the rate at which **sync** operations are performed, set the minimum and maximum I/O rate for those operations with the **--minrecoveryrate** and **--maxrecoveryrate** options of the **lvcreate** command.

You can specify these options as follows:

--maxrecoveryrate Rate[bBsSkKmMgG]

Sets the maximum recovery rate for a RAID logical volume so that it will not expel nominal I/O operations. Specify the Rate as an amount per second for each device in the array. If you do not provide a suffix, then it assumes kiB/sec/device. Setting the recovery rate to 0 means it will be unbounded.

--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. Specify the Rate as an amount per second for each device in the array. If you do not give a suffix, then it assumes kiB/sec/device.

For example, use the **lvcreate --type raid10 -i *2* -m *1* -L *10G* --maxrecoveryrate *128* -n *my_lv my_vg*** command to create 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. You can also specify minimum and maximum recovery rates for a RAID scrubbing operation.

9.11. 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)
```

Additional resources

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

9.12. 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)
```

Additional resources

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

9.13. 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)
```

Additional resources

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

9.14. COMMANDS TO RESIZE A RAID LOGICAL VOLUME

You can resize a RAID logical volume in the following ways:

- You can increase the size of a RAID logical volume of any type with the **lvresize** or **lvextend** command. This does not change the number of RAID images. For striped RAID logical volumes, the same stripe rounding constraints apply when you create a striped RAID logical volume.
- You can reduce the size of a RAID logical volume of any type with the **lvresize** or **lvreduce** command. This does not change the number of RAID images. As with the **lvextend** command, the same stripe rounding constraints apply when you create a striped RAID logical volume.
- You can change the number of stripes on a striped RAID logical volume such as RAID4, RAID5, RAID6, or RAID10 with the **--stripes *N*** parameter of the **lvconvert** command. This increases or reduces the size of the RAID logical volume by the capacity of the stripes added or removed. Note that raid10 volumes are capable only of adding stripes. This capability is part of the RAID reshaping feature and with this feature, you can change attributes of a RAID logical volume while keeping the same RAID level.

9.15. 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)
```

Additional resources

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

9.16. 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)
```

Additional resources

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

9.17. 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              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sdc1(1)
  [my_lv_rimage_1]         /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

5. Merge the volume back into the array:

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

Verification

- View the merged 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/sdc1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

Additional resources

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

9.18. SETTING A RAID FAULT POLICY

Based on the **raid_fault_policy** field preferences in the **/etc/lvm/lvm.conf** file, LVM RAID automatically handles device failures. You can set **raid_fault_policy** field to any one of the following parameter depending on the requirement:

warn

You can this parameter to manually repair the failed device and display warnings by using system logs.
By default, the value of the **raid_fault_policy** field is **warn** in **lvm.conf**. If enough devices are operational, the RAID logical volume continues to operate.

allocate

You can use this parameter to automatically replace the failed device.

9.18.1. 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. For more information, see Replacing a failed RAID device in a logical volume .

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

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

9.18.2. 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.

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

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

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

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. For more information, see Replacing a failed RAID device in a logical volume .

Additional resources

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

9.19. REPLACING A RAID DEVICE IN A LOGICAL VOLUME

You can replace a RAID device in a logical volume depending on the following scenarios:

- Replacing a working RAID device.
- Replacing a failed RAID device in a logical volume.

9.19.1. 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)
```

Additional resources

- **lvconvert(8)** man page

9.19.2. 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)
```

Additional resources

- **lvconvert(8)** and **vgreduce(8)** man pages

9.20. 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. The following attributes provide details about data coherency:

- The **raid_sync_action** field displays the current synchronization action that the RAID logical volume is performing. It can be one of the following values:

  idle

  Completed all **sync** actions (doing nothing).

  resync

  Initializing or resynchronizing an array after an unclean machine shutdown.

  recover

  Replacing a device in the array.

  check

  Looking for array inconsistencies.

  repair

  Looking for and repairing inconsistencies.

- The **raid_mismatch_count** field displays the number of discrepancies found during a **check** action.

- The **Cpy%Sync** field displays the progress of the **sync** actions.

- The **lv_attr** field provides additional indicators. Bit 9 of this field displays the health of the logical volume, and it supports the following indicators:

  m or **mismatches**

  Indicates that there are discrepancies in a RAID logical volume. You can see this character after the scrubbing operation detects the portions of the RAID, which are not coherent.

  r or **refresh**

  Indicates a failed device in a RAID array, even though LVM can read the device label and considers the device to be operational. Refresh the logical volume to notify the kernel that the device is now available, or replace the device if you suspect that it failed.

Procedure

1. Optional: Limit the I/O bandwidth that the scrubbing process uses. 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.
   You can control the rate of the scrubbing operation by implementing recovery throttling. You can set the recovery rate using **--maxrecoveryrate Rate[bBsSkKmMgG]** or **--minrecoveryrate Rate[bBsSkKmMgG]** with the **lvchange --syncaction** commands. For more information, see Minimum and maximum I/O rate options .

   Specify the *Rate* value as an amount per second for each device in the array. If you provide no suffix, the options assume kiB per second per 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
   ```

Additional resources

- **lvchange(8)** and **lvmraid(7)** man pages
- Minimum and maximum I/O rate options

9.21. 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
```

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--- 172.00m
/dev/sda(1)                                                         linear
```

Additional resources

- **lvconvert(8)** and **lvmraid(8)** man pages

9.22. 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/ly_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, 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/ly_lv* command. Setting the **writemostly** attribute's value to zero clears the preference. With this setting, the system chooses the value arbitrarily.

9.23. 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.
```

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
  [...]
```

Additional resources

- **lvconvert(8)** and **lvmraid(8)** man pages

9.24. 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:

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

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

4. Change the region size of a RAID logical volume:

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

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

5. Resynchronize the RAID logical volume:

```
# lvchange --resync my_vg/my_lv

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:

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

# Configuration option activation/raid_region_size.
 # raid_region_size = 4096
```

Additional resources

- **lvconvert(8)** man page

CHAPTER 10. SNAPSHOT OF LOGICAL VOLUMES

Using the LVM snapshot feature, you can create virtual images of a volume, for example, */dev/sda*, at a particular instant without causing a service interruption.

10.1. OVERVIEW OF SNAPSHOT VOLUMES

When you modify the original volume (the origin) after you take a snapshot, the snapshot feature makes a copy of the modified data area as it was prior to the change so that it can reconstruct the state of the volume. When you create a snapshot, full read and write access to the origin stays possible.

Since a snapshot copies only the data areas that change after the snapshot is created, the snapshot feature requires a minimal amount of storage. For example, with a rarely updated origin, 3-5 % of the origin's capacity is sufficient to maintain the snapshot. It does not provide a substitute for a backup procedure. Snapshot copies are virtual copies and are not an actual media backup.

The size of the snapshot controls the amount of space set aside for storing the changes to the origin volume. For example, if you create a snapshot and then completely overwrite the origin, the snapshot should be at least as big as the origin volume to hold the changes. You should regularly monitor the size of the snapshot. For example, a short-lived snapshot of a read-mostly volume, such as /**usr**, would need less space than a long-lived snapshot of a volume because it contains many writes, such as /**home**.

If a snapshot is full, the snapshot becomes invalid because it can no longer track changes on the origin volume. But you can configure LVM to automatically extend a snapshot whenever its usage exceeds the **snapshot_autoextend_threshold** value to avoid snapshot becoming invalid. Snapshots are fully resizable and you can perform the following operations:

- If you have the storage capacity, you can increase the size of the snapshot volume to prevent it from getting dropped.
- If the snapshot volume is larger than you need, you can reduce the size of the volume to free up space that is needed by other logical volumes.

The snapshot volume provide the following benefits:

- Most typically, you take a snapshot when you need to perform a backup on a logical volume without halting the live system that is continuously updating the data.
- You can execute the **fsck** command on a snapshot file system to check the file system integrity and determine if the original file system requires file system repair.
- Since the snapshot is read/write, you can test applications against production data by taking a snapshot and running tests against the snapshot without touching the real data.
- You can create LVM volumes for use with Red Hat Virtualization. You can use LVM snapshots to create snapshots of virtual guest images. These snapshots can provide a convenient way to modify existing guests or create new guests with minimal additional storage.

10.2. CREATING A SNAPSHOT OF THE ORIGINAL VOLUME

Use the **lvcreate** command to create a snapshot of the original volume (the origin). A snapshot of a volume is writable. By default, a snapshot volume is activated with the origin during normal activation commands as compared to the thinly-provisioned snapshots. LVM does not support creating a snapshot

volume that is larger than the sum of the origin volume's size and the required metadata size for the volume. If you specify a snapshot volume that is larger than this, LVM creates a snapshot volume that is required for the size of the origin.

NOTE

The nodes in a cluster do not support LVM snapshots. You cannot create a snapshot volume in a shared volume group. However, if you need to create a consistent backup of data on a shared logical volume you can activate the volume exclusively and then create the snapshot.

The following procedure creates an origin logical volume named *origin* and a snapshot volume of this original volume named *snap*.

Prerequisites

- You have created volume group *vg001*. For more information, see Creating LVM volume group.

Procedure

1. Create a logical volume named *origin* from the volume group *vg001*:

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

2. Create a snapshot logical volume named *snap* of */dev/vg001/origin* that is *100 MB* in size:

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

You can also use the **-L** argument instead of using **--size**, **-n** instead of using **--name**, and **-s** instead of using **--snapshot** to create a snapshot.

If the original logical volume contains a file system, you can mount the snapshot logical volume on an arbitrary directory in to access the contents of the file system to run a backup while the original file system continues to get updated.

3. Display the origin volume and the current percentage of the snapshot volume being used:

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

You can also display the status of logical volume */dev/vg001/origin* with all the snapshot logical volumes and their status, such as active or inactive by using the **lvdisplay** ***/dev/vg001/origin*** command.

WARNING

Space in the snapshot LV is consumed after the origin LV is written to. The **lvs** command reports the current snapshot space usage in the **Data%** data_percent field value. If the snapshot space reaches 100%, the snapshot becomes invalid and unusable.

An invalid snapshot is reported with **I** in the fifth position of the **Attr** column, or the **lv_snapshot_invalid** reporting field in **lvs**. You can remove the invalid snapshot by using the **lvremove** command.

4. Optional: Extend the snapshot before its space becomes 100% full and becomes invalid by using any one of the following options:

   - Configure LVM to automatically extend the snapshot by using the following parameters in the **/etc/lvm.conf** file:

     snapshot_autoextend_threshold

     Extends the snapshot after its usage exceeds the value set for this parameter. By default, it is set to 100, which disables automatic extension. The minimum value of this parameter is 50.

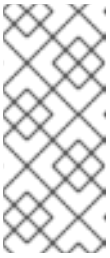
     snapshot_autoextend_percent

     Adds an additional space to the snapshot, which is the percent of its current size. By default, it is set to 20.

     In the following example, after setting the following parameters, the created 1G snapshot extends to 1.2G when its usage exceeds 700M:

     Example 10.1. Automatically extend the snapshot

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

     NOTE

     This feature requires unallocated space in the volume group. An automatic extension of a snapshot does not increase the size of a snapshot volume beyond the maximum calculated size that is necessary for the snapshot. Once a snapshot has grown large enough to cover the origin, it is no longer monitored for automatic extension.

   - Extend this snapshot manually by using the **lvextend** command:

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

Additional resources

- **lvcreate(8)**, **lvextend(8)**, and **lvs(8)** man pages
- **/etc/lvm/lvm.conf** file

10.3. MERGING SNAPSHOT TO ITS ORIGINAL VOLUME

Use the **lvconvert** command with the **--merge** option to merge a snapshot into its original (the origin) volume. You can perform a system rollback if you have lost data or files, or otherwise you have to restore your system to a previous state. After you merge the snapshot volume, the resulting logical volume has the origin volume's name, minor number, and UUID. While the merge is in progress, reads or writes to the origin appear as they were directed to the snapshot being merged. When the merge finishes, the merged snapshot is removed.

If both the origin and snapshot volume are not open and active, the merge starts immediately. Otherwise, the merge starts after either the origin or snapshot are activated and both are closed. You can merge a snapshot into an origin that cannot be closed, for example a **root** file system, after the origin volume is activated.

Procedure

1. Merge the snapshot volume. The following command merges snapshot volume *vg001/snap* into its *origin*:

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

2. View the origin volume:

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

Additional resources

- **lvconvert(8)** man page

CHAPTER 11. CREATING AND MANAGING THIN PROVISIONED VOLUMES (THIN VOLUMES)

Red Hat Enterprise Linux supports thin provisioned snapshot volumes and logical volumes.

Logical volumes and snapshot volumes can be thinly provisioned:

- Using thin-provisioned logical volumes, you can create logical volumes that are larger than the available physical storage.
- Using thin-provisioned snapshot volumes, you can store more virtual devices on the same data volume.

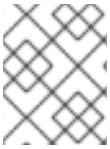
11.1. OVERVIEW OF THIN PROVISIONING

Many modern storage stacks now provide the ability to choose between thick provisioning and thin provisioning:

- Thick provisioning provides the traditional behavior of block storage where blocks are allocated regardless of their actual usage.
- Thin provisioning grants the ability to provision a larger pool of block storage that may be larger in size than the physical device storing the data, resulting in over-provisioning. Over-provisioning is possible because individual blocks are not allocated until they are actually used. If you have multiple thin-provisioned devices that share the same pool, then these devices can be over-provisioned.

By using thin provisioning, you can over-commit the physical storage, and instead can manage a pool of free space known as a thin pool. You can allocate this thin pool to an arbitrary number of devices when needed by applications. You can expand the thin pool dynamically when needed for cost-effective allocation of storage space.

For example, if ten users each request a 100GB file system for their application, then you can create what appears to be a 100GB file system for each user but which is backed by less actual storage that is used only when needed.



NOTE

When using thin provisioning, it is important that you monitor the storage pool and add more capacity as the available physical space runs out.

The following are a few advantages of using thin-provisioned devices:

- You can create logical volumes that are larger than the available physical storage.
- You can have more virtual devices to be stored on the same data volume.
- You can create file systems that can grow logically and automatically to support the data requirements and the unused blocks are returned to the pool for use by any file system in the pool

The following are the potential drawbacks of using thin-provisioned devices:

- Thin-provisioned volumes have an inherent risk of running out of available physical storage. If you have over-provisioned your underlying storage, it could possibly result in an outage due to

the lack of available physical storage. For example, if you create 10T of thinly provisioned storage with only 1T physical storage for backing, the volumes will become unavailable or unwritable after the 1T is exhausted.

- If volumes are not sending discards to the layers after thin-provisioned devices, then the accounting for usage will not be accurate. For example, placing a file system without the **-o discard mount** option and not running **fstrim** periodically on top of thin-provisioned devices will never unallocate previously used storage. In such cases, you end up using the full provisioned amount over time even if you are not really using it.
- You must monitor the logical and physical usage so as to not run out of available physical space.
- 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 of the underlying storage even when it does not appear that way to the end user.

11.2. CREATING THINLY-PROVISIONED LOGICAL VOLUMES

Using thin-provisioned logical volumes, you can create logical volumes that are larger than the available physical storage. Creating a thinly provisioned set of volumes allows the system to allocate what you use instead of allocating the full amount of storage that is requested.

Using the **-T** or **--thin** option of the **lvcreate** command, you can create either a thin pool or a thin volume. You can also use the **-T** option of the **lvcreate** command to create both a thin pool and a thin volume at the same time with a single command. This procedure describes how to create and grow thinly-provisioned logical volumes.

Prerequisites

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

Procedure

1. Create a thin pool:

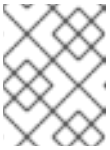
```
# lvcreate -L 100M -T vg001/mythinpool
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "mythinpool" created.
```

Note that since you are creating a pool of physical space, you must specify the size of the pool. The **-T** option of the **lvcreate** command does not take an argument; it determines what type of device is to be created from the other options that are added with the command. You can also create thin pool using additional parameters as shown in the following examples:

- You can also create a thin pool using the **--thinpool** parameter of the **lvcreate** command. Unlike the **-T** option, the **--thinpool** parameter requires that you specify the name of the thin pool logical volume you are creating. The following example uses the **--thinpool** parameter to create a thin pool named *mythinpool* in the volume group *vg001* that is *100M* in size:

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

- As striping is supported for pool creation, you can use the **-i** and **-I** options to create stripes. The following command creates a *100M* thin pool named as *thinpool* in volume group *vg001* with two *64 kB* stripes and a chunk size of *256 kB*. It also creates a *1T* thin volume named *vg001/thinvolume*.



NOTE

Ensure that there are two physical volumes with sufficient free space in the volume group or you cannot create the thin pool.

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

2. Create a thin volume:

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

In this case, you are specifying virtual size for the volume that is greater than the pool that contains it. You can also create thin volumes using additional parameters as shown in the following examples:

- To create both a thin volume and a thin pool, use the **-T** option of the **lvcreate** command and specify both the size and virtual size argument:

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

- To use the remaining free space to create a thin volume and thin pool, use the **100%FREE** option:

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

- To convert an existing logical volume to a thin pool volume, use the **--thinpool** parameter of the **lvconvert** command. You must also use the **--poolmetadata** parameter in conjunction with the **--thinpool** parameter to convert an existing logical volume to a thin pool volume's metadata volume.
  The following example converts the existing logical volume *lv1* in volume group *vg001* to a thin pool volume and converts the existing logical volume *lv2* in volume group *vg001* to the metadata volume for that thin pool volume:

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

NOTE

Converting a logical volume to a thin pool volume or a thin pool metadata volume destroys the content of the logical volume, as **lvconvert** does not preserve the content of the devices but instead overwrites the content.

- By default, the **lvcreate** command approximately sets the size of the thin pool metadata logical volume by using the following formula:

```
Pool_LV_size / Pool_LV_chunk_size * 64
```

If you have large numbers of snapshots or if you have have small chunk sizes for your thin pool and therefore expect significant growth of the size of the thin pool at a later time, you may need to increase the default value of the thin pool's metadata volume using the **--poolmetadatasize** parameter of the **lvcreate** command. The supported value for the thin pool's metadata logical volume is in the range between 2MiB and 16GiB.

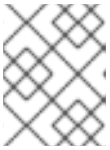
The following example illustrates how to increase the default value of the thin pools' metadata volume:

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

3. View the created thin pool and thin volume:

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

4. Optional: Extend the size of a thin pool with the **lvextend** command. You cannot, however, reduce the size of a thin pool.



NOTE

This command fails if you use **-l 100%FREE** argument while creating a thin pool and thin volume.

The following command resizes an existing thin pool that is *100M* in size by extending it another *100M*:

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

  Logical volume vg001/mythinpool successfully resized
```

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

5. Optional: To rename the thin pool and thin volume, use the following command:

```
# lvrename vg001/mythinpool vg001/mythinpool1
  Renamed "mythinpool" to "mythinpool1" in volume group "vg001"

# lvrename vg001/thinvolume vg001/thinvolume1
  Renamed "thinvolume" to "thinvolume1" in volume group "vg001"
```

View the thin pool and thin volume after renaming:

```
# lvs
  LV          VG       Attr     LSize   Pool       Origin Data%  Move Log Copy%  Convert
mythinpool1 vg001   twi-a-tz 100.00m                    0.00
thinvolume1 vg001   Vwi-a-tz   1.00g mythinpool1         0.00
```

6. Optional: To remove the thin pool, use the following command:

```
# lvremove -f vg001/mythinpool1
  Logical volume "thinvolume1" successfully removed.
  Logical volume "mythinpool1" successfully removed.
```

Additional resources

- **lvcreate(8)**, **lvrename(8)**, **lvs(8)**, and **lvconvert(8)** man pages

11.3. OVERVIEW OF CHUNK SIZE

A chunk is the largest unit of physical disk dedicated to snapshot storage.

Use the following criteria for using the chunk size:

- A smaller chunk size requires more metadata and hinders performance, but provides better space utilization with snapshots.
- A bigger chunk size requires less metadata manipulation, but makes the snapshot less space efficient.

Be default, **lvm2** starts with a 64KiB chunk size and estimates good metadata size for such chunk size. The minimal metadata size **lvm2** can create and use is 2 MiB. If the metadata size needs to be larger than 128 MiB it begins to increase the chunk size, so the metadata size stays compact. However, this may result in some big chunk size values, which are less space efficient for snapshot usage. In such cases, a smaller chunk size and bigger metadata size is a better option.

To specify the chunk size according to your requirement, use the **-c** or **--chunksize** parameter to overrule **lvm2** estimated chunk size. Be aware that you cannot change the chunk size once the thinpool is created.

If the volume data size is in the range of TiB, use ~15.8GiB as the metadata size, which is the maximum supported size, and set the chunk size according to your requirement. But, note that it is not possible to increase the metadata size if you need to extend the volume's data size and have a small chunk size.

NOTE

Using the inappropriate combination of chunk size and metadata size may result in potentially problematic situation, when user runs out of space in **metadata** or they may not further grow their thin-pool size because of limited maximum addressable thin-pool data size.

Additional resources

- **lvmthin(7)** man page

11.4. THINLY-PROVISIONED SNAPSHOT VOLUMES

Red Hat Enterprise Linux supports thinly-provisioned snapshot volumes. A snapshot of a thin logical volume also creates a thin logical volume (LV). A thin snapshot volume has the same characteristics as any other thin volume. You can independently activate the volume, extend the volume, rename the volume, remove the volume, and even snapshot the volume.

NOTE

Similarly to all LVM snapshot volumes, and all thin volumes, thin snapshot volumes are not supported across the nodes in a cluster. The snapshot volume must be exclusively activated on only one cluster node.

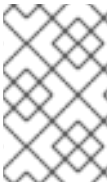
Traditional snapshots must allocate new space for each snapshot created, where data is preserved as changes are made to the origin. But thin-provisioning snapshots share the same space with the origin. Snapshots of thin LVs are efficient because the data blocks common to a thin LV and any of its snapshots are shared. You can create snapshots of thin LVs or from the other thin snapshots. Blocks common to recursive snapshots are also shared in the thin pool.

Thin snapshot volumes provide the following benefits:

- Increasing the number of snapshots of the origin has a negligible impact on performance.

- A thin snapshot volume can reduce disk usage because only the new data is written and is not copied to each snapshot.
- There is no need to simultaneously activate the thin snapshot volume with the origin, which is a requirement of traditional snapshots.
- When restoring an origin from a snapshot, it is not required to merge the thin snapshot. You can remove the origin and instead use the snapshot. Traditional snapshots have a separate volume where they store changes that must be copied back, that is, merged to the origin to reset it.
- There is a significantly higher limit on the number of allowed snapshots as compared to the traditional snapshots.

Although there are many advantages for using thin snapshot volumes, there are some use cases for which the traditional LVM snapshot volume feature might be more appropriate to your needs. You can use traditional snapshots with all types of volumes. However, to use thin-snapshots requires you to use thin-provisioning.



NOTE

You cannot limit the size of a thin snapshot volume; the snapshot uses all of the space in the thin pool, if necessary. In general, you should consider the specific requirements of your site when deciding which snapshot format to use.

By default, a thin snapshot volume is skipped during normal activation commands.

11.5. CREATING THINLY-PROVISIONED SNAPSHOT VOLUMES

Using thin-provisioned snapshot volumes, you can have more virtual devices stored on the same data volume.

IMPORTANT

When creating a thin snapshot volume, do not specify the size of the volume. If you specify a size parameter, the snapshot that will be created will not be a thin snapshot volume and will not use the thin pool for storing data. For example, the command **lvcreate -s vg/thinvolume -L10M** will not create a thin snapshot, even though the origin volume is a thin volume.

Thin snapshots can be created for thinly-provisioned origin volumes, or for origin volumes that are not thinly-provisioned. The following procedure describes different ways to create a thinly-provisioned snapshot volume.

Prerequisites

- You have created a thinly-provisioned logical volume. For more information, see Overview of thin provisioning.

Procedure

- Create a thinly-provisioned snapshot volume. The following command creates a thinly-provisioned snapshot volume named as *mysnapshot1* of the thinly-provisioned logical volume *vg001/thinvolume*:

```
# lvcreate -s --name mysnapshot1 vg001/thinvolume
  Logical volume "mysnapshot1" created
```

```
# lvs
  LV          VG      Attr     LSize   Pool       Origin     Data%  Move Log Copy%  Convert
  mysnapshot1 vg001   Vwi-a-tz   1.00g mythinpool thinvolume   0.00
  mythinpool  vg001   twi-a-tz 100.00m                        0.00
  thinvolume  vg001   Vwi-a-tz   1.00g mythinpool             0.00
```

NOTE

When using thin provisioning, it is important that the storage administrator monitor the storage pool and add more capacity if it starts to become full. For information about extending the size of a thin volume, see Creating thinly-provisioned logical volumes .

- You can also create a thinly-provisioned snapshot of a non-thinly-provisioned logical volume. Since the non-thinly-provisioned logical volume is not contained within a thin pool, it is referred to as an external origin. External origin volumes can be used and shared by many thinly-provisioned snapshot volumes, even from different thin pools. The external origin must be inactive and read-only at the time the thinly-provisioned snapshot is created.
  The following example creates a thin snapshot volume of the read-only, inactive logical volume named *origin_volume*. The thin snapshot volume is named *mythinsnap*. The logical volume *origin_volume* then becomes the thin external origin for the thin snapshot volume *mythinsnap* in volume group *vg001* that uses the existing thin pool *vg001/pool*. The origin volume must be in the same volume group as the snapshot volume. Do not specify the volume group when specifying the origin logical volume.

  ```
  # lvcreate -s --thinpool vg001/pool origin_volume --name mythinsnap
  ```

- You can create a second thinly-provisioned snapshot volume of the first snapshot volume by executing the following command.

  ```
  # lvcreate -s vg001/mysnapshot1 --name mysnapshot2
  Logical volume "mysnapshot2" created.
  ```

  To create a third thinly-provisioned snapshot volume, use the following command:

  ```
  # lvcreate -s vg001/mysnapshot2 --name mysnapshot3
  Logical volume "mysnapshot3" created.
  ```

Verification

- Display a list of all ancestors and descendants of a thin snapshot logical volume:

  ```
  $ lvs -o name,lv_ancestors,lv_descendants vg001
    LV          Ancestors                  Descendants
    mysnapshot2 mysnapshot1,thinvolume     mysnapshot3
    mysnapshot1 thinvolume                 mysnapshot2,mysnapshot3
    mysnapshot3 mysnapshot2,mysnapshot1,thinvolume
    mythinpool
    thinvolume                             mysnapshot1,mysnapshot2,mysnapshot3
  ```

Here,

- *thinvolume* is an origin volume in volume group *vg001*.
- *mysnapshot1* is a snapshot of *thinvolume*
- *mysnapshot2* is a snapshot of *mysnapshot1*
- *mysnapshot3* is a snapshot of *mysnapshot2*

NOTE

The **lv_ancestors** and **lv_descendants** fields display existing dependencies. However, they do not track removed entries which can break a dependency chain if the entry was removed from the middle of the chain.

Additional resources

- **lvcreate(8)** man page

CHAPTER 12. ENABLING CACHING TO IMPROVE LOGICAL VOLUME PERFORMANCE

You can add caching to an LVM logical volume to improve performance. LVM then caches I/O operations to the logical volume using a fast device, such as an SSD.

The following procedures create a special LV from the fast device, and attach this special LV to the original LV to improve the performance.

12.1. CACHING METHODS IN LVM

LVM provides the following kinds of caching. Each one is suitable for different kinds of I/O patterns on the logical volume.

dm-cache

This method speeds up access to frequently used data by caching it on the faster volume. The method caches both read and write operations.
The **dm-cache** method creates logical volumes of the type **cache**.

dm-writecache

This method caches only write operations. The faster volume stores the write operations and then migrates them to the slower disk in the background. The faster volume is usually an SSD or a persistent memory (PMEM) disk.
The **dm-writecache** method creates logical volumes of the type **writecache**.

Additional resources

- **lvmcache(7)** man page

12.2. LVM CACHING COMPONENTS

LVM provides support for adding a cache to LVM logical volumes. LVM caching uses the following LVM logical volume types:

Main LV

The larger, slower, and original volume.

Cache pool LV

A composite LV that you can use for caching data from the main LV. It has two sub-LVs: data for holding cache data and metadata for managing the cache data. You can configure specific disks for data and metadata. You can use the cache pool only with **dm-cache**.

Cachevol LV

A linear LV that you can use for caching data from the main LV. You cannot configure separate disks for data and metadata. **cachevol** can be only used with either **dm-cache** or **dm-writecache**.

All of these associated LVs must be in the same volume group.

You can combine a main logical volume (LV) with a faster, usually smaller, LV that holds the cached data. The fast LV is created from fast block devices, such as SSD drives. When you enable caching for a logical volume, LVM renames and hides the original volumes, and presents a new logical volume that is

composed of the original logical volumes. The composition of the new logical volume depends on the caching method and whether you are using the **cachevol** or **cachepool** option.

The **cachevol** and **cachepool** options expose different levels of control over the placement of the caching components:

- With the **cachevol** option, the faster device stores both the cached copies of data blocks and the metadata for managing the cache.
- With the **cachepool** option, separate devices can store the cached copies of data blocks and the metadata for managing the cache.
  The **dm-writecache** method is not compatible with **cachepool**.

In all configurations, LVM exposes a single resulting device, which groups together all the caching components. The resulting device has the same name as the original slow logical volume.

Additional resources

- **lvmcache(7)** man page
- Creating and managing thin provisioned volumes (thin volumes)

12.3. ENABLING DM-CACHE CACHING FOR A LOGICAL VOLUME

This procedure enables caching of commonly used data on a logical volume using the **dm-cache** method.

Prerequisites

- A slow logical volume that you want to speed up using **dm-cache** exists on your system.
- The volume group that contains the slow logical volume also contains an unused physical volume on a fast block device.

Procedure

1. Create a **cachevol** volume on the fast device:

```
# lvcreate --size cachevol-size --name <fastvol> <vg> </dev/fast-pv>
```

   Replace the following values:

   cachevol-size
   The size of the **cachevol** volume, such as **5G**

   fastvol
   A name for the **cachevol** volume

   vg
   The volume group name

   /dev/fast-pv
   The path to the fast block device, such as **/dev/sdf**

   Example 12.1. Creating a cachevol volume

```
# lvcreate --size 5G --name fastvol vg /dev/sdf
Logical volume "fastvol" created.
```

2. Attach the **cachevol** volume to the main logical volume to begin caching:

```
# lvconvert --type cache --cachevol <fastvol> <vg/main-lv>
```

Replace the following values:

fastvol

The name of the **cachevol** volume

vg

The volume group name

main-lv

The name of the slow logical volume

Example 12.2. Attaching the cachevol volume to the main LV

```
# lvconvert --type cache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]: y
Logical volume vg/main-lv is now cached.
```

Verification steps

- Verify if the newly created logical volume has **dm-cache** enabled:

```
# lvs --all --options +devices <vg>

LV              Pool           Type   Devices
main-lv         [fastvol_cvol] cache  main-lv_corig(0)
[fastvol_cvol]                 linear /dev/fast-pv
[main-lv_corig]                linear /dev/slow-pv
```

Additional resources

- **lvmcache(7)** man page

12.4. ENABLING DM-CACHE CACHING WITH A CACHEPOOL FOR A LOGICAL VOLUME

This procedure enables you to create the cache data and the cache metadata logical volumes individually and then combine the volumes into a cache pool.

Prerequisites

- A slow logical volume that you want to speed up using **dm-cache** exists on your system.

- The volume group that contains the slow logical volume also contains an unused physical volume on a fast block device.

Procedure

1. Create a **cachepool** volume on the fast device:

```
# lvcreate --type cache-pool --size <cachepool-size> --name <fastpool> <vg /dev/fast>
```

Replace the following values:

cachepool-size

The size of the **cachepool**, such as **5G**

fastpool

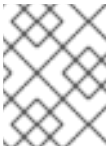
A name for the **cachepool** volume

vg

The volume group name

/dev/fast

The path to the fast block device, such as **/dev/sdf1**



NOTE

You can use **--poolmetadata** option to specify the location of the pool metadata when creating the cache-pool.

Example 12.3. Creating a cachepool volume

```
# lvcreate --type cache-pool --size 5G --name fastpool vg /dev/sde
Logical volume "fastpool" created.
```

2. Attach the **cachepool** to the main logical volume to begin caching:

```
# lvconvert --type cache --cachepool <fastpool> <vg/main>
```

Replace the following values:

fastpool

The name of the **cachepool** volume

vg

The volume group name

main

The name of the slow logical volume

Example 12.4. Attaching the cachepool to the main LV

```
# lvconvert --type cache --cachepool fastpool vg/main
Do you want wipe existing metadata of cache pool vg/fastpool? [y/n]: y
Logical volume vg/main is now cached.
```

Verification steps

- Examine the newly created devicevolume with the **cache-pool** type:

```
# lvs --all --options +devices <vg>

LV                  Pool              Type        Devices
[fastpool_cpool]                      cache-pool  fastpool_pool_cdata(0)
[fastpool_cpool_cdata]                   linear      /dev/sdf1(4)
[fastpool_cpool_cmeta]                    linear      /dev/sdf1(2)
[lvol0_pmspare]                        linear      /dev/sdf1(0)
main                [fastpoool_cpool]  cache       main_corig(0)
[main_corig]                           linear      /dev/sdf1(O)
```

Additional resources

- **lvcreate(8)** man page
- **lvmcache(7)** man page
- **lvconvert(8)** man page

12.5. ENABLING DM-WRITECACHE CACHING FOR A LOGICAL VOLUME

This procedure enables caching of write I/O operations to a logical volume using the **dm-writecache** method.

Prerequisites

- A slow logical volume that you want to speed up using **dm-writecache** exists on your system.
- The volume group that contains the slow logical volume also contains an unused physical volume on a fast block device.
- If the slow logical volume is active, deactivate it.

Procedure

1. If the slow logical volume is active, deactivate it:

```
# lvchange --activate n <vg>/<main-lv>
```

Replace the following values:

vg

The volume group name

main-lv

The name of the slow logical volume

2. Create a deactivated **cachevol** volume on the fast device:

```
# lvcreate --activate n --size <cachevol-size> --name <fastvol> <vg> </dev/fast-pv>
```

Replace the following values:

cachevol-size

The size of the **cachevol** volume, such as **5G**

fastvol

A name for the **cachevol** volume

vg

The volume group name

/dev/fast-pv

The path to the fast block device, such as **/dev/sdf**

Example 12.5. Creating a deactivated cachevol volume

```
# lvcreate --activate n --size 5G --name fastvol vg /dev/sdf
WARNING: Logical volume vg/fastvol not zeroed.
Logical volume "fastvol" created.
```

3. Attach the **cachevol** volume to the main logical volume to begin caching:

```
# lvconvert --type writecache --cachevol <fastvol> <vg/main-lv>
```

Replace the following values:

fastvol

The name of the **cachevol** volume

vg

The volume group name

main-lv

The name of the slow logical volume

Example 12.6. Attaching the cachevol volume to the main LV

```
# lvconvert --type writecache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]?: y
Using writecache block size 4096 for unknown file system block size, logical block
size 512, physical block size 512.
WARNING: unable to detect a file system block size on vg/main-lv
WARNING: using a writecache block size larger than the file system block size may
corrupt the file system.
Use writecache block size 4096? [y/n]: y
Logical volume vg/main-lv now has writecache.
```

4. Activate the resulting logical volume:

```
# lvchange --activate y <vg/main-lv>
```

Replace the following values:

vg

The volume group name

main-lv

The name of the slow logical volume

Verification steps

- Examine the newly created devices:

```
# lvs --all --options +devices vg

LV              VG Attr       LSize   Pool          Origin          Data%  Meta%  Move Log
Cpy%Sync Convert Devices
 main-lv         vg Cwi-a-C--- 500.00m [fastvol_cvol] [main-lv_wcorig] 0.00
main-lv_wcorig(0)
 [fastvol_cvol]   vg Cwi-aoC--- 252.00m
/dev/sdc1(0)
 [main-lv_wcorig] vg owi-aoC--- 500.00m
/dev/sdb1(0)
```

Additional resources

- **lvmcache(7)** man page

12.6. DISABLING CACHING FOR A LOGICAL VOLUME

This procedure disables **dm-cache** or **dm-writecache** caching that is currently enabled on a logical volume.

Prerequisites

- Caching is enabled on a logical volume.

Procedure

1. Deactivate the logical volume:

```
# lvchange --activate n <vg>/<main-lv>
```

Replace *vg* with the volume group name, and *main-lv* with the name of the logical volume where caching is enabled.

2. Detach the **cachevol** or **cachepool** volume:

```
# lvconvert --splitcache <vg>/<main-lv>
```

Replace the following values:

Replace *vg* with the volume group name, and *main-lv* with the name of the logical volume where caching is enabled.

Example 12.7. Detaching the cachevol or cachepool volume

```
# lvconvert --splitcache vg/main-lv
Detaching writecache already clean.
Logical volume vg/main-lv writecache has been detached.
```

Verification steps

- Check that the logical volumes are no longer attached together:

```
# lvs --all --options +devices <vg>

LV      Attr       Type   Devices
fastvol -wi------- linear /dev/fast-pv
main-lv -wi------- linear /dev/slow-pv
```

Additional resources

- The **lvmcache(7)** man page

CHAPTER 13. LOGICAL VOLUME ACTIVATION

By default, when you create a logical volume, it is in an active state. A logical volume that is an active state can be used through a block device. An activated logical volume is accessible and is subject to change.

There are various circumstances, where you need to make an individual logical volume inactive and therefore unknown to the kernel. You can activate or deactivate individual logical volume with the **-a** option of the **lvchange** command.

The following is the format to deactivate an individual logical volume:

```
# lvchange -an vg/lv
```

The following is the format to activate an individual logical volume:

```
# lvchange -ay vg/lv
```

You can activate or deactivate all of the logical volumes in a volume group with the **-a** option of the **vgchange** command. This is the equivalent of running the **lvchange -a** command on each individual logical volume in the volume group.

The following is the format to deactivate all of the logical volumes in a volume group:

```
# vgchange -an vg
```

The following is the format to activate all of the logical volumes in a volume group:

```
# vgchange -ay vg
```

NOTE

During manual activation, the **systemd** automatically mounts LVM volumes with the corresponding mount point from the **/etc/fstab** file unless the **systemd-mount** unit is masked.

13.1. CONTROLLING AUTOACTIVATION OF LOGICAL VOLUMES AND VOLUME GROUPS

Autoactivation of a logical volume refers to the event-based automatic activation of a logical volume during system startup. As devices become available on the system (device online events), **systemd/udev** runs the **lvm2-pvscan** service for each device. This service runs the **pvscan --cache -aay *device*** command, which reads the named device. If the device belongs to a volume group, the **pvscan** command will check if all of the physical volumes for that volume group are present on the system. If so, the command will activate logical volumes in that volume group.

You can set the autoactivation property on a VG or LV. When the autoactivation property is disabled, the VG or LV will not be activated by a command doing autoactivation, such as **vgchange**, **lvchange**, or **pvscan** using **-aay** option. If autoactivation is disabled on a VG, no LVs will be autoactivated in that VG, and the autoactivation property has no effect. If autoactivation is enabled on a VG, autoactivation can be disabled for individual LVs.

Procedure

- You can update the autoactivation settings in one of the following ways:
  - Control autoactivation of a VG using the command line:

```
# vgchange --setautoactivation <y|n>
```

  - Control autoactivation of a LV using the command line:

```
# lvchange --setautoactivation <y|n>
```

  - Control autoactivation of a LV in the **/etc/lvm/lvm.conf** configuration file using one of the following configuration options:
    - **global/event_activation**
      When **event_activation** is disabled, **systemd/udev** will autoactivate logical volume only on whichever physical volumes are present during system startup. If all physical volumes have not appeared yet, then some logical volumes may not be autoactivated.
    - **activation/auto_activation_volume_list**
      Setting **auto_activation_volume_list** to an empty list disables autoactivation entirely. Setting **auto_activation_volume_list** to specific logical volumes and volume groups limits autoactivation to those logical volumes.

Additional resources

- **/etc/lvm/lvm.conf** configuration file
- **lvmautoactivation(7)** man page

13.2. CONTROLLING LOGICAL VOLUME ACTIVATION

You can control the activation of logical volume in the following ways:

- Through the **activation/volume_list** setting in the **/etc/lvm/conf** file. This allows you to specify which logical volumes are activated. For information about using this option, see the **/etc/lvm/lvm.conf** configuration file.
- By means of the activation skip flag for a logical volume. When this flag is set for a logical volume, the volume is skipped during normal activation commands.

Alternatively, you can use the **--setactivationskip y|n** option with the **lvcreate** or the **lvchange** commands to enable or disable the activation skip flag.

Procedure

- You can set the activation skip flag on a logical volume in the following ways:
  - To determine whether the activation skip flag is set for a logical volume run the **lvs** command, which displays the **k** attribute as in the following example:

```
# lvs vg/thin1s1
  LV      VG  Attr       LSize Pool  Origin
  thin1s1 vg  Vwi---tz-k 1.00t pool0 thin1
```

You can activate a logical volume with the **k** attribute set by using the **-K** or **--ignoreactivationskip** option in addition to the standard **-ay** or **--activate y** option.

By default, thin snapshot volumes are flagged for activation skip when they are created. You can control the default activation skip setting on new thin snapshot volumes with the **auto_set_activation_skip** setting in the **/etc/lvm/lvm.conf** file.

- The following command activates a thin snapshot logical volume that has the activation skip flag set:

```
# lvchange -ay -K VG/SnapLV
```

- The following command creates a thin snapshot without the activation skip flag:

```
# lvcreate -n SnapLV -kn -s vg/ThinLV --thinpool vg/ThinPoolLV
```

- The following command removes the activation skip flag from a snapshot logical volume:

```
# lvchange -kn VG/SnapLV
```

Verification steps

- Verify if a thin snapshot without the activation skip flag has been created:

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize    Pool       Origin Data%  Meta%  Move Log
Cpy%Sync Convert Devices             Type
  SnapLV          vg          Vwi-a-tz--  100.00m ThinPoolLV ThinLV 0.00
thin
  ThinLV          vg          Vwi-a-tz--  100.00m ThinPoolLV        0.00
thin
  ThinPoolLV      vg          twi-aotz--  100.00m                   0.00   10.94
ThinPoolLV_tdata(0) thin-pool
  [ThinPoolLV_tdata] vg        Twi-ao----  100.00m
/dev/sdc1(1)       linear
  [ThinPoolLV_tmeta] vg        ewi-ao----    4.00m
/dev/sdd1(0)       linear
  [lvol0_pmspare]    vg        ewi-------    4.00m
/dev/sdc1(0)       linear
```

13.3. ACTIVATING SHARED LOGICAL VOLUMES

You can control logical volume activation of a shared logical volume with the **-a** option of the **lvchange** and **vgchange** commands, as follows:

Command	Activation
lvchange -ay|-aey	Activate the shared logical volume in exclusive mode, allowing only a single host to activate the logical volume. If the activation fails, as would happen if the logical volume is active on another host, an error is reported.

Command	Activation
lvchange -asy	Activate the shared logical volume in shared mode, allowing multiple hosts to activate the logical volume concurrently. If the activation fails, as would happen if the logical volume is active exclusively on another host, an error is reported. If the logical type prohibits shared access, such as a snapshot, the command will report an error and fail. Logical volume types that cannot be used concurrently from multiple hosts include thin, cache, raid, and snapshot.
lvchange -an	Deactivate the logical volume.

13.4. ACTIVATING A LOGICAL VOLUME WITH MISSING DEVICES

You can control whether LVs that are missing devices can be activated by using the **lvchange** command with the **--activationmode partial|degraded|complete** option. The values are described below:

Activation Mode	Meaning
complete	Allows only logical volumes with no missing physical volumes to be activated. This is the most restrictive mode.
degraded	Allows RAID logical volumes with missing physical volumes to be activated.
partial	Allows any logical volume with missing physical volumes to be activated. This option should be used for recovery or repair only.

The default value of **activationmode** is determined by the **activationmode** setting in the **/etc/lvm/lvm.conf** file. It is used if no command line option is given.

Additional resources

- **lvmraid(7)** man page

CHAPTER 14. 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.

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.

14.1. 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?

14.2. 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.

14.2.1. LVM device filter pattern characteristics

The patterns of LVM device filter are in the form of regular expression. A regular expression delimits with a character and precedes with either **a** for acceptance, or **r** for rejection. The first regular expression in the list that matches a device determines if LVM accepts or rejects (ignores) a specific device. Then, LVM looks for the initial regular expression in the list that matches the path of a device. LVM uses this regular expression to determine whether the device should be approved with an **a** outcome or rejected with an **r** outcome.

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|.*|**.

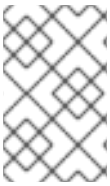
You can find the LVM devices filter configuration in the **devices/filter** and **devices/global_filter** configuration fields in the **lvm.conf** file. The **devices/filter** and **devices/global_filter** configuration fields are equivalent.

Additional resources

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

14.2.2. Examples of LVM device filter configurations

The following examples display the filter configurations to control the devices that LVM scans and uses later. To configure the device filter in the **lvm.conf** file, see Applying an LVM device filter configuration



NOTE

You may encountered duplicate Physical Volume (PV) warnings when dealing with copied or cloned PVs. You can set up filters to resolve this. See the example filter configurations in Troubleshooting duplicate physical volume warnings for multipathed LVM devices.

- 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

- **lvm.conf(5)** man page
- Applying an LVM device filter configuration
- Example LVM device filters that prevent duplicate PV warnings

14.2.3. 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 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 15. CONTROLLING LVM ALLOCATION

By default, a volume group uses the **normal** allocation policy. This allocates physical extents according to common-sense rules such as not placing parallel stripes on the same physical volume. You can specify a different allocation policy (**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.

15.1. 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.

Example 15.1. Allocating extents from one PV

In this example, **lv1** extents will be allocated from **sda**.

```
# lvcreate -n lv1 -L1G vg /dev/sda
```

Example 15.2. Allocating extents from two PVs

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
```

Example 15.3. Allocating extents without specifying PV

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
```

or

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:

- ***<segment_type>*** 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.

> **Example 15.4. Allocating raid images from two PVs**
>
> 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
> ```

> **Example 15.5. Allocating raid images from three PVs**
>
> 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
> ```

Additional resources

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

15.2. LVM ALLOCATION POLICIES

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.

15.3. 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.

Additional resources

- **pvchange(8)** man page

CHAPTER 16. GROUPING LVM OBJECTS WITH TAGS

You can assign tags to logical volume management (LVM) objects to group them. With this feature, you can automate the control of LVM behavior, such as activation, by a group. You can also use tags on LVM objects as a command.

16.1. LVM OBJECT TAGS

A logical volume management (LVM) tag is a word that is used to group LVM2 objects of the same type. You can attach tags to objects such as physical volumes, volume groups, and logical volumes , as well as 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.

16.2. LISTING LVM TAGS

The following example shows how to list LVM tags.

Procedure

- Use the following command to list all the logical volumes with the **database** tag:

```
# lvs @database
```

- Use the following command to list the currently active host tags:

```
# lvm tags
```

16.3. 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> ...
```

16.4. 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
```

16.5. DEFINING LVM HOST TAGS

This procedure describes how to define LVM host tags in a cluster configuration. You can define host tags in the configuration files.

Procedure

- Set **hosttags = 1** in the **tags** section to automatically define host tag using the machine's host name.
  This allows you to use a common configuration file which can be replicated on all your machines so they hold identical copies of the file, but the behavior can differ between machines according to the host name.

For each host tag, an extra configuration file is read if it exists: **lvm_hosttag.conf**. If that file defines new tags, then further configuration files will be appended to the list of files to read in.

For example, the following entry in the configuration file always defines **tag1**, and defines **tag2** if the host name is **host1**:

```
tags { tag1 { }  tag2 { host_list = ["host1"] } }
```

16.6. CONTROLLING LOGICAL VOLUME ACTIVATION WITH TAGS

This procedure describes how to specify in the configuration file that only certain logical volumes should be activated on that host.

Procedure

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 17. TROUBLESHOOTING LVM

You can use Logical Volume Manager (LVM) tools to troubleshoot a variety of issues in LVM volumes and groups.

17.1. 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.

Additional resources

- **lvmdump(8)** man page

17.2. 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 17.1. 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 17.2. 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 17.3. 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)
```

17.3. 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.

17.4. 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-iVI8-7qTD-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
```

17.5. 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.

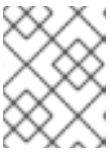
Prerequisites

- 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.

Example 17.4. Restoring a physical volume on *dev/vdb1*

The following example labels the ***/dev/vdb1*** device as a physical volume with the following properties:

- The UUID of **FmGRh3-zhok-iVI8-7qTD-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-iVI8-7qTD-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 steps

- 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)
```

17.6. 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.

17.7. 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
```

Example 17.5. Free extents in a volume group

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.

Example 17.6. Creating a logical volume by specifying the number of extents

```
# lvcreate --extents 8780 --name mylv myvg
```

Example 17.7. Creating a logical volume to occupy all the remaining space

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 steps

- 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
```

17.8. 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.

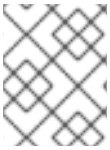
The LVM metadata area offset will match the page size of the machine that created the PV, so the metadata area can also begin 8K, 16K or 64K from the start of the disk.

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.

17.9. EXTRACTING VG METADATA FROM A DISK

Choose one of the following procedures to extract VG metadata from a disk, depending on your situation. For information about how to save extracted metadata, see Saving extracted metadata to a file.



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>
```

Example 17.8. 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>
```

Example 17.9. Locations of metadata copies in the metadata area

```
# 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>
```

Example 17.10. Copies of metadata in the metadata area 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>
```

Example 17.11. Showing description from each copy of metadata

```
# 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-5EIL-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.

17.10. 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>".

Example 17.12. Output of the command

```
# pvck --dump metadata_search --settings metadata_offset=5632 -f meta.txt /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-
Xl4i-zb6G-BYat-u53Fxv
# head -2 meta.txt
test {
id = "FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv"
```

17.11. 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 using the procedure in 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

- **pvck(8) man page**
- Extracting LVM metadata backups from a physical volume
- How to repair metadata on physical volume online?
- 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?

17.12. 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.

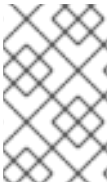
Procedure

- 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

- **pvck(8) man page**
- Extracting LVM metadata backups from a physical volume .
- How to repair metadata on physical volume online?.

- 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?.

17.13. TROUBLESHOOTING LVM RAID

You can troubleshoot various issues in LVM RAID devices to correct data errors, recover devices, or replace failed devices.

17.13.1. 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. The following attributes provide details about data coherency:

- The **raid_sync_action** field displays the current synchronization action that the RAID logical volume is performing. It can be one of the following values:

  idle

  Completed all **sync** actions (doing nothing).

  resync

  Initializing or resynchronizing an array after an unclean machine shutdown.

  recover

  Replacing a device in the array.

  check

  Looking for array inconsistencies.

  repair

  Looking for and repairing inconsistencies.

- The **raid_mismatch_count** field displays the number of discrepancies found during a **check** action.

- The **Cpy%Sync** field displays the progress of the **sync** actions.

- The **lv_attr** field provides additional indicators. Bit 9 of this field displays the health of the logical volume, and it supports the following indicators:

  m or **mismatches**

  Indicates that there are discrepancies in a RAID logical volume. You can see this character after the scrubbing operation detects the portions of the RAID, which are not coherent.

  r or **refresh**

  Indicates a failed device in a RAID array, even though LVM can read the device label and considers the device to be operational. Refresh the logical volume to notify the kernel that the device is now available, or replace the device if you suspect that it failed.

Procedure

1. Optional: Limit the I/O bandwidth that the scrubbing process uses. 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.

You can control the rate of the scrubbing operation by implementing recovery throttling. You can set the recovery rate using **--maxrecoveryrate Rate[bBsSkKmMgG]** or **--minrecoveryrate Rate[bBsSkKmMgG]** with the **lvchange --syncaction** commands. For more information, see Minimum and maximum I/O rate options .

Specify the *Rate* value as an amount per second for each device in the array. If you provide no suffix, the options assume kiB per second per 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
```

Additional resources

- **lvchange(8)** and **lvmraid(7)** man pages
- Minimum and maximum I/O rate options

17.13.2. 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)
```

Additional resources

- **lvconvert(8)** and **vgreduce(8)** man pages

17.14. 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 messages such as the following:

```
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.

17.14.1. 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 17.1. 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.

17.14.2. Cases of duplicate PV warnings

LVM displays the duplicate PV warnings in either of the following cases:

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.

17.14.3. Example LVM device filters that prevent duplicate PV warnings

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

17.14.4. Additional resources

- Limiting LVM device visibility and usage
- The LVM device filter