Red Hat Enterprise Linux 8

論理ボリュームの設定および管理

RHEL での LVM の設定および管理

Last Updated: 2024-05-08

Red Hat Enterprise Linux 8 論理ボリュームの設定および管理

RHEL での LVM の設定および管理

法律上の通知

概要

論理ボリューム管理 (LVM) は、物理ストレージ上に抽象化レイヤーを作成して、ファイルシステム、データベース、またはアプリケーションが使用できる仮想ブロックストレージデバイスである論理ストレージボリュームを作成します。物理ボリューム (PV) は、パーティションまたはディスク全体のいずれかです。これらの PV を使用すると、ボリュームグループ (VG) を作成し、利用可能なストレージから論理ボリューム (LV) 用のディスク領域のプールを作成できます。 論理ボリューム (LV) は、物理ボリュームをボリュームグループに結合することで作成できます。LV は物理ストレージを使用するよりも柔軟性があり、作成した LV は物理デバイスのパーティション再設定や再フォーマットを行わずに拡張または縮小できます。 また、LVM を使用すると、シンプロビ

ジョニングされた論理ボリューム、元のボリュームのスナップショット、RAID ボリューム、キャッシュボリューム、ストライプ化論理ボリュームの作成など、いくつかの高度な操作を実行することもできます。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 論理ボリューム管理の概要

論理ボリューム管理 (LVM) により、物理ストレージに抽象化レイヤーが作成され、論理ストレージボリュームを作成するのに役立ちます。様々な面で、物理ストレージを直接使用するよりも柔軟性が高くなります。

また、ハードウェアストレージ設定がソフトウェアから見えなくなるため、アプリケーションを停止したりファイルシステムをアンマウントしたりせずに、サイズ変更や移動が可能になります。したがって、運用コストが削減できます。

1.1. LVM のアーキテクチャー

以下は、LVM のコンポーネントです。

物理ボリューム

物理ボリューム (PV) は、LVM 使用用に指定されたパーティションまたはディスク全体です。詳細は、LVM 物理ボリュームの管理 を参照してください。

ボリュームグループ

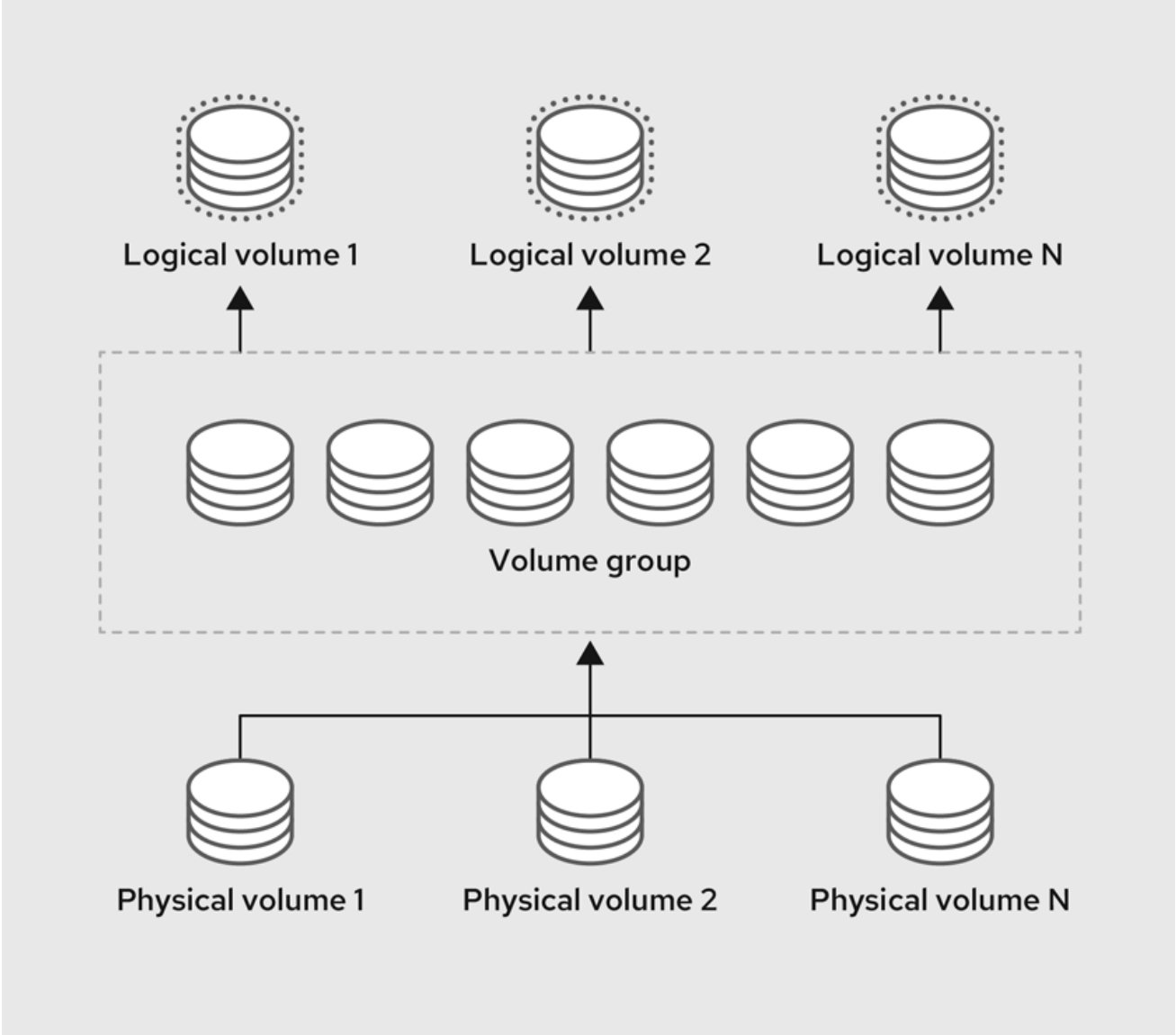
ボリュームグループ (VG) は、物理ボリューム (PV) の集合です。これにより、論理ボリュームに割り当て可能なディスク領域のプールが作成されます。詳細は、LVM ボリュームグループの管理 を参照してください。

論理ボリューム

論理ボリュームは、マウント可能なストレージデバイスを表します。詳細は、LVM 論理ボリュームの管理 を参照してください。

以下の図は、LVM のコンポーネントを示しています。

図1.1 LVM 論理ボリュームのコンポーネント



1.2. LVM の利点

物理ストレージを直接使用する場合と比較して、論理ボリュームには、以下のような利点があります。

容量の柔軟性

論理ボリュームを使用すると、ディスクとパーティションを 1 つの論理ボリュームに集約できます。この機能を使用すると、ファイルシステムを複数のデバイスにまたがって拡張でき、1 つの大きなファイルシステムとして扱うことができます。

便利なデバイスの命名

論理ストレージボリュームは、ユーザー定義のカスタマイズした名前で管理できます。

サイズ変更可能なストレージボリューム

基になるデバイスを再フォーマットしたり、パーティションを再作成したりせずに、簡単なソフトウェアコマンドを使用して論理ボリュームのサイズを拡大または縮小できます。詳細は、論理ボリュームのサイズ変更 を参照してください。

オンラインデータ移動

より新しく、高速で、耐障害性の高いストレージサブシステムをデプロイするには、**pvmove** コマンドを使用して、システムがアクティブな間にデータを移動できます。データは、ディスクが使用中の場合でもディスクに再配置できます。たとえば、ホットスワップ可能なディスクを削除する前

に空にできます。
データの移行方法の詳細は、**pvmove** man ページおよび ボリュームグループからの物理ボリュームの削除 を参照してください。

ストライプ化ボリューム

2 つ以上のデバイスにまたがってデータをストライプ化する論理ボリュームを作成できます。これにより、スループットが大幅に向上します。詳細は、ストライプ化論理ボリュームの拡張 を参照してください。

RAID ボリューム

論理ボリュームは、データの RAID を設定する際に便利な方法を提供します。これにより、デバイス障害に対する保護が可能になり、パフォーマンスが向上します。詳細は、RAID 論理ボリュームの設定 を参照してください。

ボリュームスナップショット

論理ボリュームの特定の時点のコピーであるスナップショットを作成して、一貫性のあるバックアップを作成したり、実際のデータに影響を与えずに変更の影響をテストしたりすることができます。詳細は、論理ボリュームのスナップショット を参照してください。

シンボリューム

論理ボリュームは、シンプロビジョニングにできます。これにより、利用可能な物理容量よりも大きな論理ボリュームを作成できます。詳細は、シンプロビジョニングされたボリューム (シンボリューム) の作成および管理 を参照してください。

キャッシュボリューム

キャッシュ論理ボリュームは、SSD ドライブなどの高速なブロックデバイスを使用して、大規模で低速なブロックデバイスのパフォーマンスを向上させます。詳細は、 キャッシュを有効にして論理ボリュームのパフォーマンスを改善 を参照してください。

関連情報

- LVM レポートのカスタマイズ

第2章 RHEL システムロールを使用したローカルストレージの管理

Ansible を使用して LVM およびローカルファイルシステム (FS) を管理するには、RHEL 8 で使用可能な RHEL システムロールの 1 つである **ストレージ** ロールを使用できます。

storage ロールを使用すると、ディスク上のファイルシステム、複数のマシンにある論理ボリューム、および RHEL 7.7 以降の全バージョンでのファイルシステムの管理を自動化できます。

RHEL システムロールとその適用方法の詳細は、RHEL システムロールの概要 を参照してください。

2.1. STORAGE RHEL システムロールの概要

storage ロールは以下を管理できます。

- パーティションが分割されていないディスクのファイルシステム
- 論理ボリュームとファイルシステムを含む完全な LVM ボリュームグループ
- MD RAID ボリュームとそのファイルシステム

storage ロールを使用すると、次のタスクを実行できます。

- ファイルシステムを作成する
- ファイルシステムを削除する
- ファイルシステムをマウントする
- ファイルシステムをアンマウントする
- LVM ボリュームグループを作成する
- LVM ボリュームグループを削除する
- 論理ボリュームを作成する
- 論理ボリュームを削除する
- RAID ボリュームを作成する
- RAID ボリュームを削除する
- RAID で LVM ボリュームグループを作成する
- RAID で LVM ボリュームグループを削除する
- 暗号化された LVM ボリュームグループを作成する
- RAID で LVM 論理ボリュームを作成する

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.2. STORAGE RHEL システムロールのストレージデバイスを識別するパラメーター

storage ロールの設定は、以下の変数に記載されているファイルシステム、ボリューム、およびプールにのみ影響します。

storage_volumes

マネージドのパーティションが分割されていない全ディスク上のファイルシステムのリスト
storage_volumes には **raid** ボリュームを含めることもできます。

現在、パーティションはサポートされていません。

storage_pools

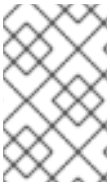
管理するプールのリスト
現在、サポートされている唯一のプールタイプは LVM です。LVM では、プールはボリュームグループ (VG) を表します。各プールの下には、ロールで管理されるボリュームのリストがあります。
LVM では、各ボリュームは、ファイルシステムを持つ論理ボリューム (LV) に対応します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.3. STORAGE RHEL システムロールを使用してブロックデバイスに XFS ファイルシステムを作成する

Ansible Playbook の例では、**storage** ロールを適用して、デフォルトのパラメーターを使用してブロックデバイス上に XFS ファイルシステムを作成します。



注記

storage ロールは、パーティションが分割されていないディスク全体または論理ボリューム (LV) でのみファイルシステムを作成できます。パーティションにファイルシステムを作成することはできません。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

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

- 現在、ボリューム名 (この例では **barefs**) は任意です。**storage** ロールは、**disks:** 属性にリスト表示されているディスクデバイスでボリュームを特定します。
- XFS は RHEL 8 のデフォルトファイルシステムであるため、**fs_type: xfs** 行を省略することができます。
- 論理ボリュームにファイルシステムを作成するには、エンクロージングボリュームグループを含む **disks:** 属性の下に LVM 設定を指定します。詳細は、ストレージ RHEL システムロールを使用した論理ボリュームの管理 を参照してください。
  LV デバイスへのパスを指定しないでください。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.4. STORAGE RHEL システムロールを使用してファイルシステムを永続的にマウントする

Ansible の例では、**storage** ロールを適用して、XFS ファイルシステムを即時かつ永続的にマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

   - この Playbook は、ファイルシステムを **/etc/fstab** ファイルに追加し、ファイルシステムを即座にマウントします。
   - **/dev/sdb** デバイス上のファイルシステム、またはマウントポイントのディレクトリーが存在しない場合は、Playbook により作成されます。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.5. STORAGE RHEL システムロールを使用して論理ボリュームを管理する

Ansible Playbook の例では、**storage** ロールを適用して、ボリュームグループに LVM 論理ボリュームを作成します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

   - **myvg** ボリュームグループは、ディスク **/dev/sda**、**/dev/sdb**、および **/dev/sdc** で構成されています。
   - **myvg** ボリュームグループがすでに存在する場合は、Playbook により論理ボリュームがボリュームグループに追加されます。
   - **myvg** ボリュームグループが存在しない場合は、Playbook により作成されます。
   - この Playbook は、**mylv** 論理ボリュームに Ext4 ファイルシステムを作成し、そのファイルシステムを **/mnt** に永続的にマウントします。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.6. STORAGE RHEL システムロールを使用してオンラインのブロック破棄を有効にする

Ansible Playbook の例では、**storage** ロールを適用して、オンラインのブロック破棄を有効にして XFS ファイルシステムをマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.7. STORAGE RHEL システムロールを使用して EXT4 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して、Ext4 ファイルシステムを作成してマウントします。

前提条件

- 制御ノードと管理ノードを準備している

- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

   - Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを **/mnt/data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.8. STORAGE RHEL システムロールを使用して EXT3 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して Ext3 ファイルシステムを作成してマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

   - Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを **/mnt/data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.9. STORAGE RHEL システムロールを使用して LVM 上の既存のファイルシステムのサイズを変更する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用して、ファイルシステムを持つ LVM 論理ボリュームのサイズを変更します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

   この Playbook は、以下の既存のファイルシステムのサイズを変更します。

   - /**opt**/**mount1** にマウントされる **mylv1** ボリュームの Ext4 ファイルシステムは、そのサイズを 10 GiB に変更します。
   - /**opt**/**mount2** にマウントされる **mylv2** ボリュームの Ext4 ファイルシステムは、そのサイズを 50 GiB に変更します。

2. Playbook の構文を検証します。

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

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.10. STORAGE RHEL システムロールを使用してスワップボリュームを作成する

本セクションでは、Ansible Playbook の例を紹介します。この Playbook は、**storage** ロールを適用し、デフォルトのパラメーターを使用して、ブロックデバイスにスワップボリュームが存在しない場合は作成し、スワップボリュームがすでに存在する場合はそれを変更します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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
```

現在、ボリューム名 (この例では **swap_fs**) は任意です。**storage** ロールは、**disks:** 属性にリスト表示されているディスクデバイスでボリュームを特定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.11. STORAGE システムロールを使用して RAID ボリュームを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform と Ansible-Core を使用して RHEL に RAID ボリュームを設定できます。要件に合わせて RAID ボリュームを設定するためのパラメーターを使用して、Ansible Playbook を作成します。

警告

特定の状況でデバイス名が変更する場合があります。たとえば、新しいディスクをシステムに追加するときなどです。したがって、データの損失を防ぐために、Playbook で特定のディスク名を使用しないでください。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.12. STORAGE RHEL システムロールを使用して RAID を備えた LVM プールを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RAID を備えた LVM プールを RHEL に設定できます。利用可能なパラメーターを使用して Ansible Playbook をセットアップし、LVM pool with RAID を設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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
```

RAID を備えた LVM プールを作成するには、**raid_level** パラメーターを使用して RAID タイプを指定する必要があります。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.13. STORAGE RHEL システムロールを使用して RAID LVM ボリュームのストライプサイズを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL の RAID LVM ボリュームのストライプサイズを設定できます。利用可能なパラメーターを使用して Ansible Playbook をセットアップし、LVM pool with RAID を設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

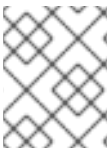
```
$ ansible-playbook ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.14. STORAGE RHEL システムロールを使用して LVM 上の VDO ボリュームを圧縮および重複排除する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用し、Virtual Data Optimizer (VDO) を使用した論理ボリューム (LVM) の圧縮と重複排除を有効にします。



注記

storage システムロールが LVM VDO を使用するため、圧縮と重複排除を使用できるのはプールごとに 1 つのボリュームのみです。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- 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
```

この例では、**compression** プールおよび **deduplication** プールを true に設定します。これは、VDO が使用されることを指定します。以下では、このパラメーターの使用方法を説明します。

- **deduplication** は、ストレージボリュームに保存されている重複データの重複排除に使用されます。
- 圧縮は、ストレージボリュームに保存されているデータを圧縮するために使用されます。これにより、より大きなストレージ容量が得られます。
- vdo_pool_size は、ボリュームがデバイスで使用する実際のサイズを指定します。VDO ボリュームの仮想サイズは、**size** パラメーターで設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.15. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する

storage ロールを使用し、Ansible Playbook を実行して、LUKS で暗号化されたボリュームを作成および設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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>
```

また、**encryption_key**、**encryption_cipher**、**encryption_key_size**、**encryption_luks** など、他の暗号化パラメーターを Playbook ファイルに追加することもできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 暗号化ステータスを表示します。

```
# 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. 作成された LUKS 暗号化ボリュームを確認します。

```
# 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]
...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- LUKS を使用したブロックデバイスの暗号化

2.16. STORAGE RHEL システムロールを使用してプールボリュームのサイズをパーセンテージで表す

このサンプル Ansible Playbook は、**storage** システムロールを適用して、論理マネージャーボリューム (LVM) のボリュームサイズをプールの合計サイズのパーセンテージで表現できるようにします。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

この例では、LVM ボリュームのサイズをプールサイズのパーセンテージで指定します (例: **60%**)。LVM ボリュームのサイズは、人間が判読できるファイルシステムのサイズ (例: **10g** または **50 GiB**) に占めるプールサイズのパーセンテージで指定することもできます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

第3章 LVM 物理ボリュームの管理

物理ボリューム (PV) は、LVM 使用用に指定されたパーティションまたはディスク全体です。LVM 論理ボリューム用にデバイスを使用する場合は、デバイスを物理ボリュームとして初期化する必要があります。

ディスクデバイス全体を物理ボリュームに使用している場合は、そのディスクにはパーティションテーブルを含めないでください。ディスクパーティションが DOS の場合は、**fdisk**、**cfdisk** などのコマンドを使用して、パーティション ID を 0x8e に設定している必要があります。ディスクデバイス全体を物理ボリュームに使用している場合は、そのディスクにはパーティションテーブルを含めないでください。既存のパーティションテーブルはすべて消去する必要があります。これにより、そのディスク上のすべてのデータが効果的に破壊されます。root として、**wipefs -a <PhysicalVolume>** コマンドを使用して、既存のパーティションテーブルを削除できます。

3.1. 物理ボリュームの概要

ブロックデバイスを物理ボリュームとして初期化すると、デバイスの先頭位置にラベルが付けられます。以下は、LVM ラベルについて説明しています。

- LVM ラベルにより物理デバイスの正しい識別とデバイスの順序付けが行われます。ラベルが付けられていない、LVM 以外のデバイスは、起動時にシステムが検出した順序に応じて、再起動後に名前が変更される場合があります。LVM ラベルは、再起動してもクラスター全体で維持されます。

- LVM ラベルは、デバイスを LVM 物理ボリュームとして識別するものです。これには、物理ボリューム用のランダムな一意識別子 (UUID) が含まれます。また、ブロックデバイスのサイズもバイト単位で保存し、LVM メタデータがデバイスのどこに保存されているかも記録します。

- LVM ラベルは、デフォルトでは 2 番目の 512 バイトセクターに配置されます。物理ボリュームを作成する場合は、先頭の 4 つのセクターのいずれかにラベルを配置することにより、このデフォルト設定を書き換えることができます。これにより、必要に応じて LVM ボリュームを、このセクターを利用する他のユーザーと併用できるようになります。

以下は、LVM メタデータについて説明しています。

- LVM メタデータには、システムにある LVM ボリュームグループの設定詳細が含まれています。デフォルトでは、メタデータの複製コピーが、ボリュームグループ内で、すべての物理ボリュームの、すべてのメタデータ領域に保存されています。LVM メタデータのサイズは小さく、ASCII 形式が使用されます。

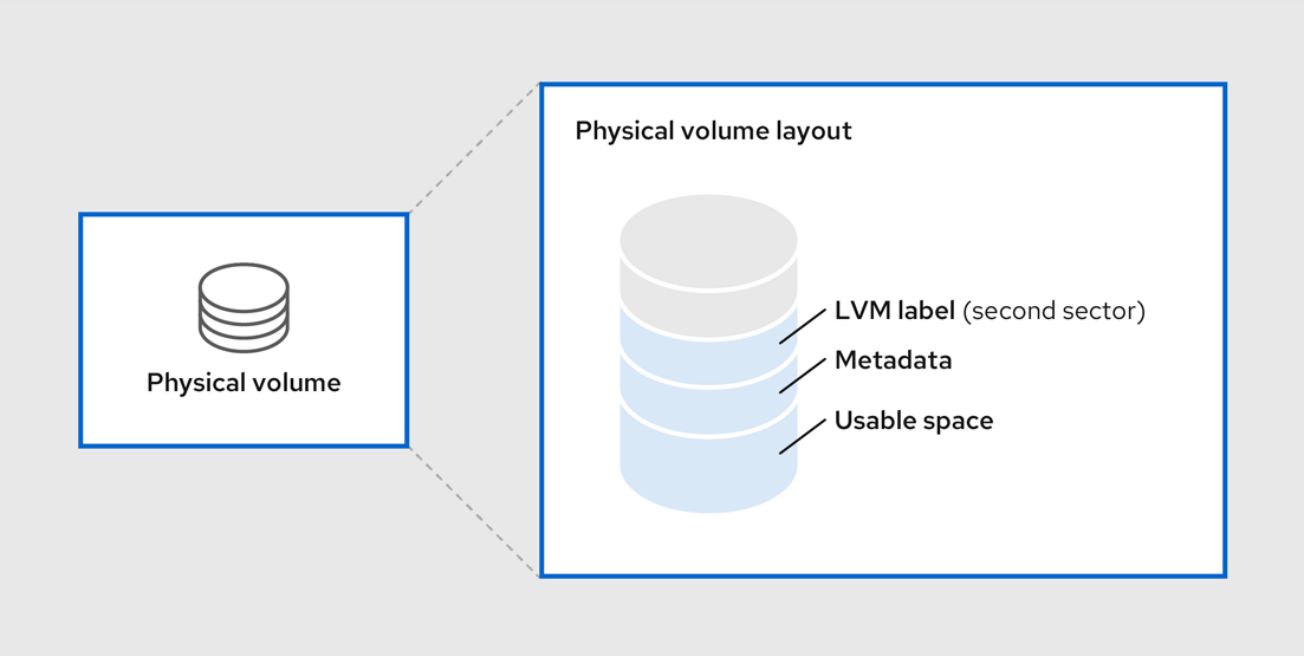
- 現在、LVM では、各物理ボリュームにメタデータのコピーを 1 つまたは 2 つ保存できます。コピーをゼロにすることもできます。デフォルトでは 1 つ保存されます。物理ボリューム上に保存するメタデータのコピー数を一度設定したら、その数を後で変更することはできません。最初のコピーはデバイスの先頭にあるラベルの後に保存されます。2 つ目のコピーがある場合は、デバイスの最後に配置されます。意図したものとは別のディスクに誤って書き込みを行い、ディスクの先頭領域を上書きしてしまった場合でも、デバイス後部にある 2 つ目のコピーでメタデータを復元できます。

次の図は、LVM 物理ボリュームのレイアウトを示しています。LVM ラベルが 2 番目のセクターにあり、その後にメタデータ領域、使用可能なデバイス領域と続きます。

注記

Linux カーネルおよび本書では、セクターのサイズを 512 バイトとしています。

図3.1 物理ボリュームのレイアウト



関連情報

- ディスク上の複数パーティション

3.2. ディスク上の複数パーティション

LVM を使用して、ディスクパーティションから物理ボリューム (PV) を作成できます。

Red Hat では、以下のような理由により、ディスク全体に対応するパーティションを 1 つ作成し、1 つの LVM 物理ボリュームとしてラベルを付けることを推奨しています。

管理上の利便性

各ディスクが一度だけ表示されると、システムのハードウェアの追跡が簡単になります。これは、特にディスクに障害が発生した場合に役に立ちます。

ストライピングのパフォーマンス

LVM は、2 つの物理ボリュームが同じ物理ディスクにあるかどうかを認識しません。2 つの物理ボリュームが同じ物理ディスクにあるときに、ストライプ化された論理ボリュームを作成すると、作成されたボリュームのディスクは同じでも、パーティションは異なる可能性があります。このとき、パフォーマンスは、改善ではなく低下します。

RAID の冗長性

LVM は、2 つの物理ボリュームが同じデバイスにあるかどうかを判断できません。2 つの物理ボリュームが同じデバイスにある場合に RAID 論理ボリュームを作成すると、パフォーマンスとフォールトトレランスが失われる可能性があります。

1 つのディスクを、複数の LVM 物理ボリュームに分割しないといけない場合があります (推奨はされません)。たとえば、ディスクがほとんどないシステムで、既存システムを LVM ボリュームに移行する場合に、パーティション間でデータを移動しなければならない場合があります。さらに、大容量のディスクが存在し、管理目的で複数のボリュームグループを必要とする場合は、そのディスクにパーティションを設定する必要があります。ディスクに複数のパーティションがあり、そのパーティションがいずれも同じボリュームグループにある場合に、ボリュームを作成するときは、論理ボリュームに追加するパーティションを注意して指定してください。

LVM は、パーティション化していないディスクを物理ボリュームとして使用することをサポートしま

すが、パーティションなしで PV を作成すると、混合オペレーティングシステム環境で問題が発生する可能性があるため、ディスク全体を単一のパーティションとして作成することが推奨されます。他のオペレーティングシステムはデバイスを空き状態として解釈し、ドライブの先頭にある PV ラベルを上書きする可能性があります。

3.3. LVM 物理ボリュームの作成

この手順では、LVM 物理ボリューム (PV) を作成し、ラベルを付ける方法を説明します。

この手順では、**/dev/vdb1**、**/dev/vdb2**、および **/dev/vdb3** を、システムで利用可能なストレージデバイスに置き換えます。

前提条件

- **lvm2** パッケージがインストールされている。

手順

1. **pvcreate** コマンドに、スペースで区切られたデバイス名を引数として使用して、複数の物理ボリュームを作成します。

```
# 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.
```

   これにより、ラベルが **/dev/vdb1**、**/dev/vdb2**、および **/dev/vdb3** に配置され、それらが LVM に属する物理ボリュームとしてマークされます。

2. 要件に応じて、以下のコマンドのいずれかを使用して、作成した物理ボリュームを表示します。

   a. **pvdisplay** コマンド: 各物理ボリュームの詳細をそれぞれ複数行出力します。物理プロパティー (サイズ、エクステント、ボリュームグループなど) および他のオプションが、決められた形式で表示されます。

```
# 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. **pvs** コマンド: 物理ボリュームの情報を設定可能な形式で出力します。

```
# 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. **pvscan** コマンド: システムにある物理ボリュームで対応している LVM ブロックデバイスをすべてスキャンします。このコマンドで、特定の物理ボリュームがスキャンされないように、**lvm.conf** ファイルでフィルターを定義することができます。

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

関連情報

- **pvcreate (8)**、**pvdisplay (8)**、**pvs (8)**、**pvscan (8)**、および **lvm (8)** の man ページ

3.4. LVM 物理ボリュームの削除

デバイスを LVM で使用する必要がなくなった場合、**pvremove** コマンドを使用して LVM ラベルを削除できます。**pvremove** コマンドを実行すると、空の物理ボリュームにある LVM メタデータをゼロにします。

手順

1. 物理ボリュームを削除します。

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

2. 既存の物理ボリュームを表示し、必要なボリュームが削除されているかどうかを確認します。

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

削除する物理ボリュームがボリュームグループの一部になっている場合は、**vgreduce** コマンドで、ボリュームグループから物理ボリュームを取り除く必要があります。詳細は、ボリュームグループからの物理ボリュームの削除 を参照してください。

関連情報

- **pvremove(8)** の man ページ

3.5. 関連情報

- parted でディスクにパーティションテーブルを作成
- **parted(8)** man ページ

第4章 LVM ボリュームグループの管理

ボリュームグループ (VG) は、物理ボリューム (PV) の集合です。これにより、論理ボリューム (LV) に割り当て可能なディスク領域のプールが作成されます。

ボリュームグループ内で、割り当て可能なディスク領域は、エクステントと呼ばれる固定サイズの単位に分割されます。割り当て可能な領域の最小単位は、1 エクステントです。エクステントは、物理ボリュームでは物理エクステントと呼ばれます。

論理ボリュームには、物理エクステントと同じサイズの論理エクステントが割り当てられます。そのため、エクステントのサイズは、ボリュームグループ内のすべての論理ボリュームで同じになります。ボリュームグループは、論理エクステントを物理エクステントにマッピングします。

4.1. LVM ボリュームグループの作成

/dev/vdb1 および **/dev/vdb2** 物理ボリューム (PV) を使用して、LVM ボリュームグループ (VG) myvg を作成できます。デフォルトでは、物理ボリュームを使用してボリュームグループを作成すると、そのディスク領域は 4 MB のエクステントに分割されます。このエクステントサイズは、論理ボリュームのサイズを増減する際の最小単位です。エクステントサイズは、**vgcreate** コマンドの **-s** 引数を使用して変更できます。エクステントの数が多くても、論理ボリュームの I/O パフォーマンスに影響を与えることはありません。**vgcreate** コマンドに **-p** 引数と **-l** 引数を使用すると、ボリュームグループに追加可能な物理ボリュームまたは論理ボリュームの数に制限をかけることができます。

前提条件

- **lvm2** パッケージがインストールされている。
- 物理ボリュームが作成されます。物理ボリュームの作成方法は、LVM 物理ボリュームの作成 を参照してください。

手順

1. 次のいずれかの方法を使用して、**myvg** VG を作成します。
   - オプションを指定しない場合:

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

   - **-s** 引数を使用してボリュームグループのエクステントサイズを指定する方法:

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

   - **-p** および **-l** 引数を使用して、VG に追加可能な物理ボリュームまたは論理ボリュームの数に制限をかける方法:

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

2. 要件に応じて、以下のコマンドのいずれかを使用して、作成したボリュームグループを表示します。

- **vgs** コマンド: ボリュームグループの情報を設定可能な形式で提供し、1 ボリュームグループにつき 1 行ずつ表示します。

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

- **vgdisplay** コマンド: 決められた形式でボリュームグループのプロパティー (サイズ、エクステント、物理ボリュームの数など) およびその他のオプションを表示します。以下の例は、ボリュームグループ **myvg** に関する **vgdisplay** コマンドの出力を示しています。既存のすべてのボリュームグループを表示するには、ボリュームグループを指定しないでください。

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

- **vgscan** コマンド: ボリュームグループ用に、システムにあるサポートされるすべての LVM ブロックデバイスをスキャンします。

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

3. オプション: 空き物理ボリュームを 1 つまたは複数追加して、ボリュームグループの容量を増やします。

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

4. オプション: 既存のボリュームグループの名前を変更します。

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

関連情報

- **vgcreate (8)**、**vgextend (8)**、**vgdisplay (8)**、**vgs (8)**、**vgscan (8)**、**vgrename (8)**、および **lvm (8)** の man ページ

4.2. LVM ボリュームグループの統合

2 つのボリュームグループを統合して 1 つのボリュームグループにするには、**vgmerge** コマンドを使用します。ボリュームの物理エクステントサイズが同じで、かつ両ボリュームグループの物理ボリュームおよび論理ボリュームのサマリーがマージ先ボリュームグループの制限内に収まる場合は、非アクティブなマージ元のボリュームを、アクティブまたは非アクティブのマージ先ボリュームにマージができます。

手順

- 非アクティブなボリュームグループ **databases** をアクティブまたは非アクティブなボリュームグループ **myvg** にマージして、詳細なランタイム情報を提供します。

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

関連情報

- **vgmerge(8)** の man ページ

4.3. ボリュームグループからの物理ボリュームの削除

ボリュームグループ (VG) から未使用の物理ボリューム (PV) を削除するには、**vgreduce** コマンドを使用します。**vgreduce** コマンドは、空の物理ボリュームを 1 つまたは複数削除して、ボリュームグループの容量を縮小します。これにより、物理ボリュームが解放され、異なるボリュームグループで使用したり、システムから削除できるようになります。

手順

1. 物理ボリュームがまだ使用中の場合は、データを同じボリュームグループから別の物理ボリュームに移行します。

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

2. 既存のボリュームグループ内の他の物理ボリュームに空きエクステントが十分にない場合は、以下を行います。

   a. **/dev/vdb4** から、物理ボリュームを新規作成します。

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

   b. 新規作成した物理ボリュームを **myvg** ボリュームグループに追加します。

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

   c. データを **/dev/vdb3** から **/dev/vdb4** に移動します。

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

3. ボリュームグループから物理ボリューム **/dev/vdb3** を削除します。

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

検証

- /dev/vdb3 物理ボリュームが myvg ボリュームグループから削除されていることを確認します。

```
# 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
```

関連情報

- **vgreduce(8)**、**pvmove(8)**、および **pvs(8)** の man ページ

4.4. LVM ボリュームグループの分割

この物理ボリュームに未使用領域が十分にあれば、新たにディスクを追加しなくてもボリュームグループを作成できます。

初期設定では、ボリュームグループ myvg は/dev/vdb1、/dev/vdb2、および /dev/vdb3 で設定されます。この手順を完了すると、ボリュームグループ myvg は /dev/vdb1 および /dev/vdb2 で設定され、2 番目のボリュームグループ yourvg は /dev/vdb3 で設定されます。

前提条件

- ボリュームグループに十分な空き領域がある。**vgscan** コマンドを使用すると、現在ボリュームグループで利用可能な空き領域の容量を確認できます。
- 既存の物理ボリュームの空き容量に応じて、**pvmove** コマンドを使用して、使用されている物理エクステントをすべて他の物理ボリュームに移動します。詳細は、ボリュームグループからの物理ボリュームの削除 を参照してください。

手順

1. 既存のボリュームグループ myvg を新しいボリュームグループ yourvg に分割します。

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

注記

既存のボリュームグループを使用して論理ボリュームを作成した場合は、次のコマンドを実行して論理ボリュームを非アクティブにします。

```
# lvchange -a n /dev/myvg/mylv
```

論理ボリュームを作成する方法は、LVM 論理ボリュームの管理 を参照してください。

2. 2 つのボリュームグループの属性を表示します。

```
# 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
```

検証

- 新規作成したボリュームグループ **yourvg** が、**/dev/vdb3** 物理ボリュームで設定されていることを確認します。

```
# 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
```

関連情報

- **vgsplit(8)**、**vgs(8)**、および **pvs(8)** の man ページ

4.5. ボリュームグループを別のシステムへ移動

次のコマンドを使用して、LVM ボリュームグループ (VG) 全体を別のシステムに移動できます。

vgexport

既存のシステムでこのコマンドを使用して、システムから非アクティブな VG にアクセスできないようにします。VG にアクセスできなくなったら、その物理ボリューム (PV) の接続を解除できます。

vgimport

他のシステムでこのコマンドを使用して、新しいシステムで、古いシステムで非アクティブだった VG にアクセスできるようにします。

前提条件

- 移動するボリュームグループ内のアクティブなボリュームのファイルにアクセスしているユーザーがいない。

手順

1. **mylv** 論理ボリュームをアンマウントします。

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

2. ボリュームグループ内のすべての論理ボリュームを非アクティブ化します。これにより、ボリュームグループでこれ以上の動作が発生しないようにします。

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

3. ボリュームグループをエクスポートして、削除元のシステムがボリュームグループにアクセスできないようにします。

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

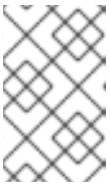
4. エクスポートされたボリュームグループを表示します。

```
# 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. システムをシャットダウンし、ボリュームグループを構成するディスクを取り外し、新しいシステムに接続します。

6. ディスクを新しいシステムに接続し、ボリュームグループをインポートして、新しいシステムからアクセスできるようにします。

```
# vgimport myvg
```



注記

vgimport コマンドの **--force** 引数を使用すると、物理ボリュームがないボリュームグループをインポートし、その後 **vgreduce --removemissing** コマンドを実行できます。

7. ボリュームグループをアクティブ化します。

```
# vgchange -ay myvg
```

8. ファイルシステムをマウントして使用できるようにします。

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

関連情報

- **vgimport(8)**、**vgexport(8)**、および **vgchange(8)** man ページ

4.6. LVM ボリュームグループの削除

vgremove コマンドを使用して、既存のボリュームグループを削除できます。

前提条件

- ボリュームグループには論理ボリュームがありません。ボリュームグループから論理ボリュームを削除するには、LVM 論理ボリュームの削除 を参照してください。

手順

1. クラスター環境にボリュームグループが存在する場合は、その他のすべてのノードで、ボリュームグループのロックスペースを停止します。削除を実行しているノードを除くすべてのノードで次のコマンドを使用します。

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

ロックが停止するのを待ちます。

2. ボリュームグループを削除します。

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

関連情報

- **vgremove(8)** の man ページ

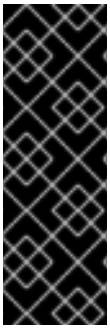
第5章 LVM 論理ボリュームの管理

論理ボリュームは、ファイルシステム、データベース、またはアプリケーションが使用できる仮想のブロックストレージデバイスです。LVM 論理ボリュームを作成する場合は、物理ボリューム (PV) をボリュームグループ (Volume Group: VG) に統合します。これによりディスク領域のプールが作成され、そこから LVM 論理ボリューム (Logical Volume: LV) を割り当てます。

5.1. 論理ボリュームの概要

管理者は、標準のディスクパーティションとは異なり、データを破棄せずに論理ボリュームを拡大または縮小できます。ボリュームグループの物理ボリュームが別のドライブまたは RAID アレイにある場合は、ストレージデバイスに論理ボリュームを分散することもできます。

論理ボリュームを、ボリュームに必要なデータよりも小さい容量に縮小すると、データが失われる可能性があります。さらに、ファイルシステムの中には縮小できないものもあります。柔軟性を最大限にするために、現在のニーズに合わせて論理ボリュームを作成し、過剰なストレージ容量を未割り当ての状態にします。必要に応じて、未割り当ての領域を使用するように、論理ボリュームを安全に拡張できます。



重要

AMD システム、Intel システム、ARM システム、および IBM Power Systems サーバーで、ブートローダーは LVM ボリュームを読み取ることができません。このため、**/boot** パーティションは、LVM ではなく標準のパーティションで作成してください。IBM Z の場合は、**zipl** ブートローダーによりリニアマッピングを使用した LVM 論理ボリューム上の **/boot** に対応しています。デフォルトのインストールプロセスでは、/ パーティションと swap パーティションは常に LVM ボリューム内に、**/boot** パーティションは別途、物理ボリューム上に作成されます。

以下は、論理ボリュームの種類になります。

リニアボリューム

リニアボリュームは、複数の物理ボリュームの領域を 1 つの論理ボリュームに統合します。たとえば、60GB ディスクが 2 つある場合は、120GB の論理ボリュームを作成できます。物理ストレージは連結されます。

ストライプ化論理ボリューム

LVM 論理ボリュームにデータを書き込む際に、ファイルシステムは、基になる物理ボリューム全体にデータを分配します。このとき、ストライプ化論理ボリュームを作成すると、データを物理ボリュームに書き込む方法を制御できます。順次の読み取りおよび書き込みが大量に行われる場合には、これによりデータ I/O の効率を向上できます。

ストライピングは、ラウンドロビン式で、指定した数の物理ボリュームにデータを書き込んでいくことで、パフォーマンスを向上させます。I/O は、ストライピングでは並行して実行されます。これにより、ストライプで追加される各物理ボリュームでは、ほぼ直線的なパフォーマンスの向上が期待できます。

RAID 論理ボリューム

LVM は、RAID レベル 0、1、4、5、6、10 に対応します。RAID 論理ボリュームはクラスターには対応していません。RAID 論理ボリュームを作成するとき、LVM は、データまたはアレイ内のパリティーサブボリュームごとに、サイズが 1 エクステントのメタデータサブボリュームを作成します。

シンプロビジョニングされた論理ボリューム (シンボリューム)

シンプロビジョニングされた論理ボリュームを使用すると、利用可能な物理ストレージよりも大きな論理ボリュームを作成できます。シンプロビジョニングされたボリュームセットを作成すると、

システムは要求されるストレージの全量を割り当てる代わりに、実際に使用する容量を割り当てることができます。

スナップショットボリューム

LVM スナップショット機能により、サービスを中断せずに任意の時点でデバイスの仮想イメージを作成できます。スナップショットの取得後に作成元のデバイスに変更が加えられると、データが変更する前に、これから変更する部分のコピーがスナップショット機能により作成されるため、このコピーを使用して、デバイスの状態を再構築できます。

シンプロビジョニングされたスナップショットボリューム

シンプロビジョニングされたスナップショットボリュームを使用すると、同じデータボリュームにより多くの仮想デバイスを格納できます。シンプロビジョニングされたスナップショットは、特定のタイミングでキャプチャーする際にすべてのデータをコピーしていないため、便利です。

キャッシュボリューム

LVM は、高速ブロックデバイス (SSD ドライブなど) を、大規模で低速なブロックデバイスのライトバックまたはライトスルーのキャッシュとして使用することに対応します。既存の論理ボリュームのパフォーマンスを改善するためにキャッシュ論理ボリュームを作成したり、大規模で低速なデバイスと共に小規模で高速なデバイスで設定される新規のキャッシュ論理ボリュームを作成したりできます。

5.2. LVM 論理ボリュームの作成

前提条件

- **lvm2** パッケージがインストールされている。
- ボリュームグループが作成されます。詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. 論理ボリュームを作成します。

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

-n オプションを使用して LV 名を **mylv** に設定し、**-L** オプションを使用して、Mb 単位で LV のサイズを設定しますが、他の単位を使用することもできます。デフォルトでは、論理ボリュームのタイプはリニアですが、**--type** オプションを使用して必要なタイプを指定できます。

重要

ボリュームグループに要求されるサイズとタイプの空き物理エクステントが十分にない場合、このコマンドは失敗します。

2. 要件に応じて、以下のコマンドのいずれかを使用して、作成した論理ボリュームを表示します。
   a. **lvs** コマンド: 論理ボリューム情報を設定可能な形式で提供して、1 つの論理ボリュームにつき 1 行ずつ表示します。

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

b. **lvdisplay** コマンド: 決められた形式で、論理ボリュームのプロパティー (サイズ、レイアウト、マッピングなど) を表示します。

```
# lvdisplay -v /dev/myvg/mylv
  --- Logical volume ---
  LV Path                /dev/myvg/mylv
  LV Name                mylv
  VG Name                myvg
  LV UUID                YTnAk6-kMlT-c4pG-HBFZ-Bx7t-ePMk-7YjhaM
  LV Write Access        read/write
[..]
```

c. **lvscan** コマンド: システム内のすべての論理ボリュームをスキャンし、それらをリスト表示します。

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

3. 論理ボリュームにファイルシステムを作成します。以下のコマンドを使用すると、論理ボリュームに **xfs** ファイルシステムが作成されます。

```
# 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. 論理ボリュームをマウントして、ファイルシステムのディスクの領域使用率を報告します。

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

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

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

関連情報

- **lvcreate(8)**、**lvdisplay(8)**、**lvs(8)**、**lvscan(8)**、**lvm(8)**、および **mkfs.xfs(8)** の man ページ

5.3. RAID0 ストライピング 論理ボリュームの作成

RAID0 論理ボリュームは、論理ボリュームデータをストライプサイズ単位で複数のデータサブボリューム全体に分散します。以下の手順では、ディスク間でデータをストライピングする **mylv** という LVM RAID0 論理ボリュームを作成します。

前提条件

1. 3 つ以上の物理ボリュームを作成している。物理ボリュームの作成方法は、LVM 物理ボリュームの作成 を参照してください。
2. ボリュームグループを作成している。詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. 既存のボリュームグループから RAID0 論理ボリュームを作成します。次のコマンドは、ボリュームグループ **myvg** から RAID0 ボリューム **mylv** を作成します。これは、サイズが **2G** で、ストライプが 3 つ、ストライプ サイズが **4kB** です。

```
# 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. RAID0 論理ボリュームにファイルシステムを作成します。以下のコマンドを使用すると、論理ボリュームに ext4 ファイルシステムが作成されます。

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

3. 論理ボリュームをマウントして、ファイルシステムのディスクの領域使用率を報告します。

```
# 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
```

検証

- 作成された RAID0 ストライピング論理ボリュームを表示します。

```
# 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. LVM 論理ボリュームの名前の変更

この手順では、既存の論理ボリューム **mylv** の名前を **mylv1** に変更する方法を説明します。

手順

1. 論理ボリュームが現在マウントされている場合は、ボリュームをアンマウントします。

```
# umount /mnt
```

/mnt は、マウントポイントに置き換えます。

2. 既存の論理ボリュームの名前を変更します。

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

また、デバイスへの完全パスを指定して、論理ボリュームの名前を変更することもできます。

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

関連情報

- **lvrename(8)** の man ページ

5.5. 論理ボリュームからのディスクの削除

この手順では、ディスクを交換するか、別のボリュームで使用するために、既存の論理ボリュームからディスクを削除する方法を説明します。

ディスクを削除する前に、LVM 物理ボリュームのエクステントを、別のディスクまたはディスクセットに移動する必要があります。

手順

1. LV を使用する際に、物理ボリュームの使用済み容量と空き容量を表示します。

```
# 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. データを他の物理ボリュームに移動します。

   a. 既存のボリュームグループ内の他の物理ボリュームに空きエクステントが十分にある場合は、以下のコマンドを使用してデータを移動します。

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

   b. 既存のボリュームグループ内の他の物理ボリュームに空きエクステントが十分にない場合は、以下のコマンドを使用して新しい物理ボリュームを追加し、新たに作成した物理ボリュームを使用してボリュームグループを拡張し、この物理ボリュームにデータを移動します。

```
# 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. 物理ボリュームを削除します。

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

論理ボリュームに、障害のある物理ボリュームが含まれる場合は、その論理ボリュームを使用することはできません。見つからない物理ボリュームをボリュームグループから削除します。その物理ボリュームに論理ボリュームが割り当てられていない場合は、**vgreduce** コマンドの **--removemissing** パラメーターを使用できます。

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

関連情報

- **pvmove(8)**、**vgextend(8)**、**vereduce(8)**、および **pvs(8)** の man ページ

5.6. LVM 論理ボリュームの削除

この手順では、既存の論理ボリューム **/dev/myvg/mylv1** をボリュームグループ **myvg** から削除する方法を説明します。

手順

1. 論理ボリュームが現在マウントされている場合は、ボリュームをアンマウントします。

```
# umount /mnt
```

2. クラスター環境に論理ボリュームが存在する場合は、アクティブになっているすべてのノードで、論理ボリュームを非アクティブにします。アクティブになっている各ノードで、次のコマンドを実行します。

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

3. **lvremove** ユーティリティーを使用して、論理ボリュームを削除します。

```
# lvremove /dev/myvg/mylv1

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

注記

この場合は、論理ボリュームが非アクティブになっていません。削除する前に論理ボリュームを明示的に非アクティブにした場合は、アクティブな論理ボリュームを削除するかどうかを確認するプロンプトが表示されません。

関連情報

- **lvremove(8)** の man ページ

5.7. RHEL システムロールを使用して LVM 論理ボリュームを管理する

storage ロールを使用して、次のタスクを実行します。

- 複数のディスクで設定されるボリュームグループに LVM 論理ボリュームを作成します。
- 論理ボリューム上に特定のラベルを付けて ext4 ファイルシステムを作成します。
- ext4 ファイルシステムを永続的にマウントします。

前提条件

- **storage** ロールを含む Ansible Playbook がある。

5.7.1. storage RHEL システムロールを使用して論理ボリュームを管理する

Ansible Playbook の例では、**storage** ロールを適用して、ボリュームグループに LVM 論理ボリュームを作成します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

   - **myvg** ボリュームグループは、ディスク **/dev/sda**、**/dev/sdb**、および **/dev/sdc** で構成されています。
   - **myvg** ボリュームグループがすでに存在する場合は、Playbook により論理ボリュームがボリュームグループに追加されます。

- **myvg** ボリュームグループが存在しない場合は、Playbook により作成されます。
- この Playbook は、**mylv** 論理ボリュームに Ext4 ファイルシステムを作成し、そのファイルシステムを **/mnt** に永続的にマウントします。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

5.7.2. 関連情報

- **ストレージ** ロールの詳細は、RHEL システムロールを使用したローカルストレージの管理 を参照してください。

5.8. LVM ボリュームグループの削除

vgremove コマンドを使用して、既存のボリュームグループを削除できます。

前提条件

- ボリュームグループには論理ボリュームがありません。ボリュームグループから論理ボリュームを削除するには、LVM 論理ボリュームの削除 を参照してください。

手順

1. クラスター環境にボリュームグループが存在する場合は、その他のすべてのノードで、ボリュームグループのロックスペースを停止します。削除を実行しているノードを除くすべてのノードで次のコマンドを使用します。

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

ロックが停止するのを待ちます。

2. ボリュームグループを削除します。

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

関連情報

- **vgremove(8)** の man ページ

第6章 論理ボリュームのサイズ変更

論理ボリュームを作成したら、ボリュームのサイズを変更できます。

6.1. 論理ボリュームとファイルシステムの拡張

lvextend コマンドを使用して、論理ボリューム (LV) を拡張できます。追加する LV の容量、または拡張後の LV のサイズを指定できます。LV とともに基礎となるファイルシステムを拡張するには、**lvextend** コマンドの **-r** オプションを使用します。

警告

lvresize コマンドを使用して論理ボリュームを拡張することもできますが、このコマンドでは、誤って縮小されない保証はありません。

前提条件

- ファイルシステムを持つ既存の論理ボリューム (LV) がある。**df -Th** コマンドを使用して、ファイルシステムのタイプとサイズを確認します。論理ボリュームおよびファイルシステムの作成に関する詳細は、LVM 論理ボリュームの作成 を参照してください。
- LV およびファイルシステムを拡張するのに十分な領域がボリュームグループにある。**vgs -o name,vgfree** コマンドを使用して、利用可能な領域を確認します。ボリュームグループの作成の詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. オプション: ボリュームグループに LV を拡張するのに十分な領域がない場合は、ボリュームグループに新しい物理ボリュームを追加します。

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

2. LV とファイルシステムを拡張します。

注記

-r 引数を指定せずに **lvextend** コマンドを使用すると、LV のみが拡張されます。基礎となる XFS ファイルシステムを拡張するには、XFS ファイルシステムのサイズの拡大 を参照してください。GFS2 ファイルシステムの場合は、GFS2 ファイルシステムの拡張 を、ext4 ファイルシステムの場合は、ext4 ファイルシステムのサイズ変更 を参照してください。

注記

-L オプションを使用して、LV を新しいサイズに拡張します。**-l** オプションを使用して、増やす論理ボリュームのサイズに応じてエクステントの数を指定します。

```
# 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.
```

また、**mylv** 論理ボリュームを拡張して、**myvg** ボリュームグループの未割り当て領域をすべて埋めることもできます。

```
# 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.
```

検証

- ファイルシステムと LV が拡張されたことを確認します。

```
# 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
```

関連情報

- **vgextend(8)**、**lvextend(8)**、および **xfs_growfs(8)** の man ページ

6.2. 論理ボリュームとファイルシステムの縮小

lvreduce コマンドと **resizefs** オプションを使用して、論理ボリュームとそのファイルシステムを縮小できます。

縮小する論理ボリュームにファイルシステムが含まれている場合は、データの損失を防ぐため、ファイルシステムが、縮小する論理ボリュームにある領域を使用しないようにしてください。そのため、論理ボリュームにファイルシステムが含まれている場合は、**lvreduce** コマンドの **--resizefs** オプションを使用してください。

--resizefs を使用すると、**lvreduce** は論理ボリュームを縮小する前にファイルシステムを縮小しようとします。ファイルシステムがいっぱいであるか、縮小をサポートしていないためにファイルシステムの縮小が失敗した場合、**lvreduce** コマンドは失敗し、論理ボリュームの縮小は試行されません。

警告

ほとんどの場合、**lvreduce** コマンドはデータ損失の可能性を警告し、確認を要求します。しかし、論理ボリュームが非アクティブな状態であったり、**--resizefs** オプションが使用されなかった場合など、警告が表示されない場合があるため、データの損失を防ぐのに確認プロンプトのみを信頼しないようにしてください。

lvreduce コマンドの **--test** オプションを使用しても、このオプションはファイルシステムのチェックやファイルシステムのサイズ変更のテストを行わないため、操作が安全かどうかは示されないことに注意してください。

前提条件

- 論理ボリュームのファイルシステムが縮小をサポートしている。**df -Th** コマンドを使用して、ファイルシステムのタイプとサイズを確認します。

注記

たとえば、GFS2 および XFS ファイルシステムは縮小をサポートしていません。

- 基礎となるファイルシステムが、削減する LV 内の領域を使用していない。

手順

1. 次のオプションのいずれかを使用して、**myvg** ボリュームグループ内の **mylv** 論理ボリュームとそのファイルシステムを縮小します。
   - LV とそのファイルシステムを希望の値まで縮小します。

```
# 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.
```

   - 論理ボリュームとファイルシステムを 64 メガバイト分縮小します。

```
# 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
```

関連情報

- **lvreduce(8)** の man ページ

6.3. ストライプ化論理ボリュームの拡張

lvextend コマンドを使用して必要なサイズを指定すると、ストライプ化論理ボリューム (LV) を拡張できます。

前提条件

1. ボリュームグループ (VG) を構成する基礎となる物理ボリューム (PV) に、ストライプをサポートするのに十分な空き領域がある。

手順

1. **オプション:** ボリュームグループを表示します。

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

2. **オプション:** ボリュームグループの全領域を使用して、ストライプを作成します。

```
# 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. **オプション:** 新しい物理ボリュームを追加して、**myvg** ボリュームグループを拡張します。

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

この手順を繰り返して、ストライプのタイプと使用する領域の量に応じて、十分な物理ボリュームを追加します。たとえば、ボリュームグループ全体を使用する双方向ストライプの場合、少なくとも 2 つの物理ボリュームを追加する必要があります。

4. **myvg** VG の一部であるストライプ論理ボリューム **stripe1** を拡張します。

```
# 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
```

また、**stripe1** 論理ボリュームを拡張して、**myvg** ボリュームグループの未割り当て領域をすべて埋めることもできます。

```
# 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.
```

検証

- 拡張したストライプ LV の新しいサイズを確認します。

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

第7章 LVM レポートのカスタマイズ

LVM では、カスタマイズされたレポートを生成したり、レポートの出力をフィルタリングしたりするための様々な設定およびコマンドラインオプションが提供されます。出力のソート、単位の指定、選択基準の使用、および **lvm.conf** ファイルの更新を行って LVM レポートをカスタマイズできます。

7.1. LVM 表示の形式の制御

pvs、**lvs**、**vgs** のいずれのコマンドも、表示するデフォルトのフィールドセットとソート順序を決定します。次のコマンドを実行することで、これらのコマンドの出力を制御できます。

手順

- **-o** オプションを使用して、LVM 表示のデフォルトのフィールドを変更します。

```
# 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
```

- **-O** オプションを使用して、LVM 表示をソートします。

```
# 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
```

- **-O** 引数と **-** 文字を使用して、表示を逆順でソートします。

```
# 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
```

関連情報

- **lvmreport(7)**、**lvs(8)**、**vgs(8)**、および **pvs(8)** の man ページ
- LVM レポート表示への単位の指定
- LVM 設定ファイルのカスタマイズ

7.2. LVM レポート表示への単位の指定

report コマンドの **--units** 引数を指定すると、LVM デバイスのサイズを 2 進数単位または 10 進数単位で表示できます。

2 進数単位

デフォルトの単位は、1024 の倍数である 2 のべき乗で表示されます。丸め記号 **<** および **>** 付きの人間が判読できる形式 (**r**)、バイト (**b**)、セクター (**s**)、キロバイト (**k**)、メガバイト (**m**)、ギガバイト

(**g**)、テラバイト (**t**)、ペタバイト (**p**)、エクサバイト (**e**)、および人間が判読できる形式 (**h**) を使用して指定できます。
--units が指定されていない場合、デフォルトの表示は **r** です。このデフォルト設定を上書きするには、**/etc/lvm/lvm.conf** ファイルの global セクションに units パラメーターを設定します。

10 進数単位

単位指定 (**R**、**B**、**S**、**K**、**M**、**G**、**T**、**P**、**E**、**H**) を大文字にすることで、表示する単位を 1000 の倍数で指定できます。

手順

- LVM の単位を 2 進数のギガバイト単位で指定します。

```
# 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
```

- **r** オプションを使用すると、出力では LVM の実際のサイズが **<** または **>** 接頭辞付きで示されます。

```
# 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
```

r 単位は、**h** (人間が読める形式) と同様に機能しますが、さらに、報告される値に **<** または **>** の接頭辞を付けて、実際のサイズが表示サイズよりわずかに大きいまたは小さいことを示します。LVM は 10 進数値を四捨五入するため、正確でないサイズが報告されます。

また、**--units g** または他の **--units** が常に正確なサイズを表示するとは限らないことも示しています。また、表示されたサイズが正確でないことを示す **<** である **r** の主な目的も示しています。この例では、VG サイズがギガバイトの正確な倍数ではなく、.01 も分数の正確な表現ではないため、値は正確ではありません。

- LVM の単位を 10 進数のギガバイト単位で指定します。

```
# 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
```

- 512 バイトとして定義されるセクター (**s**) またはカスタム単位を指定できます。次の例は、**pvs** コマンドの出力を複数のセクターとして表示します。

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

- メガバイト (**m**) を指定します。以下の例は、**pvs** コマンドの出力を 4 MB 単位で表示しています。

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

7.3. LVM 設定ファイルのカスタマイズ

lvm.conf ファイルを編集することで、特定のストレージおよびシステム要件に応じて LVM をカスタマイズできます。たとえば、**lvm.conf** を使用して、フィルター設定の変更、ボリュームグループの自動アクティブ化の設定、シンプールの管理、またはスナップショットの自動拡張を行うことができます。

手順:

1. デフォルトの **lvm.conf** ファイルを表示します。

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

デフォルトでは、**lvm.conf** ファイルには、可能な設定を表示するコメントのみが含まれています。

2. **lvm.conf** 内の設定をコメント解除して、要件に応じて **lvm.conf** ファイルをカスタマイズします。次の設定は、特定のコマンドのデフォルト表示の変更に焦点を当てています。

   - **lvm.conf** ファイルで、指定されたフィールドのみを出力するように **lvs_cols** パラメーターを調整します。

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

-o オプションを不必要に頻繁に使用しないように、**lvs -o lv_name,vg_name,lv_attr** コマンドの代わりにこのオプションを使用します。

- **lvm.conf** ファイルで、**compact_output=1** 設定を使用して、**pvs**、**vgs**、および **lvs** コマンドで空のフィールドが出力されないようにします。

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

3. **lvm.conf** ファイルを変更した後のデフォルト値を表示します。

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

関連情報

- **lvm.conf(5)** man ページ

7.4. LVM 選択基準の定義

選択基準は、**<field> <operator> <value>** 形式の一連のステートメントであり、比較演算子を使用して特定のフィールドの値を定義します。選択基準に一致するオブジェクトが処理または表示されます。ステートメントは、論理演算子とグループ化演算子によって結合します。選択基準を定義するには、**-S** または **--select** オプションの後に 1 つまたは複数のステートメントを使用します。

一部の LVM コマンドは、特定の属性に基づいて処理するオブジェクトを選択する **-S** オプションをサポートしています。これらのオブジェクトは、物理ボリューム (PV)、ボリュームグループ (VG)、または論理ボリューム (LV) です。

-S オプションは、各オブジェクトに名前を付けるのではなく、処理するオブジェクトを記述することによって機能します。これは、多くのオブジェクトを処理する場合や、各オブジェクトを個別に検索して名前を付けることが難しい場合や、複雑な特性セットを持つオブジェクトを検索する場合に役立ちます。選択オプションは、多くの名前を入力することを回避するためのショートカットとしても使用できます。

lvs -S help コマンドを使用して、フィールドの完全なセットと使用可能な演算子を表示します。**lvs** をレポートまたは処理コマンドに置き換えると、そのコマンドの詳細が表示されます。

LVM レポートおよび処理コマンドで選択基準を使用して、選択した基準を満たすオブジェクトのみを表示または処理します。

- レポートコマンドには、**pvs**、**vgs**、**lvs**、**pvdisplay**、**vgdisplay**、**lvdisplay**、および **dmsetup info -c** が含まれます。

- 処理コマンドには、**pvchange**、**vgchange**、**lvchange**、**vgimport**、**vgexport**、**vgremove**、および **lvremove** が含まれます。

手順

- **pvs** コマンドを使用した選択基準の例:

```
# 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
```

- **lvs** コマンドを使用した選択基準の例:

```
# 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
```

- より高度な例:

```
# 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
```

関連情報

- **lvmreport(7)** man ページ

第8章 共有ストレージ上での LVM の設定

共有ストレージは、複数のノードが同時にアクセスできるストレージです。LVM を使用して共有ストレージを管理できます。共有ストレージは通常、クラスターおよび高可用性セットアップで使用されます。共有ストレージがシステム上でどのように表示されるかについては、次の 2 つの一般的なシナリオがあります。

- LVM デバイスはホストに接続され、ゲストの仮想マシンに渡されて使用されます。この場合、デバイスはホストによって使用されることは決して意図されておらず、ゲストの仮想マシンによってのみ使用されます。
- マシンはファイバーチャネルなどを使用してストレージエリアネットワーク (SAN) に接続されており、SAN LUN は複数のマシンから認識されます。

8.1. 仮想マシンディスク用の LVM の設定

仮想マシンストレージがホストに公開されるのを防ぐために、LVM デバイスアクセスと LVM の **system ID** を設定できます。これを行うには、問題のデバイスをホストから除外します。これにより、ホスト上の LVM がゲストの VM に渡されたデバイスを認識したり使用したりすることがなくなります。VG 内に LVM の **system ID** をゲストの仮想マシンと一致するように設定することで、ホスト上の仮想マシンのの VG が誤って使用されるのを防ぐことができます。

手順

1. **lvm.conf** ファイルで、デバイスを除外するパスをフィルタリングします。

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

2. オプション：LVM デバイスをさらに保護できます。
   a. **lvm.conf** ファイルのホストと VM の両方で LVM の **system ID** 機能を設定します。

```
system_id_source = "uname"
```

   b. VG の **system ID** を仮想マシンの **system ID** と一致するように設定します。これにより、ゲスト仮想マシンのみが VG をアクティブ化できるようになります。

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

8.2. 1 台のマシンで SAN ディスクを使用するように LVM を設定する

SAN LUN が間違ったマシンで使用されていないことを防ぐには、それらを使用する 1 台のマシンを除くすべてのマシンの **lvm.conf** フィルターでこれらのディスクを除外します。

また、すべてのマシンで **system ID** を設定し、VG 内の **system ID** を使用するマシンと一致するように設定することで、VG が間違ったマシンによって使用されないように保護することもできます。

手順

1. **lvm.conf** ファイルで、デバイスへのパスをフィルターして除外します。

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

2. **lvm.conf** ファイルで LVM の **system ID** 機能を設定します。

```
system_id_source = "uname"
```

3. VG の **system ID** を、この VG を使用するマシンの **system ID** と一致するように設定します。

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

8.3. フェイルオーバーに SAN ディスクを使用するための LVM の設定

フェイルオーバーなどの目的で、マシン間で LUN を移動するように設定できます。使用する可能性のあるすべてのマシンに LUN を含めるように **lvm.conf** フィルターを設定し、各マシンに LVM **システム ID** を設定することで、LVM を設定できます。

次の手順では、LVM の初期設定を説明します。フェイルオーバー用の LVM のセットアップを完了し、マシン間で VG を移動するには、VG が使用できるマシンのシステム ID と一致するように VG のシステム ID を自動的に変更する **pacemaker** と LVM アクティブ化リソースエージェントを設定する必要があります。詳細は、高可用性クラスターの設定と管理 を参照してください。

手順

1. **lvm.conf** ファイルで、デバイスを除外するパスをフィルタリングします。

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

2. すべてのマシンの LVM の **system ID** 機能を **lvm.conf** ファイルに設定します。

```
system_id_source = "uname"
```

8.4. 複数のマシン間で SAN ディスクを共有するための LVM の設定

lvmlockd デーモンと **dlm** や **sanlock** などのロックマネージャーを使用すると、複数のマシンから SAN ディスク上の共有 VG にアクセスできるようになります。特定のコマンドは、使用されているロックマネージャーとオペレーティングシステムによって異なる場合があります。次の手順では、複数のマシン間で SAN ディスクを共有するように LVM を設定するために必要な手順の概要を説明します。

警告

pacemaker を使用する場合は、代わりに 高可用性クラスターの設定と管理 に示されているペースメーカーの手順を使用してシステムを設定し、起動する必要があります。

手順

1. **lvm.conf** フィルターを設定して、それらを使用するすべてのマシンの LUN を含めます。

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

2. すべてのマシンで **lvmlockd** デーモンを使用するように **lvm.conf** ファイルを設定します。

```
use_lvmlockd=1
```

3. すべてのマシンで **lvmlockd** デーモンファイルを開始します。

4. すべてのマシン上で **dlm** や **sanlock** などの **lvmlockd** で使用するロックマネージャーを開始します。

5. **vgcreate --shared** コマンドを使用して、新しい共有 VG を作成します。

6. すべてのマシンで **vgchange --lockstart** コマンドおよび **vgchange --lockstop** コマンドを使用して、既存の共有 VG へのアクセスを開始および停止します。

関連情報

- **lvmlockd(8)** man ページ

第9章 RAID 論理ボリュームの設定

論理ボリュームマネージャー (LVM) を使用して、Redundant Array of Independent Disks (RAID) ボリュームを作成および管理できます。

9.1. RAID 論理ボリューム

論理ボリュームマネージャー (LVM) は、Redundant Array of Independent Disks (RAID) レベル 0、1、4、5、6、10 をサポートします。LVM RAID ボリュームには以下の特徴があります。

- LVM は、Multiple Devices (MD) カーネルドライバーを活用した RAID 論理ボリュームを作成して管理する
- アレイから RAID1 イメージを一時的に分割し、後でアレイにマージし直すことが可能
- LVM RAID ボリュームはスナップショットに対応

その他にも、以下のような特徴があります。

クラスター

RAID 論理ボリュームはクラスターには対応していません。
RAID 論理ボリュームは 1 台のマシンに排他的に作成およびアクティブ化できますが、複数のマシンで同時にアクティブにすることはできません。

Subvolumes

RAID 論理ボリューム (LV) を作成するとき、LVM は、データまたはアレイ内のパリティーサブボリュームごとに、サイズが 1 エクステントのメタデータサブボリュームを作成します。
たとえば、2 方向の RAID1 アレイを作成すると、メタデータサブボリュームが 2 つ (**lv_rmeta_0** および **lv_rmeta_1**) と、データサブボリュームが 2 つ (**lv_rimage_0** および **lv_rimage_1**) 作成されます。同様に、3 方向ストライプ (および暗黙的なパリティーデバイスが 1 つ) の RAID4 を作成すると、メタデータサブボリュームが 4 つ (**lv_rmeta_0**、**lv_rmeta_1**、**lv_rmeta_2**、**lv_rmeta_3**)、データサブボリュームが 4 つ (**lv_rimage_0**、**lv_rimage_1**、**lv_rimage_2**、**lv_rimage_3**) 作成されます。

インテグリティー

RAID デバイスに障害が発生したり、ソフト破損が発生したときにデータが失われる場合があります。データストレージにおけるソフト破損は、ストレージデバイスから取得したデータが、そのデバイスに書き込まれるデータとは異なることを意味します。RAID LV に整合性を追加すると、ソフト破損が軽減または防止します。詳しくは、DM 整合性を備えた RAID LV の作成 を参照してください。

9.2. RAID レベルとリニアサポート

レベル 0、1、4、5、6、10、リニアなど、RAID 別の対応設定は以下のとおりです。

レベル 0

ストライピングとも呼ばれる RAID レベル 0 は、パフォーマンス指向のストライピングデータマッピング技術です。これは、アレイに書き込まれるデータがストライプに分割され、アレイのメンバーディスク全体に書き込まれることを意味します。これにより低い固有コストで高い I/O パフォーマンスを実現できますが、冗長性は提供されません。
RAID レベル 0 実装は、アレイ内の最小デバイスのサイズまで、メンバーデバイス全体にだけデータをストライピングします。つまり、複数のデバイスのサイズが少し異なる場合、それぞれのデバイスは最小ドライブと同じサイズであるかのように処理されます。したがって、レベル 0 アレイの共

通ストレージ容量は、すべてのディスクの合計容量です。メンバー ディスクのサイズが異なる場合、RAID0 は使用可能なゾーンを使用して、それらのディスクのすべての領域を使用します。

レベル 1

RAID レベル 1 (ミラーリング) は、アレイの各メンバーディスクに同一のデータを書き込み、ミラー化されたコピーを各ディスクに残すことによって冗長性を提供します。ミラーリングは、データの可用性の単純化と高レベルにより、いまでも人気があります。レベル 1 は 2 つ以上のディスクと連携して、非常に優れたデータ信頼性を提供し、読み取り集中型のアプリケーションに対してパフォーマンスが向上しますが、比較的コストが高くなります。
RAID レベル 1 は、アレイ内のすべてのディスクに同じ情報を書き込むためコストがかかります。これにより、データの信頼性が提供されますが、レベル 5 などのパリティーベースの RAID レベルよりもスペース効率が大幅に低下します。ただし、この領域の非効率性にはパフォーマンス上の利点があります。パリティーベースの RAID レベルは、パリティーを生成するためにかなり多くの CPU 電力を消費しますが、RAID レベル 1 は単に同じデータを、CPU オーバーヘッドが非常に少ない複数の RAID メンバーに複数回書き込むだけです。そのため、RAID レベル 1 は、ソフトウェア RAID が使用されているマシンや、マシンの CPU リソースが一貫して RAID アクティビティー以外の操作でアレイ化されます。

レベル 1 アレイのストレージ容量は、ハードウェア RAID 内でミラーリングされている最小サイズのハードディスクの容量と同じか、ソフトウェア RAID 内でミラーリングされている最小のパーティションと同じ容量になります。レベル 1 の冗長性は、すべての RAID タイプの中で最も高いレベルであり、アレイは 1 つのディスクのみで動作できます。

レベル 4

レベル 4 は、1 つのディスクドライブでパリティー連結を使用して、データを保護します。パリティー情報は、アレイ内の残りのメンバーディスクのコンテンツに基づいて計算されます。この情報は、アレイ内のいずれかのディスクに障害が発生した場合にデータの再構築に使用できます。その後、再構築されたデータを使用して、交換前に失敗したディスクに I/O 要求に対応でき、交換後に失敗したディスクを接続します。
パリティー専用ディスクは、RAID アレイへのすべての書き込みトランザクションにおいて固有のボトルネックとなるため、ライトバックキャッシングなどの付随する技術なしにレベル 4 が使用されることはほとんどありません。または、システム管理者が意図的にこのボトルネックを考慮してソフトウェア RAID デバイスを設計している特定の状況下で使用されます。たとえば、アレイにデータが格納されると書き込みトランザクションがほとんどないようなアレイです。RAID レベル 4 にはほとんど使用されないため、Anaconda ではこのオプションとしては使用できません。ただし、実際には必要な場合は、ユーザーが手動で作成できます。

ハードウェア RAID レベル 4 のストレージ容量は、最小メンバーパーティションの容量にパーティションの数を掛けて 1 を引いた値に等しくなります。RAID レベル 4 アレイのパフォーマンスは常に非対称です。つまり、読み込みは書き込みを上回ります。これは、パリティーを生成するときに書き込み操作が余分な CPU リソースとメインメモリー帯域幅を消費し、実際のデータをディスクに書き込むときに余分なバス帯域幅も消費するためです。これは、データだけでなくパリティーも書き込むためです。読み取り操作は、アレイが劣化状態にない限り、データを読み取るだけでパリティーを読み取る必要はありません。その結果、読み取り操作では、通常の操作条件下で同じ量のデータ転送を行う場合でも、ドライブおよびコンピューターのバス全体に生成されるトラフィックが少なくなります。

レベル 5

これは RAID の最も一般的なタイプです。RAID レベル 5 は、アレイのすべてのメンバーディスクドライブにパリティーを分散することにより、レベル 4 に固有の書き込みボトルネックを排除します。パリティー計算プロセス自体のみがパフォーマンスのボトルネックです。最近の CPU はパリティーを非常に高速に計算できます。しかし、RAID 5 アレイに多数のディスクを使用していて、すべてのデバイスの合計データ転送速度が十分に高い場合、パリティー計算がボトルネックになる可能性があります。

レベル 5 のパフォーマンスは非対称であり、読み取りは書き込みよりも大幅に優れています。RAID レベル 5 のストレージ容量は、レベル 4 と同じです。

レベル 6

パフォーマンスではなくデータの冗長性と保存が最重要事項であるが、レベル 1 の領域の非効率性が許容できない場合は、これが RAID の一般的なレベルです。レベル 6 では、複雑なパリティースキームを使用して、アレイ内の 2 つのドライブから失われたドライブから復旧できます。複雑なパリティースキームにより、ソフトウェア RAID デバイスで CPU 幅が大幅に高くなり、書き込みトランザクションの際に増大度が高まります。したがって、レベル 6 はレベル 4 や 5 よりもパフォーマンスにおいて、非常に非対称です。

RAID レベル 6 アレイの合計容量は、RAID レベル 5 および 4 と同様に計算されますが、デバイス数から追加パリティーストレージ領域用に 2 つのデバイス (1 ではなく) を引きます。

レベル 10

この RAID レベルでは、レベル 0 のパフォーマンスとレベル 1 の冗長性を組み合わせます。また、2 台以上のデバイスを使用するレベル 1 アレイの無駄なスペースをある程度削減することができます。レベル 10 では、たとえば、データごとに 2 つのコピーのみを格納するように設定された 3 ドライブアレイを作成することができます。これにより、全体用のアレイサイズを最小デバイスのみと同じサイズ (3 つのデバイス、レベル 1 アレイなど) ではなく、最小デバイスのサイズの 1.5 倍にすることができます。これにより、CPU プロセスの使用量が RAID レベル 6 のようにパリティーを計算するのを防ぎますが、これは領域効率が悪くなります。

RAID レベル 10 の作成は、インストール時には対応していません。インストール後に手動で作成できます。

リニア RAID

リニア RAID は、より大きな仮想ドライブを作成するドライブのグループ化です。

リニア RAID では、あるメンバードライブからチャンクが順次割り当てられます。最初のドライブが完全に満杯になったときにのみ次のドライブに移動します。これにより、メンバードライブ間の I/O 操作が分割される可能性はないため、パフォーマンスの向上は見られません。リニア RAID は冗長性がなく、信頼性は低下します。メンバードライブが 1 台でも故障すると、アレイ全体が使用できなくなり、データが失われる可能性があります。容量はすべてのメンバーディスクの合計になります。

9.3. LVM RAID のセグメントタイプ

RAID 論理ボリュームを作成するには、RAID タイプを **lvcreate** コマンドの **--type** 引数として指定します。ほとんどのユーザーの場合、**raid1**、**raid4**、**raid5**、**raid6**、**raid10** の 5 つの使用可能なプライマリータイプのいずれかを指定するだけで十分です。

以下の表は、考えられる RAID セグメントタイプを示しています。

表9.1 LVM RAID のセグメントタイプ

セグメントタイプ	説明
raid1	RAID1 ミラーリング。**-m** 引数を指定し、ストライピングを指定しない場合は、これが **lvcreate** コマンドの **--type** 引数のデフォルト値になります。
raid4	RAID4 専用パリティーディスク

セグメントタイプ	説明
raid5_la	• RAID5 left asymmetric • ローテートパリティー 0 + データ継続
raid5_ra	• RAID5 right asymmetric • ローテートパリティー N + データ継続
raid5_ls	• RAID5 left symmetric • **raid5** と同じです。 • ローテートパリティー 0 +データ再起動
raid5_rs	• RAID5 right symmetric • ローテートパリティー N + データ再起動
raid6_zr	• RAID6 zero restart • **raid6** と同じです。 • ローテートパリティーゼロ (左から右) + データ再起動
raid6_nr	• RAID6 N restart • ローテートパリティー N (左から右) + データ再起動
raid6_nc	• RAID6 N continue • ローテートパリティー N (左から右) + データを継続
raid10	• ストライピング + ミラーリング。**-m** 引数を指定し、1 よりも大きい数をストライプの数として指定すると、これが **lvcreate** コマンドの **--type** 引数のデフォルト値になります。 • ミラーセットのストライピング
raid0/raid0_meta	ストライピング。RAID0 では、ストライプサイズの単位で、複数のデータサブボリュームに論理ボリュームデータが分散されます。これは、パフォーマンスを向上させるために使用します。論理ボリュームのデータは、いずれかのデータサブボリュームで障害が発生すると失われます。

9.4. RAID 論理ボリュームの作成

-m 引数に指定する値に応じて、複数のコピーを持つ RAID1 アレイを作成できます。同様に、 **-i** 引数を使用して、RAID 0、4、5、6、10 論理ボリュームのストライピング数を指定できます。**-I** 引数で、ストライプのサイズを指定することもできます。以下の手順では、異なるタイプの RAID 論理ボリュームを作成するさまざまな方法を説明します。

手順

- 2 方向 RAID を作成します。以下のコマンドは、ボリュームグループ **my_vg** 内にサイズが **1G** の 2 方向 RAID1 アレイ **my_lv** を作成します。

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

- ストライピングで RAID5 アレイを作成します。次のコマンドは、ボリュームグループ **my_vg** に、3 つのストライプと 1 つの暗黙のパリティードライブ (**my_lv**) を持つ、サイズが **1G** の RAID5 アレイを作成します。LVM ストライピングボリュームと同様にストライピングの数を指定できることに注意してください。正しい数のパリティードライブが自動的に追加されます。

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

- ストライピングで RAID6 アレイを作成します。次のコマンドは、ボリュームグループ **my_vg** に 3 つの 3 ストライプと 2 つの暗黙的なパリティードライブ (**my_lv** という名前) を持つ RAID6 アレイを作成します。これは、**1G** 1 ギガバイトのサイズです。

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

検証

- 2 ウェイ RAID1 アレイである LVM デバイス **my_vg/my_lv** を表示します。

```
# 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)
```

関連情報

- **lvcreate(8)** と **lvmraid(7)** の man ページ

9.5. RAID0 ストライピング 論理ボリュームの作成

RAID0 論理ボリュームは、論理ボリュームデータをストライプサイズ単位で複数のデータサブボリューム全体に分散します。以下の手順では、ディスク間でデータをストライピングする **mylv** という LVM RAID0 論理ボリュームを作成します。

前提条件

1. 3 つ以上の物理ボリュームを作成している。物理ボリュームの作成方法は、LVM 物理ボリュームの作成 を参照してください。
2. ボリュームグループを作成している。詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. 既存のボリュームグループから RAID0 論理ボリュームを作成します。次のコマンドは、ボリュームグループ **myvg** から RAID0 ボリューム **mylv** を作成します。これは、サイズが **2G** で、ストライプが 3 つ、ストライプ サイズが **4kB** です。

```
# 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. RAID0 論理ボリュームにファイルシステムを作成します。以下のコマンドを使用すると、論理ボリュームに ext4 ファイルシステムが作成されます。

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

3. 論理ボリュームをマウントして、ファイルシステムのディスクの領域使用率を報告します。

```
# 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
```

検証

- 作成された RAID0 ストライピング論理ボリュームを表示します。

```
# 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. STORAGE RHEL システムロールを使用して RAID LVM ボリュームのストライプサイズを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL の RAID LVM ボリュームのストライプサイズを設定できます。利用可能なパラメーターを使用して Ansible Playbook をセットアップし、LVM pool with RAID を設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

9.7. RAID0 を作成するためのパラメーター

RAID0 ストライピング論理ボリュームは、**lvcreate --type raid0[meta] --stripes _Stripes --stripesize StripeSize VolumeGroup [PhysicalVolumePath]** コマンドを使用して作成することができます。

次の表は、RAID0 ストライピング論理ボリュームを作成するときに使用できるさまざまなパラメーターを説明しています。

表9.2 RAID0 ストライピング論理ボリュームを作成するためのパラメーター

パラメーター	説明
--type raid0[_meta]	**raid0** を指定すると、メタデータボリュームなしで RAID0 ボリュームが作成されます。**raid0_meta** を指定すると、メタデータボリュームとともに RAID0 ボリュームが作成されます。RAID0 には耐障害性がないため、RAID1/10 の場合のようにミラーリングされたすべてのデータブロックを格納したり、RAID4/5/6 の場合のようにすべてのパリティーブロックを計算して格納したりしません。したがって、ミラーリングされたブロックまたはパリティーブロックの再同期の進行状態を把握するメタデータボリュームは必要ありません。RAID0 から RAID4/5/6/10 への変換では、メタデータボリュームが必須となります。**raid0_meta** を指定すると、これらのメタデータボリュームが事前に割り当てられ、それぞれの割り当ての失敗を防ぐことができます。
--stripes Stripes	論理ボリュームを分散するデバイスの数を指定します。
--stripesize StripeSize	各ストライプのサイズをキロバイト単位で指定します。これは、次のデバイスに移動する前にデバイスに書き込まれるデータの量です。
VolumeGroup	使用するボリュームグループを指定します。
PhysicalVolumePath	使用するデバイスを指定します。指定しない場合は、LVM により、**Stripes** オプションに指定されているデバイスの数が、各ストライプに 1 つずつ選択されます。

9.8. ソフトデータの破損

データストレージにおけるソフト破損は、ストレージデバイスから取得したデータが、そのデバイスに書き込まれるデータとは異なることを意味します。破損したデータは、ストレージデバイスで無期限に存在する可能性があります。破損したデータは、このデータを取得および使用するまで、検出されない可能性があります。

設定のタイプに応じて、Redundant Array of Independent Disks (RAID) 論理ボリューム (LV) は、デバイスに障害が発生した場合のデータ損失を防ぎます。RAID アレイで構成されているデバイスに障害が発生した場合、その RAID LV の一部である他のデバイスからデータを回復できます。ただし、RAID 設定により、データの一貫性は確保されません。ソフト破損、無兆候破損、ソフトエラー、およびサイレントエラーでは、システム設計やソフトウェアが想定どおりに機能し続けている場合でも、破損するデータを示す用語です。

デバイスマッパー (DM) 整合性は、RAID レベル 1、4、5、6、10 で使用され、ソフト破損によるデータ損失を軽減または防止します。RAID レイヤーでは、データの結合のないコピーが、ソフト破損エラーを修正できるようになります。整合性層は、各 RAID イメージの上にありますが、追加のサブ LV が、各 RAID イメージの整合性メタデータまたはデータチェックサムを格納します。整合性のある RAID LV からデータを取得すると、整合性データのチェックサムが破損のデータを分析します。破損が検出されると、整合性レイヤーはエラーメッセージを返し、RAID 層は、別の RAID イメージからデータの破損していないコピーを取得します。RAID レイヤーは、ソフト破損を修復するために、破損したデータを、破損していないデータで書き換えます。

DM 整合性で新しい RAID LV を作成したり、既存の RAID LV に整合性を追加する場合は、以下の点を考慮してください。

- 整合性メタデータには、追加のストレージ領域が必要です。各 RAID イメージには、データに追加されるチェックサムがあるため、500MB の全データに 4 MB のストレージ領域が必要になります。
- 一部の RAID 設定には、より多くの影響がありますが、データにアクセスする際のレイテンシーにより、DM 整合性を追加するとパフォーマンスに影響が及びます。RAID1 設定は通常、RAID5 またはそのバリアントよりも優れたパフォーマンスを提供します。
- RAID 整合性ブロックサイズは、パフォーマンスにも影響を及ぼします。RAID 整合性ブロックサイズが大きいと、パフォーマンスが向上します。ただし、RAID 整合性ブロックのサイズが小さくなると、後方互換性がより高くなります。
- 利用可能な整合性モードには、**bitmap** または **journal** の 2 つがあります。通常、 **bitmap** 整合性モードは、**journal** モードよりも優れたパフォーマンスを提供します。

ヒント

パフォーマンスの問題が発生した場合は、整合性で RAID1 を使用するか、特定の RAID 設定のパフォーマンスをテストして、要件を満たすことを確認してください。

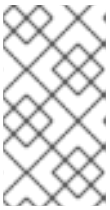
9.9. DM 整合性での RAID LV の作成

デバイスマッパー (DM) 整合性を持つ RAID LV を作成したり、既存の RAID LV に整合性を追加したりすると、ソフト破損によるデータ損失のリスクが軽減されます。LV を使用する前に、整合性の同期と RAID メタデータが完了するのを待ちます。そうしないと、バックグラウンドの初期化が LV のパフォーマンスに影響する可能性があります。

手順

1. DM 整合性のある RAID LV を作成します。次の例では、**my_vg** ボリュームグループに **test-lv** という名前の整合性を持つ新しい RAID LV を作成します。使用可能なサイズは **256M** で、RAID レベルは **1** です。

```
# 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.
```



注記

既存の RAID LV に DM 整合性を追加するには、次のコマンドを実行します。

```
# lvconvert --raidintegrity y my_vg/test-lv
```

RAID LV に整合性を追加すると、その RAID LV で実行可能な操作の数が制限されます。

2. オプション: 特定の操作を実行する前に整合性を削除します。

```
# lvconvert --raidintegrity n my_vg/test-lv
Logical volume my_vg/test-lv has removed integrity.
```

検証

- 追加された DM 整合性に関する情報を表示します。
  - **my_vg** ボリュームグループ内に作成された test-lv RAID LV の情報を表示します。

```
# 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
 [...]
```

以下は、この出力から得られるさまざまなオプションについて説明したものです。

g 属性

これは、Attr 列の下にある属性のリストで、RAID イメージが整合性を使用していることを示します。整合性は、チェックサムを **_imeta** RAID LV に保存します。

Cpy%Sync 列

最上位の RAID LV と各 RAID イメージの両方の同期の進行状況を示します。

RAID イメージ

LV 列に **raid_image_N** で表示されます。

LV 列

これにより、最上位の RAID LV と各 RAID イメージの同期の進行状況が 100% と表示されるようになります。

  - 各 RAID LV のタイプを表示します。

```
# 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
 [...]
```

  - 各 RAID イメージで検出された不一致の数をカウントする増分カウンターがあります。**my_vg/test-lv** の下の **rimage_0** から整合性で検出されたデータの不一致を表示します。

```
# 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
```

この例では、整合性はデータの不一致を検出していないため、**IntegMismatches** カウンターはゼロ (0) を示しています。

- 以下の例に示すように、**/var/log/messages** ログファイル内のデータ整合性情報を表示します。

> **例9.1 カーネルメッセージログから dm-integrity の不一致の例**
>
> device-mapper: integrity: dm-12: チェックサムがセクター 0x24e7 で失敗しました

> **例9.2 カーネルメッセージログからの dm-integrity データ修正の例**
>
> md/raid1:mdX: 読み込みエラーが修正されました (dm-16 の 9448 の 8 セクター)

関連情報

- **lvcreate(8)** と **lvmraid(7)** の man ページ

9.10. 最小/最大 I/O レートオプション

RAID10 論理ボリュームを作成する際に、sync 操作で論理ボリュームを初期化するのに必要なバックグラウンド I/O は、特に RAID 論理ボリュームを多数作成している場合に、他の I/O 操作 (ボリュームグループメタデータへの更新など) を LVM デバイスに押し出す可能性があります。これにより、他の LVM 操作が遅くなる可能性があります。

RAID 論理ボリュームが初期化される速度は、復旧スロットルを実装することで制御できます。**sync** 操作が実行される速度を制御するには、**lvcreate** コマンドの **--minrecoveryrate** および **--maxrecoveryrate** オプションを使用して、これらの操作の最小および最大 I/O 速度を設定します。

これらのオプションは次のように指定できます。

--maxrecoveryrate Rate[bBsSkKmMgG]

RAID 論理ボリュームの最大復旧速度を設定し、通常の I/O 操作が押し出されないようにします。Rate は、アレイ内の各デバイスに対する 1 秒あたりの量を指定します。接尾辞を指定しない場合は、kiB/sec/device とみなします。復旧速度を 0 に設定すると無制限になります。

--minrecoveryrate Rate[bBsSkKmMgG]

RAID 論理ボリュームの最小復旧速度を設定し、負荷の高い通常の I/O がある場合でも、同期操作の I/O が最小スループットを達成できるようにします。Rate は、アレイ内の各デバイスに対する 1 秒あたりの量を指定します。接尾辞を指定しない場合は、kiB/sec/device とみなします。

たとえば、**lvcreate --type raid10 -i 2 -m 1 -L 10G --maxrecoveryrate 128 -n my_lv my_vg** コマンドを使用して、ボリュームグループ **my_vg** 内に **3** ストライプでサイズ **10G**、最大回復速度 **128** kiB/sec/device の 2 方向の RAID10 アレイ **my_lv** を作成します。RAID のスクラブ操作の最小および最大復旧速度を指定することもできます。

9.11. リニアデバイスの RAID 論理ボリュームへの変換

既存のリニア論理ボリュームを RAID 論理ボリュームに変換することができます。この操作を行うには、**lvconvert** コマンドの **--type** 引数を使用します。

RAID 論理ボリュームは、メタデータとデータのサブボリュームのペアで構成されています。リニアデバイスを RAID1 アレイに変換すると、新しいメタデータサブボリュームが作成され、リニアボリュームと同じ物理ボリューム上の元の論理ボリュームと関連付けられます。追加のイメージは、メタデータ/データ サブボリュームのペアに追加されます。複製元の論理ボリュームとペアのメタデータイメージを同じ物理ボリュームに配置できないと、**lvconvert** は失敗します。

手順

1. 変換が必要な論理ボリュームデバイスを表示します。

```
# lvs -a -o name,copy_percent,devices my_vg
  LV     Copy%  Devices
  my_lv         /dev/sde1(0)
```

2. リニア論理ボリュームを RAID デバイスに変換します。以下のコマンドは、ボリュームグループ __my_vg のリニア論理ボリューム **my_lv** を、2 方向の RAID1 アレイに変換します。

```
# 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.
```

検証

- 論理ボリュームが RAID デバイスに変換されているかどうかを確認します。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

9.12. LVM RAID1 論理ボリュームを LVM リニア論理ボリュームに変換

既存の RAID1 LVM 論理ボリュームを LVM リニア論理ボリュームに変換することができます。この操作を行うには、**lvconvert** コマンドを使用し、**-m0** 引数を指定します。これにより、RAID アレイを構成する全 RAID データサブボリュームおよび全 RAID メタデータサブボリュームが削除され、最高レベルの RAID1 イメージがリニア論理ボリュームとして残されます。

手順

1. 既存の LVM RAID1 論理ボリュームを表示します。

```
# 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. 既存の RAID1 LVM 論理ボリュームを LVM リニア論理ボリュームに変換します。以下のコマンドは、LVM RAID1 論理ボリューム **my_vg/my_lv** を、LVM リニアデバイスに変換します。

```
# 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.
```

LVM RAID1 論理ボリューム を LVM リニアボリュームに変換する場合は、削除する物理ボリュームを指定することもできます。以下の例では、**lvconvert** コマンドは **/dev/sde1** を削除して、**/dev/sdf1** をリニアデバイスを構成する物理ボリュームとして残すように指定します。

```
# lvconvert -m0 my_vg/my_lv /dev/sde1
```

検証

- RAID1 論理ボリュームが LVM リニアデバイスに変換されたかどうかを確認します。

```
# lvs -a -o name,copy_percent,devices my_vg
  LV    Copy%  Devices
  my_lv        /dev/sdf1(1)
```

関連情報

- **lvconvert(8)** の man ページ

9.13. ミラーリングされた LVM デバイスの RAID1 論理ボリュームへの変換

セグメントタイプのミラーを持つ既存のミラーリングされた LVM デバイスを RAID1 LVM デバイスに変換できます。この操作を行うには、**--type** raid1 引数を指定して、**lvconvert** コマンドを使用します。これにより、**mimage** という名前のミラーサブボリュームの名前が、**rimage** という名前の RAID サブボリュームに変更されます。

さらに、ミラーログも削除し、対応するデータサブボリュームと同じ物理ボリューム上に、データサブボリューム用の **rmeta** という名前のメタデータサブボリュームを作成します。

手順

1. ミラーリングされた論理ボリューム **my_vg/my_lv の** のレイアウトを表示します。

```
# 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. ミラーリングされた論理ボリューム **my_vg/my_lv** を RAID1 論理ボリュームに変換します。

```
# 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.
```

検証

- ミラーリングされた論理ボリュームが RAID1 論理ボリュームに変換されているかどうかを確認します。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

9.14. RAID 論理ボリュームのサイズを変更するコマンド

RAID 論理ボリュームのサイズは、以下の方法で変更できます。

- いずれのタイプの RAID 論理ボリュームのサイズも、**lvresize** コマンドまたは **lvextend** コマンドで増やすことができます。これは、RAID イメージの数を変更するものではありません。ストライプ化 RAID 論理ボリュームでは、ストライプ化 RAID 論理ボリュームの作成時に同様のストライプを丸める制約が適用されます。

- いずれのタイプの RAID 論理ボリュームのサイズも、**lvresize** コマンドまたは **lvreduce** コマンドで減らすことができます。これは、RAID イメージの数を変更するものではありません。**lvextend** コマンドと同様に、ストライプ化 RAID 論理ボリュームの作成時にも同じストライプを丸める制約が制約が適用されます。

- **lvconvert** コマンドの **--stripes N** パラメーターを使用して、RAID4、RAID5、RAID6、または RAID10 などのストライプ化 RAID 論理ボリューム上のストライプの数を変更できます。このように、ストライプを追加または削除することで、RAID 論理ボリュームのサイズを増減できます。raid10 ボリュームにはストライプを追加することしかできないため注意してください。この機能は RAID の再形成機能の一部です。この機能を使用すると、同じ RAID レベルを維持したまま RAID 論理ボリュームの属性を変更できます。

9.15. 既存の RAID1 デバイスのイメージ数を変更

LVM ミラーリングの実装でイメージの数を変更できる方法と同様に、既存の RAID1 アレイのイメージの数を変更できます。

lvconvert コマンドを使用して RAID1 論理ボリュームにイメージを追加すると、次の操作を実行できます。

- 結果として作成されるデバイス用イメージの総数を指定する

- デバイスに追加するイメージの数

- オプションで、新しいメタデータ/データイメージのペアが存在する物理ボリュームを指定する

手順

1. 2 ウェイ RAID1 アレイである LVM デバイス **my_vg/my_lv** を表示します。

```
# 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)
```

メタデータサブボリューム (**rmeta** と呼ばれる) は、対応するデータサブボリューム (**rimage**) と同じ物理デバイスに常に存在します。メタデータ/データのサブボリュームのペアは、**--alloc** をどこかに指定しない限り、RAID アレイにある別のメタデータ/データサブボリュームのペアと同じ物理ボリュームには作成されません。

2. 2 ウェイ RAID1 論理ボリューム **my_vg/my_lv** を 3 ウェイ RAID1 論理ボリュームに変換します。

```
# 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.
```

既存の RAID1 デバイスのイメージ数を変更する場合の例を以下に示します。

- また、RAID にイメージを追加する際に、使用する物理ボリュームを指定することもできます。次のコマンドは、アレイに使用する物理ボリューム **/dev/sdd1** を指定することにより、2 方向 RAID1 論理ボリューム **my_vg/my_lv** を 3 方向 RAID1 論理ボリュームに変換します。

  ```
  # lvconvert -m 2 my_vg/my_lv /dev/sdd1
  ```

- 3 方向 RAID1 論理ボリュームを 2 方向 RAID1 論理ボリュームに変換します。

  ```
  # 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.
  ```

- 削除するイメージを含む物理ボリューム **/dev/sde1** を指定して、3 方向 RAID1 論理ボリュームを 2 方向 RAID1 論理ボリュームに変換します。

  ```
  # lvconvert -m1 my_vg/my_lv /dev/sde1
  ```

  また、イメージとその関連付けられたメタデータのサブボリュームを削除すると、それよりも大きな番号のイメージが下に移動してそのスロットを引き継ぎます。**lv_rimage_0**、**lv_rimage_1**、および **lv_rimage_2** で構成される 3 方向 RAID1 アレイから**lv_rimage_1** を削除すると、**lv_rimage_0** と **lv_rimage_1** で構成される RAID1 アレイになります。サブボリューム **lv_rimage_2** の名前が、空のスロットを引き継いで **lv_rimage_1** になります。

検証

- 既存の RAID1 デバイスのイメージ数を変更した後に、RAID1 デバイスを表示します。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

9.16. RAID イメージを複数の論理ボリュームに分割

RAID 論理ボリュームのイメージを分割して新しい論理ボリュームを形成できます。既存の RAID1 論理ボリュームから RAID イメージを削除する場合と同様に、RAID データのサブボリューム (およびその関連付けられたメタデータのサブボリューム) をデバイスから削除する場合、それより大きい番号のイメージは、そのスロットを埋めるために番号が変更になります。そのため、RAID アレイを構成する論理ボリューム上のインデックス番号は連続する整数となります。

注記

RAID1 アレイがまだ同期していない場合は、RAID イメージを分割できません。

手順

1. 2 ウェイ RAID1 アレイである LVM デバイス **my_vg/my_lv** を表示します。

```
# 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. RAID イメージを別の論理ボリュームに分割します。

   - 以下の例は、2 方向の RAID1 論理ボリューム **my_lv** を、**my_lv** と **new** の 2 つのリニア論理ボリュームに分割します。

     ```
     # 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
     ```

   - 以下の例は、3 方向の RAID1 論理ボリューム **my_lv** を、2 方向の RAID1 論理ボリューム **my_lv** と、リニア論理ボリューム **new** に分割します。

     ```
     # lvconvert --splitmirror 1 -n new my_vg/my_lv
     ```

検証

- RAID 論理ボリュームのイメージを分割した後に、論理ボリュームを表示します。

```
# lvs -a -o name,copy_percent,devices my_vg
  LV      Copy%  Devices
  my_lv          /dev/sde1(1)
  new            /dev/sdf1(1)
```

関連情報

- **lvconvert(8)** の man ページ

9.17. RAID イメージの分割とマージ

lvconvert コマンドで、**--splitmirrors** 引数とともに **--trackchanges** 引数を使用すると、すべての変更を追跡しながら、RAID1 アレイのイメージを一時的に読み取り専用に分割できます。この機能を使えば、イメージを分割した後に変更した部分のみを再同期しながら、後でイメージをアレイに統合することができます。

--trackchanges 引数を使用して RAID イメージを分割する場合、分割するイメージを指定することはできますが、分割されるボリューム名を変更することはできません。また、作成されたボリュームには以下の制約があります。

- 作成された新規ボリュームは読み取り専用です。
- 新規ボリュームのサイズは変更できません。
- 残りのアレイの名前は変更できません。
- 残りのアレイのサイズは変更できません。
- 新規のボリュームと、残りのアレイを個別にアクティブにすることはできません。

分割されたイメージを結合することができます。イメージをマージすると、イメージが分割されてから変更したアレイの部分のみが再同期されます。

手順

1. RAID 論理ボリュームを作成します。

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. オプション: 作成された RAID 論理ボリュームを表示します。

```
# 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. 作成した RAID 論理ボリュームからイメージを分割し、残りのアレイへの変更を追跡します。

```
# 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. オプション: イメージを分割した後、論理ボリュームを表示します。

```
# 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. ボリュームをアレイにマージして戻します。

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

検証

- マージされた論理ボリュームを表示します。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

9.18. RAID 障害ポリシーの設定

/etc/lvm/lvm.conf ファイルの **raid_fault_policy** フィールドの設定に基づいて、LVM RAID はデバイスの障害を自動的に処理します。要件に応じて、**raid_fault_policy** フィールドを次のパラメーターのいずれかに設定できます。

warn

このパラメーターを使用すると、障害が発生したデバイスを手動で修復し、システムログを使用して警告を表示できます。
デフォルトでは、**lvm.conf** の **raid_fault_policy** フィールドの値は **warn** です。十分な数のデバイスが動作している場合、RAID 論理ボリュームは動作し続けます。

allocate

このパラメーターを使用すると、障害が発生したデバイスを自動的に交換できます。

9.18.1. RAID 障害ポリシーを allocate に設定する

/etc/lvm/lvm.conf ファイルで、**raid_fault_policy** フィールドを **assign** パラメーターに設定できます。この設定を使用すると、システムは障害が発生したデバイスをボリュームグループのスペアデバイスと交換しようとします。スペアデバイスがない場合は、システムログにこの情報が追加されます。

手順

1. RAID 論理ボリュームを表示します。

```
# 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. **/dev/sdb** デバイスに障害が発生したら、RAID 論理ボリュームを表示します。

```
# 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)
  [...]
```

/dev/sdb デバイスに障害が発生した場合は、システムログを表示してエラーメッセージを確認することもできます。

3. **lvm.conf** ファイルで、**raid_fault_policy** フィールドを **allocate** に設定します。

```
# vi /etc/lvm/lvm.conf
raid_fault_policy = "allocate"
```

注記

raid_fault_policy を **allocate** に設定しても、スペアデバイスがない場合、割り当ては失敗し、論理ボリュームがそのままの状態になります。割り当てが失敗した場合は、**lvconvert --repair** コマンドを使用して、失敗したデバイスを修復および交換できます。詳細は、論理ボリュームに障害が発生した RAID デバイスの交換 を参照してください。

検証

- 障害が発生したデバイスがボリュームグループの新しいデバイスに置き換えられたかどうかを確認します。

```
# 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)
```

注記

障害が発生したデバイスは交換されたが、デバイスがまだボリュームグループから削除されていないため、LVM によって障害が発生したデバイスが検出されなかったことが表示されます。**vgreduce --removemissing my_vg** コマンドを実行すると、障害が発生したデバイスをボリュームグループから削除できます。

関連情報

- **lvm.conf(5)** man ページ

9.18.2. RAID 障害ポリシーを warn に設定する

lvm.conf ファイルで、**raid_fault_policy** フィールドを **warn** パラメーターに設定できます。この設定を使用すると、システムは、障害が発生したデバイスを示す警告をシステムログに追加します。警告に基づいて、その後の手順を決定できます。

手順

1. RAID 論理ボリュームを表示します。

```
# 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. lvm.conf ファイルで、raid_fault_policy フィールドを warn に設定します。

```
# vi /etc/lvm/lvm.conf
 # This configuration option has an automatic default value.
 raid_fault_policy = "warn"
```

3. **/dev/sdb** デバイスに障害が発生したら、システムログを表示してエラーメッセージを表示します。

```
# 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.
```

/dev/sdb デバイスに障害が発生すると、システムログにエラーメッセージが表示されます。ただし、この場合、LVM はイメージの 1 つを置き換えて、RAID デバイスを自動的に修復しようとはしません。したがって、デバイスに障害が発生したら、**lvconvert** コマンドの **--repair** 引数を使用してデバイスを置き換えることができます。詳細は、論理ボリュームに障害が発生した RAID デバイスの交換 を参照してください。

関連情報

- **lvm.conf(5)** man ページ

9.19. 論理ボリュームで RAID デバイスの交換

次のシナリオに応じて、論理ボリューム内の RAID デバイスを交換できます。

- 動作中の RAID デバイスを交換します。
- 論理ボリュームに障害が発生した RAID デバイスを交換します。

9.19.1. 動作中の RAID デバイスの交換

lvconvert コマンドの **--replace** 引数を使用して、論理ボリューム内の動作中の RAID デバイスを交換できます。

警告

RAID デバイスに障害が発生した場合、次のコマンドは機能しません。

前提条件

- RAID デバイスに障害が発生していません。

手順

1. RAID1 アレイを作成します。

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. 作成した RAID1 アレイを調べます。

```
# 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. 要件に応じて、次のいずれかの方法で RAID デバイスを交換します。

   a. 交換する物理ボリュームを指定して、RAID1 デバイスを交換します。

   ```
   # lvconvert --replace /dev/sdb2 my_vg/my_lv
   ```

   b. 交換に使用する物理ボリュームを指定して、RAID1 デバイスを交換します。

   ```
   # lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
   ```

   c. 複数の replace 引数を指定して、一度に複数の RAID デバイスを交換します。

   ```
   # lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
   ```

検証

1. 交換する物理ボリュームを指定した後、RAID1 アレイを調べます。

```
# 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. 交換に使用する物理ボリュームを指定した後、RAID1 アレイを調べます。

```
# 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. 一度に複数の RAID デバイスを交換した後、RAID1 アレイを調べます。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

9.19.2. 論理ボリュームに障害が発生した RAID デバイスの交換

RAID は従来の LVM ミラーリングとは異なります。LVM ミラーリングの場合は、障害が発生したデバイスを削除します。そうしないと、障害が発生したデバイスで RAID アレイが動作し続ける間、ミラーリングされた論理ボリュームがハングします。RAID1 以外の RAID レベルの場合、デバイスを削除すると、デバイスはより低いレベルの RAID に変換されます (たとえば、RAID6 から RAID5 へ、または RAID4 または RAID5 から RAID0 への変換)。

LVM では、障害が発生したデバイスを取り外して代替デバイスを割り当てる代わりに、**lvconvert** コマンドの **--repair** 引数を使用して、RAID 論理ボリューム内で物理ボリュームとして機能する障害が発生したデバイスを交換できます。

前提条件

- ボリュームグループには、障害が発生したデバイスを置き換えるのに十分な空き容量を提供する物理ボリュームが含まれています。
  ボリュームグループに十分な空きエクステントがある物理ボリュームがない場合は、**vgextend** ユーティリティーを使用して、十分なサイズの物理ボリュームを新たに追加します。

手順

1. RAID 論理ボリュームを表示します。

```
# 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. **/dev/sdc** デバイスに障害が発生したら、RAID 論理ボリュームを表示します。

```
# 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. 障害が発生したデバイスを交換します。

```
# 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. オプション: 障害が発生したデバイスを置き換える物理ボリュームを手動で指定します。

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 代替の論理ボリュームを調べます。

```
# 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)
```

障害が発生したデバイスをボリュームグループから削除するまで、LVM ユーティリティーは、障害が発生したデバイスが見つけられないことを示しています。

6. 障害が発生したデバイスをボリュームグループから削除します。

```
# vgreduce --removemissing my_vg
```

検証

1. 障害が発生したデバイスを取り外した後、利用可能な物理ボリュームを表示します。

```
# 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. 障害が発生したデバイスを交換した後、論理ボリュームを調べます。

```
# 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)
```

関連情報

- **lvconvert(8)** および **vgreduce(8)** man ページ

9.20. RAID 論理ボリュームでのデータ整合性の確認

LVM は、RAID 論理ボリュームのスクラビングに対応します。RAID スクラビングは、アレイ内のデータおよびパリティーブロックをすべて読み込み、それが一貫しているかどうかを確認するプロセスです。**lvchange --syncactionrepair** コマンドは、アレイでバックグラウンドの同期アクションを開始します。次の属性は、データの整合性に関する詳細を提供します。

- **raid_sync_action** フィールドには、RAID 論理ボリュームが実行している現在の同期アクションが表示されます。値は次のいずれかです。

  idle

  すべての **sync** アクションが完了しました (何も実行していません)。

  resync

  マシンの不完全なシャットダウン後にアレイを初期化または再同期しています。

  recover

  アレイ内のデバイスを交換しています。

  check

  アレイの不一致を検索しています。

  repair

  不一致を検索して修復しています。

- **raid_mismatch_count** フィールドには、**check** アクション中に検出された不一致の数が表示されます。

- **Cpy%Sync** フィールドには、**sync** アクションの進行状況が表示されます。

- **lv_attr** フィールドは、追加のインジケーターを提供します。このフィールドのビット 9 は、論理ボリュームの正常性を示し、以下のインジケーターに対応しています。

  m または mismatches

m または mismatches

RAID 論理ボリュームに不一致があることを示します。この文字は、スクラビング操作によって RAID の一貫性のない部分が検出された後に表示されます。

r または refresh

LVM がデバイスラベルを読み取ることができ、デバイスが動作しているとみなされる場合でも、RAID アレイ内に障害が発生したデバイスがあることを示します。デバイスが利用可能になったことをカーネルに通知するように論理ボリュームを更新するか、デバイスに障害が発生したと思われる場合はデバイスを交換します。

手順

1. オプション: スクラビングプロセスが使用する I/O 帯域幅を制限します。RAID スクラビング操作を実行すると、**sync** アクションに必要なバックグラウンド I/O が、LVM デバイスへの他の I/O (ボリュームグループメタデータの更新など) よりも優先される可能性があります。これにより、他の LVM 操作が遅くなる可能性があります。
リカバリースロットルを実装してスクラビング操作のレートを制御できます。**lvchange --syncaction** コマンドで **--maxrecoveryrate Rate[bBsSkKmMgG]** または **--minrecoveryrate Rate[bBsSkKmMgG]** を使用して復旧速度を設定できます。詳細は、最小/最大 I/O レートオプション を参照してください。

   Rate 値は、アレイ内の各デバイスに対する 1 秒あたりのデータ通信量を指定します。接尾辞を指定しないと、オプションはデバイスごとの 1 秒あたらりの kiB を想定します。

2. アレイ内の不一致数を修復せずに、アレイ内の不一致の数を表示します。

   ```
   # lvchange --syncaction check my_vg/my_lv
   ```

   このコマンドは、アレイでバックグラウンドの同期アクションを開始します。

3. オプション: **var/log/syslog** ファイルでカーネルメッセージを確認します。

4. アレイ内の不一致を修正します。

   ```
   # lvchange --syncaction repair my_vg/my_lv
   ```

   このコマンドは、RAID 論理ボリューム内の障害が発生したデバイスを修復または交換します。このコマンドを実行したら、**var/log/syslog** ファイルでカーネルメッセージを確認できます。

検証

1. スクラビング操作に関する情報を表示します。

   ```
   # 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
   ```

関連情報

- **lvchange(8)** および **lvmraid(7)** man ページ
- 最小/最大 I/O レートオプション

9.21. RAID 論理ボリュームの別の RAID レベルへの変換

LVM は RAID テイクオーバーをサポートしています。これは、RAID 論理ボリュームの RAID レベルを別の RAID レベルに変換 (たとえば RAID 5 から RAID 6) することを意味します。RAID レベルを変更して、デバイスの障害に対する耐障害性を増減できます。

手順

1. RAID 論理ボリュームを作成します。

```
# 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. RAID 論理ボリュームを表示します。

```
# 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. RAID 論理ボリュームを別の RAID レベルに変換します。

```
# 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. オプション: このコマンドで変換を繰り返すように求められた場合は、次のコマンドを実行します。

```
# lvconvert --type raid6 my_vg/my_lv
```

検証

1. RAID レベルを変換した RAID 論理ボリュームを表示します。

```
# 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
```

関連情報

- **lvconvert(8)** および **lvmraid(8)** man ページ

9.22. RAID1 論理ボリュームでの I/O 操作

lvchange コマンドの **--writemostly** パラメーターおよび **--writebehind** パラメーターを使用して、RAID1 論理ボリュームのデバイスに対する I/O 操作を制御できます。これらのパラメーターを使用する形式は次のとおりです。

--[raid]writemostly PhysicalVolume[:{t|y|n}]

RAID1 論理ボリューム内のデバイスは、**write-mostly** としてマークして、必要な場合を除き、これらのドライブに対するすべての読み取りアクションを回避します。このパラメーターを設定することにより、ドライブに対する I/O 操作の回数を最小限に抑えることができます。このパラメーターを設定するには、**lvchange --writemostly /dev/sdb my_vg/ly_lv** コマンドを使用します。
次の方法で **writemostly** 属性を設定できます。

:y

デフォルトでは、論理ボリューム内の指定された物理ボリュームの **writemostly** 属性の値は yes です。

:n

writemostly フラグを削除するには、物理ボリュームに **:n** を追加します。

:t

writemostly 属性の値を切り替えるには、**--writemostly** 引数を指定します。この引数を 1 つのコマンドで複数回使用すると、論理ボリューム内のすべての物理ボリュームの **writemostly** 属性を一度に切り替えることができます。

--[raid]writebehind IOCount

writemostly としてマークされた保留中の書き込みの最大数を指定します。これらは、RAID1 論理ボリューム内のデバイスに適用される書き込み操作の数です。このパラメーターの値を超えると、RAID アレイがすべての書き込みアクションの完了を通知する前に、構成デバイスへのすべての書き込みアクションが同期的に完了します。
このパラメーターは **lvchange --writebehind 100 my_vg/ly_lv** コマンドを使用して設定できます。**writemostly** 属性の値をゼロに設定すると、設定がクリアされます。この設定では、システムが値を任意に選択します。

9.23. RAID ボリュームの再形成

RAID の再形成とは、RAID レベルを変更せずに、RAID 論理ボリュームの属性を変更することを意味します。変更できる属性には、RAID レイアウト、ストライプのサイズ、ストライプの数などがあります。

手順

1. RAID 論理ボリュームを作成します。

```
# 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. RAID 論理ボリュームを表示します。

```
# 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. オプション: RAID 論理ボリュームの **stripes** イメージと **stripesize** を表示します。

```
# lvs -o stripes my_vg/my_lv
  #Str
    3
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  64.00k
```

4. 要件に応じて次の方法を使用して、RAID 論理ボリュームの属性を変更します。

   a. RAID 論理ボリュームの **stripes** イメージを変更します。

```
# 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. RAID 論理ボリュームの **stripesize** を変更します。

```
# 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. **maxrecoveryrate** 属性と **minrecoveryrate** 属性を変更します。

```
# 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. **syncaction** 属性を変更します。

```
# lvchange --syncaction check my_vg/my_lv
```

e. **writemostly** 属性と **writebehind** 属性を変更します。

```
# 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.
```

検証

1. RAID 論理ボリュームの **stripes** イメージと **stripesize** を表示します。

```
# lvs -o stripes my_vg/my_lv
  #Str
    4
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  128.00k
```

2. **maxrecoveryrate** 属性を変更した後、RAID 論理ボリュームを表示します。

```
# 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. **minrecoveryrate** 属性を変更した後、RAID 論理ボリュームを表示します。

```
# 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. **syncaction** 属性を変更した後、RAID 論理ボリュームを表示します。

```
# 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
[...]
```

関連情報

- **lvconvert(8)** および **lvmraid(8)** man ページ

9.24. RAID 論理ボリュームのリージョンサイズの変更

RAID 論理ボリュームを作成する場合、**/etc/lvm/lvm.conf** ファイルの **raid_region_size** パラメーターは、RAID 論理ボリュームのリージョンサイズを表します。RAID 論理ボリュームを作成した後、ボリュームのリージョンサイズを変更できます。このパラメーターは、ダーティーまたはクリーンな状態を追跡する粒度を定義します。ビットマップ内のダーティービットは、システム障害などの RAID ボリュームのダーティーシャットダウン後に同期するワークセットを定義します。

raid_region_size をより高い値に設定すると、ビットマップのサイズと輻輳が軽減されます。ただし、リージョンの同期が完了するまで RAID への書き込みが延期されるため、リージョンの再同期中の **write** 操作に影響が生じます。

手順

1. RAID 論理ボリュームを作成します。

```
# lvcreate --type raid1 -m 1 -L 10G test
  Logical volume "lvol0" created.
```

2. RAID 論理ボリュームを表示します。

```
# 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
```

Region 列は、raid_region_size パラメーターの値を示します。

3. オプション: **raid_region_size** パラメーターの値を表示します。

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

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

4. RAID 論理ボリュームのリージョンサイズを変更します。

```
# 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. RAID 論理ボリュームを再同期します。

```
# lvchange --resync my_vg/my_lv

Do you really want to deactivate logical volume my_vg/my_lv to resync it? [y/n]: y
```

検証

1. RAID 論理ボリュームを表示します。

```
# 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
```

Region 列には、**raid_region_size** パラメーターの変更後の値が表示されます。

2. **lvm.conf** ファイル内の **raid_region_size** パラメーターの値を確認します。

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

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

関連情報

- **lvconvert(8)** の man ページ

第10章 論理ボリュームのスナップショット

LVM スナップショット機能を使用すると、サービスを中断することなく、ある時点でのボリュームの仮想イメージ (**/dev/sda** など) を作成できます。

10.1. スナップショットボリュームの概要

スナップショットを作成した後で元のボリューム (スナップショットの元になるボリューム) を変更すると、スナップショット機能は、ボリュームの状態を再構築できるように、変更前の変更されたデータ領域のコピーを作成します。スナップショットを作成しても、作成元への完全な読み取り/書き込みのアクセスは引き続き可能です。

スナップショットは、スナップショットの作成後に変更したデータ部分のみをコピーするため、スナップショット機能に必要なストレージは最小限になります。たとえば、コピー元がほとんど更新されない場合は、作成元の 3 ~ 5 % の容量があれば十分にスナップショットを維持できます。バックアップ手順に代わるものではありません。スナップショットコピーは仮想コピーであり、実際のメディアバックアップではありません。

作成元のボリュームへの変更を保管するために確保する領域は、スナップショットのサイズによって異なります。たとえば、スナップショットを作成してから作成元を完全に上書きする場合に、その変更の保管に必要なスナップショットのサイズは、作成元のボリュームと同等か、それ以上になります。スナップショットのサイズは定期的に監視する必要があります。たとえば、**/usr** など、その大部分が読み取り用に使用されるボリュームの短期的なスナップショットに必要な領域は、**/home** のように大量の書き込みが行われるボリュームの長期的なスナップショットに必要な領域よりも小さくなります。

スナップショットが満杯になると、作成元のボリュームの変更を追跡できなくなるため、そのスナップショットは無効になります。ただし、スナップショットが無効になるのを防ぐために、使用量が **snapshot_autoextend_threshold** 値を超えるたびにスナップショットを自動的に拡張するように LVM を設定できます。スナップショットは完全にサイズ変更可能で、次の操作を実行できます。

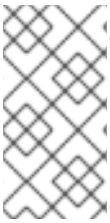
- ストレージ容量に余裕がある場合は、スナップショットボリュームのサイズを大きくして、削除されないようにすることができます。
- スナップショットのボリュームサイズが必要以上に大きければ、そのボリュームのサイズを縮小して、他の論理ボリュームで必要となる領域を確保できます。

スナップショットボリュームには、次の利点があります。

- 最も一般的な用途は、継続的にデータを更新している稼動中のシステムを停止せずに、論理ボリューム上でバックアップを実行する必要がある場合にスナップショットを撮ることです。
- スナップショットファイルシステムで **fsck** コマンドを実行し、ファイルシステムの整合性をチェックすれば、複製元のファイルシステムを修復する必要があるかどうかを判断できます。
- スナップショットは読み取りおよび書き込み用であるため、スナップショットを撮ってそのスナップショットにテストを実行することにより、実際のデータに触れることなく、実稼働データにアプリケーションのテストを実行できます。
- LVM ボリュームを作成して、Red Hat の仮想化と併用することが可能です。LVM スナップショットを使用して、仮想ゲストイメージのスナップショットを作成できます。このスナップショットは、最小限のストレージを使用して、既存のゲストの変更や新規ゲストの作成を行う上で利便性の高い方法を提供します。

10.2. 元のボリュームのスナップショット作成

lvcreate コマンドを使用して、元のボリューム (作成元) のスナップショットを作成します。ボリュームのスナップショットは書き込み可能です。デフォルトでは、シンプロビジョニングされたスナップショットと比較すると、通常のアクティベーションコマンドを実行中に、作成元のボリュームを使用して、スナップショットのボリュームを有効にします。LVM は、元のボリュームのサイズとボリュームに必要なメタデータサイズの合計よりも大きいスナップショットボリュームの作成をサポートしていません。これより大きいスナップショットボリュームを指定すると、LVM は作成元のサイズに必要なスナップショットボリュームを作成します。



注記

クラスター内のノードは LVM スナップショットをサポートしていません。共有ボリュームグループ内にスナップショットボリュームは作成できません。ただし、共有論理ボリューム上でデータの一貫したバックアップ作成が必要な場合は、ボリュームを排他的にアクティブにした上で、スナップショットを作成できます。

次の手順は、**origin** という名前の論理ボリュームを作成し、その論理ボリュームから、**snap** という名前のスナップショットボリュームを作成します。

前提条件

- ボリュームグループ **vg001** を作成している。詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. ボリュームグループ **vg001** から、論理ボリューム **origin** を作成します。

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

2. 名前が **snap** で、サイズが 100 MB のスナップショット論理ボリューム **/dev/vg001/origin** を作成します。

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

--size の代わりに **-L** 引数を、**--name** の代わりに **-n** を、そして **--snapshot** の代わりに **-s** を使用して、スナップショットを作成することもできます。

元の論理ボリュームにファイルシステムが含まれている場合は、任意のディレクトリー上でスナップショット論理ボリュームをマウントしてそのファイルシステムのコンテンツにアクセスし、元のファイルシステムが更新を継続している間にバックアップを実行できます。

3. 元のボリュームと、使用されているスナップショットボリュームの現在の割合を表示します。

```
# 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)
```

lvdisplay /dev/vg001/origin コマンドを使用して、論理ボリューム **/dev/vg001/origin** のステータスと、すべてのスナップショット論理ボリュームおよびそれらのステータス (アクティブまたは非アクティブなど) を表示することもできます。

警告

スナップショット LV の領域は、元の LV が書き込まれた後に消費されます。**lvs** コマンドは、現在のスナップショット領域の使用状況を Data% data_percent フィールド値でレポートします。スナップショットの容量が 100% に達すると、スナップショットは無効になり使用できなくなります。

無効なスナップショットは、**Attr** 列の 5 番目の位置の **I** または **lvs** の **lv_snapshot_invalid** レポートフィールドでレポートされます。**lvremove** コマンドを使用して、無効なスナップショットを削除できます。

4. オプション: 次のいずれかの方法を使用して、容量が 100% いっぱいになり無効になる前にスナップショットを拡張します。

   - **/etc/lvm.conf** ファイル内の次のパラメーターを使用して、スナップショットを自動的に拡張するように LVM を設定します。

     snapshot_autoextend_threshold

     スナップショットの使用量がこのパラメーターに設定された値を超えた後、スナップショットを拡張します。デフォルトでは 100 に設定され、自動拡張は無効になっています。このパラメーターの最小値は 50 です。

     snapshot_autoextend_percent

     現在のサイズに対する割合を指定して、その分の領域をスナップショットに追加します。デフォルトでは、20 に設定されます。

     以下の例では、次のパラメーターを設定した後、作成された 1 G のスナップショットは、使用量が 700 M を超えると 1.2 G に拡張されます。

     例10.1 スナップショットの自動拡張

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

     注記

     この機能には、ボリュームグループに未割り当てのスペースが必要です。同様に、スナップショットの自動拡張を実行しても、スナップショットに必要なサイズとして計算される最大サイズを超えて拡張されることはありません。スナップショットのサイズが複製元のボリュームを包含できるまで拡大されると、スナップショットの自動拡張はモニターされなくなります。

   - **lvextend** コマンドを使用して、このスナップショットを手動で拡張します。

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

関連情報

- **lvcreate (8)**、**lvextend (8)**、および **lvs (8)** の man ページ
- **/etc/lvm/lvm.conf** ファイル

10.3. スナップショットと元のボリュームのマージ

--merge オプションを指定して **lvconvert** コマンドを使用し、スナップショットを元のボリューム (作成元) にマージします。データやファイルを失った場合や、システムを以前の状態に復元する必要がある場合に、システムのロールバックを実行できます。スナップショットボリュームをマージすると、作成された論理ボリュームには、元のボリュームの名前、マイナー番号、および UUID が含まれます。マージの進行中、作成元に対する読み取りまたは書き込みは、マージ中のスナップショットに対して実行されているかのように見えます。マージが完了すると、マージされたスナップショットは削除されます。

作成元のボリュームとスナップショットボリュームの両方が起動されておらず、アクティブでない場合、マージはすぐに開始されます。それ以外の場合は、作成元またはスナップショットのいずれかがアクティブ化され、両方が閉じられた後にマージが開始されます。スナップショットを閉じることができない作成元 (**root** ファイルシステムなど) にマージできるのは、作成元のボリュームがアクティブ化された後です。

手順

1. スナップショットボリュームをマージします。以下のコマンドは、スナップショットボリューム **vg001/snap** をその **作成元** にマージします。

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

2. 元のボリュームを表示します。

```
# 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)
```

関連情報

- **lvconvert(8)** の man ページ

第11章 シンプロビジョニングされたボリューム (シンボリューム) の作成および管理

Red Hat Enterprise Linux は、シンプロビジョニングされたスナップショットボリュームと論理ボリュームをサポートします。

論理ボリュームとスナップショットボリュームは、シンプロビジョニングできます。

- シンプロビジョニングされた論理ボリュームを使用すると、利用可能な物理ストレージよりも大きな論理ボリュームを作成できます。
- シンプロビジョニングされたスナップショットボリュームを使用すると、同じデータボリュームにより多くの仮想デバイスを格納できます。

11.1. シンプロビジョニングの概要

最新のストレージスタックの多くは、シックプロビジョニングとシンプロビジョニングのどちらかを選択できるようになりました。

- シックプロビジョニングは、ブロックストレージの従来の動作を実行でき、実際の使用状況に関係なくブロックが割り当てられます。
- シンプロビジョニングは、ブロックストレージのプールよりも大きいサイズ (データを保存する物理デバイスよりもサイズが大きくなる可能性のあるブロックストレージ) をプロビジョニングできるので、過剰にプロビジョニングされる可能性があります。個々のブロックは実際に使用されるまで割り当てられないため、オーバープロビジョニングが発生する可能性があります。同じプールを共有する複数のシンプロビジョニングされたデバイスがある場合に、これらのデバイスは過剰にプロビジョニングされる可能性があります。

シンプロビジョニングを使用すると、物理ストレージをオーバーコミットでき、代わりにシンプールと呼ばれる空き領域のプールを管理できます。アプリケーションで必要な場合は、このシンプールを任意の数のデバイスに割り当てることができます。シンプールは、ストレージ領域をコスト効率よく割り当てる必要がある場合に、動的に拡張できます。

たとえば、10 人のユーザーから、各自のアプリケーションに使用するファイルシステムをそれぞれ 100GB 要求された場合には、各ユーザーに 100GB のファイルシステムを作成します (ただし、実際には 100GB 未満のストレージが、必要に応じて使用されます)。

注記

シンプロビジョニングを使用する場合は、ストレージプールを監視して、使用可能な物理スペースが不足したときに容量を追加することが重要です。

以下は、シンプロビジョニングされたデバイスを使用する利点です。

- 使用可能な物理ストレージよりも大きい論理ボリュームを作成できます。
- 同じデータボリュームに保存する仮想デバイスを増やすことができます。
- データ要件をサポートするために論理的かつ自動的に拡張できるファイルシステムを作成できます。未使用のブロックはプールに戻され、プール内の任意のファイルシステムで使用できます。

シンプロビジョニングされたデバイスを使用する場合に発生する可能性のある欠点は次のとおりです。

- シンプロビジョニングされたボリュームには、使用可能な物理ストレージが不足するという固有のリスクがあります。基盤となるストレージを過剰にプロビジョニングした場合に、使用可能な物理ストレージが不足しているために停止する可能性があります。たとえば、バッキング用に 1T の物理ストレージのみを使用して、10T のシンプロビジョニングストレージを作成する場合に、この 1T が使い果たされると、ボリュームは使用不可または書き込み不能になります。

- ボリュームが、シンプロビジョニングデバイスの後の階層に破棄するように送信していない場合には、使用量の計算が正確でなくなります。たとえば、**-o discard mount** オプションを指定せずにファイルシステムを配置し、シンプロビジョニングされたデバイス上で **fstrim** を定期的に実行しないと、以前に使用されたストレージの割り当てが解除されることはありません。このような場合に、実際に使用していなくても、時間の経過とともにプロビジョニングされた量をすべて使用することになります。

- 使用可能な物理スペースが不足しないように、論理的および物理的な使用状況を監視する必要があります。

- スナップショットのあるファイルシステムでは、コピーオンライト (CoW) 操作が遅くなる可能性があります。

- データブロックは複数のファイルシステム間で混在する可能性があり、エンドユーザーにそのように表示されない場合でも、基盤となるストレージのランダムアクセス制限につながります。

11.2. シンプロビジョニングされた論理ボリュームの作成

シンプロビジョニングされた論理ボリュームを使用すると、利用可能な物理ストレージよりも大きな論理ボリュームを作成できます。シンプロビジョニングされたボリュームセットを作成すると、システムは要求されるストレージの全量を割り当てる代わりに、実際に使用する容量を割り当てることができます。

lvcreate コマンドに **-T** (または **--thin**) オプションを付けて、シンプールまたはシンボリュームを作成できます。また、**lvcreate** の **-T** オプションを使用して、1 つのコマンドで同時にシンプールとシンプロビジョニングされたボリュームの両方を作成することもできます。この手順では、シンプロビジョニングされた論理ボリュームを作成および拡張する方法について説明します。

前提条件

- ボリュームグループを作成している。詳細は、LVM ボリュームグループの作成 を参照してください。

手順

1. シンプールを作成します。

```
# 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.
```

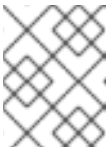
物理領域のプールを作成しているため、プールのサイズを指定する必要があります。**lvcreate** コマンドの **-T** オプションでは引数を使用できません。コマンドで指定する他のオプションをもとに、作成するデバイスのタイプを決定します。次の例に示すように、追加のパラメーターを使用してシンプールを作成することもできます。

- また、**lvcreate** コマンドの --**--thinpool** パラメーターを指定して、シンプールを作成することもできます。**-T** オプションとは異なり、**-thinpool** パラメーターでは、作成するシン

プール論理ボリュームの名前を指定する必要があります。次の例では、**-thinpool** パラメーターを使用して、サイズが 100M のボリュームグループ vg001 に mythinpool という名前のシンプールを作成します。

```
# 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.
```

- プールの作成ではストライピングがサポートされているため、**-i** および **-I** オプションを使用してストライプを作成できます。次のコマンドは、ボリュームグループ vg001 に、thinpool という名前の 2 つの 64 KB ストライプと 256 KB のチャンクサイズを持つ 100 M シンプールを作成します。また、vg001/thinvolume という名前でサイズが 1T のシンボリュームを作成します。



注記

ボリュームグループに十分な空き領域がある 2 つの物理ボリュームがあることを確認してください。そうでない場合にはシンプールを作成できません。

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

2. シンボリュームを作成します。

```
# 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.
```

このような場合には、ボリュームを含むプールのサイズよりも大きい、ボリュームの仮想サイズを指定しています。次の例に示すように、追加のパラメーターを使用してシンボリュームを作成することもできます。

- シンボリュームとシンプールの両方を作成するには、**lvcreate** コマンドの **-T** オプションを使用して、サイズと仮想サイズの両方の引数を指定します。

```
# 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.
```

- 残りの空き領域を使用してシンボリュームとシンプールを作成するには、**100%FREE** オプションを使用します。

```
# 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.
```

- 既存の論理ボリュームをシンプールボリュームに変換するには、**lvconvert** コマンドの **--thinpool** パラメーターを使用します。また、**-poolmetadata** パラメーターを **--thinpool** パラメーターと組み合わせて使用して、既存の論理ボリュームをシンプールボリュームのメタデータボリュームに変換する必要があります。
  以下の例は、ボリュームグループ **vg001** の既存の論理ボリューム **lv1** を、シンプールボリュームに変換します。また、ボリュームグループ **vg001** の既存の論理ボリューム **lv2** を、そのシンプールボリュームのメタデータボリュームに変換します。

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

注記

論理ボリュームをシンプールボリュームまたはシンプールメタデータボリュームに変換すると、論理ボリュームのコンテンツが破棄されます。**lvconvert** はデバイスのコンテンツを保存するのではなく、コンテンツを上書きするためです。

- デフォルトでは、**lvcreate** コマンドは、次の式を使用してシンプールメタデータ論理ボリュームのおおよそのサイズを設定します。

```
Pool_LV_size / Pool_LV_chunk_size * 64
```

スナップショットが大量にある場合や、シンプールのサイズが小さく、後で急激に大きくなることが予測される場合は、**lvcreate** コマンドの **--poolmetadatasize** パラメーターで、シンプールのメタデータボリュームのデフォルト値を大きくしないといけない場合があります。シンプールのメタデータ論理ボリュームで対応している値は 2 MiB - 16 GiB です。

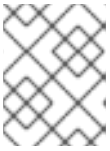
次の例は、シンプールのメタデータボリュームのデフォルト値を増やす方法を示しています。

```
# 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. 作成されたシンプールとシンボリュームを表示します。

```
# 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. オプション: **lvextend** コマンドを使用して、シンプールのサイズを拡張します。ただし、シンプールのサイズを縮小することはできません。



注記

シンプールとシンボリュームの作成中に **-l100%FREE** 引数を使用すると、このコマンドは失敗します。

以下のコマンドは、既存のシンプールのサイズ (100M) を変更し、さらに 100M 分を拡張します。

```
# 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. オプション: シンプールとシンボリュームの名前を変更するには、次のコマンドを使用します。

```
# 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"
```

名前を変更した後に、シンプールとシンボリュームを表示します。

```
# 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. オプション: シンプールを削除するには、次のコマンドを使用します。

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

関連情報

- **lvcreate (8)**、**lvrename (8)**、**lvs (8)**、および **lvconvert (8)** の man ページ

11.3. チャンクサイズの概要

チャンクは、スナップショットストレージ専用の物理ディスクの最大単位です。

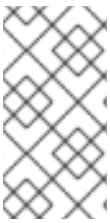
チャンクサイズを使用するには、以下の基準を使用します。

- チャンクサイズが小さいほどメタデータが増え、パフォーマンスも低下しますが、スナップショットで領域の使用率が向上します。
- チャンクサイズが大きいほどメタデータ操作は少なくなりますが、スナップショットの領域効率が低下します。

デフォルトでは、**lvm2** は 64 KiB のチャンクサイズから開始し、そのチャンクサイズに対して適切なメタデータサイズを推定します。**lvm2** が作成および使用できるメタデータの最小サイズは 2 MiB です。メタデータのサイズを 128 MiB より大きくする必要がある場合、lvm2 はチャンクサイズを増やすため、メタデータのサイズはコンパクトなまま保たれます。しかし、これによりチャンクサイズの値が大きくなり、スナップショットの使用におけるスペース効率が低下する可能性があります。このような場合、チャンクサイズを小さくし、メタデータサイズを大きくすることを推奨します。

要件に従ってチャンクサイズを指定するには、**-c** または **--chunksize** パラメーターを使用して、**lvm2** の推定チャンクサイズを無効にします。シンプールの作成後はチャンクサイズを変更できないことに注意してください。

ボリュームデータサイズが TiB の範囲にある場合は、サポートされる最大サイズである約 15.8 GiB をメタデータサイズとして使用し、要件に従ってチャンクサイズを設定します。ただし、ボリュームのデータサイズを拡張し、チャンクサイズを小さくする必要がある場合には、メタデータサイズを拡大できないことに注意してください。



注記

不適切なチャンクサイズとメタデータサイズの組み合わせを使用すると、ユーザーが **metadata** スペースを使い果たしたり、アドレス指定可能な最大シンプールデータサイズが制限されているためにシンプールサイズをそれ以上拡張できなくなったりして、問題が発生する可能性があります。

関連情報

- **lvmthin (7)** man ページ

11.4. シンプロビジョニングのスナップショットボリューム

Red Hat Enterprise Linux は、シンプロビジョニングされたスナップショットボリュームをサポートします。シン論理ボリュームのスナップショットにより、シン論理ボリューム (LV) を作成することもできます。シンプロビジョニングのスナップショットボリュームには、他のシンボリュームと同じ特性があります。ボリュームのアクティブ化、拡張、名前変更、削除、さらにはスナップショット作成も個別に行うことができます。

注記

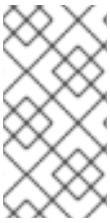
すべてのシンボリュームや、LVM スナップショットボリュームと同様に、シンプロビジョニングのスナップショットボリュームは、クラスターのノード間では対応していません。スナップショットボリュームは、1 つのクラスターノードで排他的にアクティブにする必要があります。

従来のスナップショットでは、作成されたスナップショットごとに新しい領域を割り当てる必要があります。この領域では、作成元に変更が加えられてもデータが保持されます。ただし、シンプロビジョニングスナップショットは、作成元と同じ領域を共有します。シン LV のスナップショットは、シン LV とそのスナップショットのいずれかに共通のデータブロックが共有されるので効率的です。シン LV のスナップショットを作成することも、他のシンスナップショットから作成することもできます。再帰スナップショットに共通のブロックもシンプールで共有されます。

シンプロビジョニングのスナップショットボリュームの利点は以下のとおりです。

- オリジンのスナップショットの数を増やしても、パフォーマンスへの影響はほとんどありません。
- シンスナップショットボリュームは、新しいデータのみが書き込まれ、各スナップショットにコピーされないため、ディスク使用量を減らすことができます。
- 従来のスナップショットの要件でしたが、シンスナップショットボリュームを作成元と同時にアクティブにする必要はありません。
- スナップショットからオリジンを復元する場合、シンスナップショットをマージする必要はありません。オリジンを削除して、代わりにスナップショットを使用できます。従来のスナップショットには、コピーバックする必要がある変更を保存する別のボリュームがあります。つまり、元のスナップショットにマージしてリセットする必要があります。
- 従来のスナップショットと比較して、許可されるスナップショットの上限数がはるかに増えています。

シンプロビジョニングのスナップショットボリュームを使用する利点は数多くありますが、従来の LVM スナップショットボリューム機能の方がニーズに適している場合もあります。すべてのタイプのボリュームで従来のスナップショットを使用できます。ただし、シンスナップショットを使用するには、シンプロビジョニングを使用する必要があります。



注記

シンプロビジョニングのスナップショットボリュームのサイズを制限することはできません。スナップショットは、必要な場合はシンプール内の全領域を使用します。一般的には、使用するスナップショットの形式を決定する際に、使用しているサイトの特定要件を考慮するようにしてください。

デフォルトで、シンスナップショットボリュームは、通常のアクティブ化コマンドの実行時に省略されます。

11.5. シンプロビジョニングのスナップショットボリュームの作成

シンプロビジョニングされたスナップショットボリュームを使用すると、同じデータボリュームにより多くの仮想デバイスを格納できます。

重要

シンプロビジョニングのスナップショットボリュームを作成する場合、ボリュームのサイズは指定しません。サイズパラメーターを指定すると、作成されるスナップショットはシンプロビジョニングのスナップショットボリュームにはならず、データを保管するためにシンプールを使用することもありません。たとえば、**lvcreate -s vg/thinvolume -L10M** コマンドは、作成元ボリュームがシンボリュームであっても、シンプロビジョニングのスナップショットを作成しません。

シンプロビジョニングのスナップショットは、シンプロビジョニングされた作成元ボリューム用に作成するか、シンプロビジョニングされていない作成元ボリューム用にも作成できます。次の手順では、シンプロビジョニングされたスナップショットボリュームを作成するさまざまな方法について説明します。

前提条件

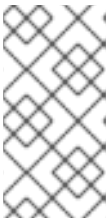
- シンプロビジョニングされた論理ボリュームを作成ている。詳細は、シンプロビジョニングの概要 を参照してください。

手順

- シンプロビジョニングされたスナップショットボリュームを作成します。以下のコマンドは、シンプロビジョニングされた論理ボリューム **vg001/thinvolume** で、シンプロビジョニングのスナップショットボリューム (名前: **mysnapshot1**) を作成します。

```
# 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
```



注記

シンプロビジョニングを使用する場合は、ストレージ管理者がストレージプールを監視し、容量が満杯になり始めたら容量を追加することが重要です。シンボリュームのサイズを拡張する方法は、シンプロビジョニングされた論理ボリュームの作成 を参照してください。

- シンプロビジョニングされていない論理ボリュームの、シンプロビジョニングされたスナップショットを作成することもできます。シンプロビジョニングされていない論理ボリュームはシンプール内に含まれていないため、外部の複製元と呼ばれます。外部の作成元ボリュームは、複数の異なるシンプールからであっても、多くのシンプロビジョニングのスナップショットボリュームで使用でき、共有できます。外部の作成元は、シンプロビジョニングのスナップショットが作成される際に非アクティブであり、かつ読み取り専用である必要があります。次の例では、**origin_volume** という名前の読み取り専用の非アクティブな論理ボリュームのシンスナップショットボリュームを作成します。このシンプロビジョニングのスナップショットボリュームの名前は **mythinsnap** です。論理ボリューム **origin_volume** は、既存のシンプール **vg001/pool** を使用する、ボリュームグループ **vg001** 内のシンプロビジョニングのスナップショットボリューム **mythinsnap** に対する外部の作成元になります。作成元のボリュームは、スナップショットボリュームと同じボリュームグループに属している必要があります。作成元の論理ボリュームを指定するときは、ボリュームグループを指定しないでください。

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

- 以下のコマンドを実行して、最初のスナップショットボリュームの 2 番目のシンプロビジョニングのスナップショットボリュームを作成できます。

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

3 番目のシンプロビジョニングされたスナップショットボリュームを作成するには、次のコマンドを使用します。

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

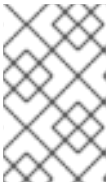
検証

- シンスナップショット論理ボリュームのすべての祖先と子孫のリストを表示します。

```
$ 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
```

ここでは、以下のようになります。

- **thinvolume** は、ボリュームグループ **vg001** で元となるボリュームです。
- **mysnapshot1** は **thinvolume** のスナップショットです。
- **mysnapshot2** は **mysnapshot1** のスナップショットです。
- **mysnapshot3** は **mysnapshot2** のスナップショットです、



注記

lv_ancestors フィールドと **lv_descendants** フィールドには、既存の依存関係が表示されます。ただし、削除されたエントリーは追跡しません。このチェーンの最中にエントリーが削除されると、依存関係チェーンが壊れるためです。

関連情報

- **lvcreate(8)** の man ページ

第12章 キャッシュを有効にして論理ボリュームのパフォーマンスを改善

LVM 論理ボリュームにキャッシュを追加して、パフォーマンスを向上できます。LVM は、SSD などの高速なデバイスを使用して、論理ボリュームに I/O 操作をキャッシュします。

以下の手順では、高速デバイスから特別な論理ボリュームを作成し、この特別な論理ボリュームを元の論理ボリュームに接続して、パフォーマンスを向上させます。

12.1. LVM でのキャッシュの取得方法

LVM は、以下のようなキャッシュの取得方法を提供します。論理ボリューム上のさまざまなタイプの I/O パターンに適しています。

dm-cache

このメソッドは、高速なボリユームで頻繁に使用されるデータをキャッシュして、このようなデータへのアクセス時間を短縮します。このメソッドは、読み取りおよび書き込みの両方の操作をキャッシュします。
dm-cache メソッドは、**cache** タイプの論理ボリュームを作成します。

dm-writecache

このメソッドは、書き込み操作のみをキャッシュします。高速なボリュームは書き込み操作を保存し、それらをバックグラウンドで低速なディスクに移行します。高速ボリュームは通常 SSD または永続メモリー (PMEM) ディスクです。
dm-writecache メソッドは、**writecache** タイプの論理ボリュームを作成します。

関連情報

- **lvmcache(7)** man ページ

12.2. LVM キャッシュコンポーネント

LVM は、キャッシュを LVM 論理ボリュームに追加するためのサポートを提供します。LVM キャッシュは、LVM 論理ボリュームタイプを使用します。

Main LV

より大きく、より遅い、元のボリューム。

キャッシュプール LV

メイン LV からデータをキャッシュするために使用できる複合 LV。キャッシュデータを保持するためのデータと、キャッシュデータを管理するためのメタデータの 2 つのサブ LV があります。データおよびメタデータ用に特定のディスクを設定できます。キャッシュプールは **dm-cache** でのみ使用できます。

Cachevol LV

メイン LV からデータをキャッシュするために使用できる線形 LV。データとメタデータ用に個別のディスクを設定することはできません。**cachevol** は、**dm-cache** または **dm-writecache** でのみ使用できます。

これらの関連付けられた LV はすべて、同じボリュームグループにある必要があります。

メインの論理ボリューム (LV) を、キャッシュされたデータを保持する高速で通常は小さい LV と組み合

わせることができます。高速 LV は、SSD ドライブなどの高速ブロックデバイスから作成されます。論理ボリュームのキャッシュを有効にすると、LVM は元のボリュームの名前を変更および非表示にし、元の論理ボリュームで設定される新しい論理ボリュームを表示します。新しい論理ボリュームの設定は、キャッシュ方法と、**cachevol** オプションまたは **cachepool** オプションを使用しているかどうかによって異なります。

cachevol オプションおよび **cachepool** オプションは、キャッシングコンポーネントの配置に対するさまざまなレベルの制御を公開します。

- **cachevol** オプションを使用すると、高速なデバイスは、データブロックのキャッシュされたコピーとキャッシュ管理用のメタデータの両方を保存します。
- **cachepool** オプションを使用すると、別のデバイスはデータブロックのキャッシュコピーとキャッシュ管理用のメタデータを保存できます。
  dm-writecache メソッドは、**cachepool** と互換性がありません。

すべての設定において、LVM は、結果として作成される 1 つのデバイスを公開し、すべてのキャッシングコンポーネントをグループ化します。作成されるデバイスは、元の低速な論理ボリュームと同じ名前になります。

関連情報

- **lvmcache(7)** man ページ
- シンプロビジョニングされたボリューム (シンボリューム) の作成および管理

12.3. 論理ボリュームの DM-CACHE キャッシュの有効化

この手順では、**dm-cache** メソッドを使用して、論理ボリュームで一般的に使用されるデータのキャッシュを有効にします。

前提条件

- システムに、**dm-cache** を使用した高速化したい低速な論理ボリュームがある。
- 低速な論理ボリュームを含むボリュームグループには、高速ブロックデバイスに未使用の物理ボリュームも含まれます。

手順

1. 高速デバイスに **cachevol** ボリュームを作成します。

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

   以下の値を置き換えます。

   cachevol-size
   : **5G** などの **cachevol** ボリュームのサイズ

   fastvol
   : **cachevol** ボリュームの名前

   vg
   : ボリュームグループ名

   /dev/fast-pv

高速ブロックデバイスへのパス (例: **/dev/sdf**)

例12.1 cachevol ボリュームの作成

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

2. **cachevol** ボリュームをメインの論理ボリュームに接続して、キャッシュを開始します。

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

以下の値を置き換えます。

fastvol
: **cachevol** ボリュームの名前

vg
: ボリュームグループ名

main-lv
: 低速な論理ボリュームの名前

例12.2 メイン LV への cachevol ボリュームの接続

```
# 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.
```

検証手順

- 新しく作成した論理ボリュームで **dm-cache** が有効になっているかどうかを確認します。

```
# 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
```

関連情報

- **lvmcache(7)** man ページ

12.4. 論理ボリュームに CACHEPOOL を使用した DM-CACHE キャッシュの有効化

この手順では、キャッシュデータとキャッシュメタデータ論理ボリュームを個別に作成し、ボリュームをキャッシュプールに統合することができます。

前提条件

- システムに、**dm-cache** を使用した高速化したい低速な論理ボリュームがある。
- 低速な論理ボリュームを含むボリュームグループには、高速ブロックデバイスに未使用の物理ボリュームも含まれます。

手順

1. 高速デバイスに **cachepool** ボリュームを作成します。

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

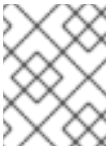
以下の値を置き換えます。

cachepool-size
cachepool のサイズ (例: **5G**)

fastpool
cachepool ボリュームの名前

vg
ボリュームグループ名

/dev/fast
高速ブロックデバイスへのパス (例: **/dev/sdf1**)



注記

--poolmetadata オプションを使用して、cache-pool の作成時にプールメタデータの場所を指定できます。

例12.3 cachevol ボリュームの作成

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

2. キャッシュを開始するために、メイン論理ボリュームに **cachepool** をアタッチします。

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

以下の値を置き換えます。

fastpool
cachepool ボリュームの名前

vg
ボリュームグループ名

main
低速な論理ボリュームの名前

例12.4 メイン LV への cachepool の接続

```
# 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.
```

検証手順

- **cache-pool** タイプで新しく作成したデバイスボリュームを調べます。

```
# 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)
```

関連情報

- **lvcreate(8)** の man ページ
- **lvmcache(7)** man ページ
- **lvconvert(8)** の man ページ

12.5. 論理ボリュームの DM-WRITECACHE キャッシュの有効化

この手順では、**dm-writecache** メソッドを使用して、論理ボリュームへの書き込み I/O 操作のキャッシュを有効にします。

前提条件

- システムに、**dm-writecache** を使用した高速化したい低速な論理ボリュームがある。
- 低速な論理ボリュームを含むボリュームグループには、高速ブロックデバイスに未使用の物理ボリュームも含まれます。
- 低速な論理ボリュームがアクティブな場合は、非アクティブ化する。

手順

1. 低速な論理ボリュームがアクティブな場合は、非アクティブにします。

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

以下の値を置き換えます。

vg

ボリュームグループ名

main-lv

低速な論理ボリュームの名前

2. 高速なデバイス上に非アクティブな **cachevol** ボリュームを作成します。

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

以下の値を置き換えます。

cachevol-size

5G などの **cachevol** ボリュームのサイズ

fastvol

cachevol ボリュームの名前

vg

ボリュームグループ名

/dev/fast-pv

高速ブロックデバイスへのパス (例: **/dev/sdf**)

例12.5 非アクティブ化された cachevol ボリュームの作成

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

3. **cachevol** ボリュームをメインの論理ボリュームに接続して、キャッシュを開始します。

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

以下の値を置き換えます。

fastvol

cachevol ボリュームの名前

vg

ボリュームグループ名

main-lv

低速な論理ボリュームの名前

例12.6 メイン LV への cachevol ボリュームの接続

```
# 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. 作成された論理ボリュームをアクティベートします。

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

以下の値を置き換えます。

vg
ボリュームグループ名

main-lv
低速な論理ボリュームの名前

検証手順

- 新たに作成されたデバイスを確認します。

```
# 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)
```

関連情報

- **lvmcache(7)** man ページ

12.6. 論理ボリュームのキャッシュの無効化

この手順では、論理ボリュームで現在有効な **dm-cache** キャッシュまたは **dm-writecache** キャッシュを無効にします。

前提条件

- キャッシュは、論理ボリュームで有効になります。

手順

1. 論理ボリュームを非アクティブにします。

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

vg はボリュームグループ名に置き換え、**main-lv** はキャッシュが有効になっている論理ボリュームの名前に置き換えます。

2. **cachevol** ボリュームまたは **cachepool** ボリュームの割り当てを解除します。

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

以下の値を置き換えます。

vg はボリュームグループ名に置き換え、**main-lv** はキャッシュが有効になっている論理ボリュームの名前に置き換えます。

例12.7 cachevol または cachepool ボリュームの接続解除

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

検証手順

- 論理ボリュームが接続されていないことを確認します。

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

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

関連情報

- **lvmcache(7)** man ページ

第13章 論理ボリュームのアクティブ化

デフォルトでは、論理ボリュームを作成すると、アクティブ状態になります。アクティブ状態の論理ボリュームは、ブロックデバイスを介して使用できます。アクティブ化された論理ボリュームにはアクセスでき、変更される可能性があります。

個々の論理ボリュームを非アクティブにして、カーネルに認識しないようにする必要がある状況はさまざまです。個々の論理ボリュームは、**lvchange** コマンドの **-a** オプションを使用してアクティブまたは非アクティブにできます。

個々の論理ボリュームを非アクティブにする形式を以下に示します。

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

個々の論理ボリュームをアクティブにする形式を以下に示します。

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

vgchange コマンドの **-a** オプションを使用して、ボリュームグループの論理ボリュームをすべてアクティブまたは非アクティブにできます。これは、ボリュームグループの個々の論理ボリュームに **lvchange -a** コマンドを実行するのと同じです。

以下は、ボリュームグループの論理ボリュームをすべて非アクティブにする形式です。

```
# vgchange -an vg
```

以下は、ボリュームグループ内のすべての論理ボリュームをアクティブにする形式です。

```
# vgchange -ay vg
```

注記

systemd-mount ユニットがマスクされていない限り、手動アクティベーション中に、**systemd** は **/etc/fstab** ファイルからの対応するマウントポイントで LVM ボリュームを自動的にマウントします。

13.1. 論理ボリュームおよびボリュームグループの自動アクティブ化の制御

論理ボリュームの自動アクティブ化は、システム起動時に論理ボリュームをイベントベースで自動的にアクティブにすることを指します。システムでデバイスが利用可能になると (デバイスのオンラインイベント)、**systemd/udev** は、各デバイスに **lvm2-pvscan** サービスを実行します。このサービスは、named デバイスを読み込む **pvscan --cache -aay device** コマンドを実行します。デバイスがボリュームグループに属している場合、**pvscan** コマンドは、そのボリュームグループに対する物理ボリュームがすべて、そのシステムに存在するかどうかを確認します。存在する場合は、このコマンドが、そのボリュームグループにある論理ボリュームをアクティブにします。

VG または LV に自動アクティブ化プロパティーを設定できます。自動アクティブ化のプロパティーが無効になっている場合、VG または LV は、**-aay** オプションを使用した **vgchange**、**lvchange**、または **pvscan** などの自動アクティブ化を実行するコマンドによってアクティブ化されません。VG で自動アクティブ化が無効になっていると、その VG で LV が自動アクティブ化されず、自動アクティブ化のプロパティーの効果はありません。VG で自動アクティブ化が有効化されている場合、個々の LV に対して自動アクティブ化を無効にできます。

手順

- 次のいずれかの方法で自動アクティブ化設定を更新できます。
  - コマンドラインを使用して VG の自動アクティブ化を制御します。

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

  - コマンドラインを使用して LV の自動アクティブ化を制御します。

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

  - 次の設定オプションのいずれかを使用して、**/etc/lvm/lvm.conf** 設定ファイルで LV の自動アクティブ化を制御します。
    - **global/event_activation**
      event_activation が無効になっている場合、**systemd/udev** は、システムの起動時に存在する物理ボリュームでのみ、論理ボリュームを自動アクティブにします。すべての物理ボリュームが表示されていないと、一部の論理ボリュームが自動的にアクティブにならない場合もあります。
    - **activation/auto_activation_volume_list**
      auto_activation_volume_list を空のリストに設定すると、自動アクティベーションは完全に無効になります。特定の論理ボリュームとボリュームグループに **auto_activation_volume_list** を設定すると、自動アクティベーションは、設定した論理ボリュームに制限されます。

関連情報

- **/etc/lvm/lvm.conf** 設定ファイル
- **lvmautoactivation(7)** の man ページ

13.2. 論理ボリュームのアクティブ化の制御

以下の方法で、論理ボリュームのアクティブ化を制御できます。

- **/etc/lvm/conf** ファイルの **activation/volume_list** 設定で行います。これにより、どの論理ボリュームをアクティブにするかを指定できます。このオプションの使用方法の詳細は **/etc/lvm/lvm.conf** 設定ファイルを参照してください。
- 論理ボリュームのアクティブ化スキップフラグで行います。このフラグが論理ボリュームに設定されていると、通常のアクティベーションコマンド時にそのボリュームがスキップされます。

または、**lvcreate** または **lvchange** コマンドで **--setactivationskip y|n** オプションを使用して、アクティブ化スキップフラグを有効化または無効化できます。

手順

- 以下の方法で、論理ボリュームのアクティブ化スキップフラグを設定できます。
  - このアクティブ化スキップフラグが論理ボリュームに設定されているかを確認するには、**lvs** コマンドを実行します。実行すると、以下のような **k** 属性が表示されます。

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

標準オプション **-ay** または **--activate y** の他に、**-K** オプションまたは **--ignoreactivationskip** オプションを使用して、**k** 属性セットで論理ボリュームをアクティブにできます。

デフォルトでは、シンプロビジョニングのスナップショットボリュームに、作成時にアクティブ化スキップのフラグが付いています。**/etc/lvm/lvm.conf** ファイルの **auto_set_activation_skip** 設定で、新たに作成した、シンプロビジョニングのスナップショットボリュームの、デフォルトのアクティブ化スキップ設定を制御できます。

- 以下のコマンドは、アクティブ化スキップフラグが設定されているシンスナップショット論理ボリュームをアクティブ化します。

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

- 以下のコマンドは、アクティブ化スキップフラグがないシンスナップショットを作成します。

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

- 以下のコマンドは、スナップショット論理ボリュームから、アクティブ化スキップフラグを削除します。

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

検証手順

- アクティブ化スキップフラグのないシンスナップショットが作成されているか確認します。

```
# 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. 共有論理ボリュームのアクティベーション

以下のように、**lvchange** コマンドおよび **vgchange** コマンドの **-a** オプションを使用して、共有論理ボリュームの論理ボリュームのアクティブ化を制御できます。

コマンド	アクティベーション
lvchange -ay|-aey	共有論理ボリュームを排他モードでアクティベートし、1台のホストのみが論理ボリュームをアクティベートできるようにします。アクティベーションが失敗 (論理ボリュームが別のホストでアクティブの場合に発生する可能性あり) すると、エラーが報告されます。
lvchange -asy	共有モードで共有論理ボリュームをアクティブにし、複数のホストが論理ボリュームを同時にアクティブにできるようにします。アクティブできない場合は (その論理ボリュームは別のホストで排他的にアクティブになっている場合に発生する可能性あり)、エラーが報告されます。スナップショットなど、論理タイプが共有アクセスを禁止している場合は、コマンドがエラーを報告して失敗します。複数のホストから同時に使用できない論理ボリュームタイプには、シン、キャッシュ、RAID、およびスナップショットがあります。
lvchange -an	論理ボリュームを非アクティブにします。

13.4. 欠落しているデバイスを含む論理ボリュームのアクティブ化

--activationmode partial|degraded|complete オプションを指定した **lvchange** コマンドを使用して、デバイスがない LV をアクティベートするかどうかを制御できます。値の説明は次のとおりです。

アクティベーションモード	意味
complete	物理ボリュームが見つからない論理ボリュームのみをアクティベートすることを許可します。これが最も制限的なモードです。
degraded	物理ボリュームが見つからない RAID 論理ボリュームをアクティベートすることを許可します。
partial	物理ボリュームが見つからない論理ボリュームをすべてアクティベートすることを許可します。このオプションは、回復または修復にのみ使用してください。

activationmode のデフォルト値は、**/etc/lvm/lvm.conf** ファイルの **activationmode** 設定により決まります。コマンドラインオプションが指定されていない場合に使用されます。

関連情報

- **lvmraid(7)** の man ページ

第14章 LVM デバイスの可視性および使用を制限する

論理ボリュームマネージャー (LVM) がスキャンできるデバイスを制御することにより、LVM で表示および使用できるデバイスを制限できます。

LVM デバイススキャンの設定を調整するには、**/etc/lvm/lvm.conf** ファイルで LVM デバイスフィルター設定を編集します。**lvm.conf** ファイル内のフィルターは、一連の単純な正規表現で設定されています。システムは、これらの式を **/dev** ディレクトリー内の各デバイス名に適用して、検出された各ブロックデバイスを受け入れるか拒否するかを決定します。

14.1. LVM フィルタリング用の永続的な識別子

/dev/sda などの従来の Linux デバイス名は、システム変更や再起動中に変更される可能性があります。World Wide Identifier (WWID)、汎用一意識別子 (UUID)、パス名などの永続的な命名属性 (PNA) は、ストレージデバイスの固有の特性に基づいており、ハードウェア設定の変更に対して耐久性があります。そのため、システムの再起動後もストレージデバイスがより安定し、予測可能になります。

LVM フィルタリングに永続的なデバイス識別子を実装すると、LVM 設定の安定性と信頼性が向上します。また、デバイス名が動的な性質を持つことによるシステム起動失敗のリスクも軽減されます。

関連情報

- 永続的な命名属性
- How to configure lvm filter, when local disk name is not persistent?

14.2. LVM デバイスフィルター

論理ボリュームマネージャー (LVM) デバイスフィルターは、デバイス名パターンのリストです。これを使用すると、システムがデバイスを評価して LVM での使用が有効であると認めるのに使用する一連の必須基準を指定できます。LVM デバイスフィルターを使用すると、LVM が使用するデバイスを制御できます。これは、偶発的なデータ損失やストレージデバイスへの不正アクセスを防ぐのに役立ちます。

14.2.1. LVM デバイスフィルターのパターンの特性

LVM デバイスフィルターのパターンは正規表現の形式です。正規表現は文字で区切られ、許可の場合は **a**、拒否の場合は **r** が前に付きます。デバイスに一致する最初の正規表現は、LVM が特定のデバイスを許可するか、拒否 (無視) するかを判断します。次に、LVM はデバイスのパスに一致する最初の正規表現をリスト内で検索します。LVM はこの正規表現を使用して、デバイスを **a** の結果により承認するか、**r** の結果により拒否するかを決定します。

単一のデバイスに複数のパス名がある場合、LVM はリストの順序に従ってこれらのパス名にアクセスします。**r** パターンの前に、少なくとも 1 つのパス名が **a** パターンと一致する場合、LVM はデバイスを許可します。しかし、**a** パターンが見つかる前にすべてのパス名が **r** パターンと一致した場合、デバイスは拒否されます。

パターンに一致しないパス名は、デバイスの許可ステータスに影響を与えません。デバイスのパターンに一致するパス名がまったくない場合も、LVM はデバイスを許可します。

システム上のデバイスごとに、**udev** ルールは複数のシンボリックリンクを生成します。ディレクトリーには、システム上の各デバイスが複数のパス名を通じてアクセスできるように、**/dev/disk/by-id/**、**/dev/disk/by-uuid/**、**/dev/disk/by-path/** などのシンボリックリンクが含まれています。

フィルター内のデバイスを拒否するには、その特定のデバイスに関連付けられたすべてのパス名が、対

応する **r** 拒否表現と一致する必要があります。ただし、考えられる拒否対象のパス名をすべて特定するのは困難な場合があります。そのため、一連の特定の **a** 表現の後に 1 つの **r|.*|** 表現を使用して、特定のパスを明確に限定して許可し、その他のすべてを拒否するフィルターを作成することを推奨します。

フィルターで特定のデバイスを定義するときは、カーネル名の代わりにそのデバイスのシンボリックリンク名を使用します。**/dev/sda** など、デバイスのカーネル名は変更される可能性がありますが、**/dev/disk/by-id/wwn-*** などの特定のシンボリックリンク名は変更されません。

デフォルトのデバイスフィルターは、システムに接続されているすべてのデバイスを許可します。理想的なユーザー設定のデバイスフィルターは、1 つ以上のパターンを許可し、それ以外はすべて拒否します。たとえば、**r|.*|** で終わるパターンリストなどが挙げられます。

LVM デバイスのフィルター設定は、**lvm.conf** ファイルの **devices/filter** および **devices/global_filter** 設定フィールドにあります。**devices/filter** 設定フィールドと **devices/global_filter** 設定フィールドは同等のものです。

関連情報

- **lvm.conf(5)** man ページ

14.2.2. LVM デバイスフィルター設定の例

次の例は、LVM がスキャンして後で使用するデバイスを制御するためのフィルター設定を示しています。**lvm.conf** ファイルでデバイスフィルターを設定するには、LVM デバイスフィルター設定の適用 を参照してください。

注記

コピーまたはクローンされた物理ボリューム (PV) を扱うときに、重複した PV の警告が発生する可能性があります。これはフィルターをセットアップすることで解決できます。マルチパス化された LVM デバイスの重複した物理ボリューム警告のトラブルシューティング のフィルター設定の例を参照してください。

- すべてのデバイスをスキャンするには、次のように入力します。

```
filter = [ "a|.*|" ]
```

- ドライブにメディアが含まれていない場合の遅延を避けるために **cdrom** デバイスを削除するには、次のように入力します。

```
filter = [ "r|^/dev/cdrom$|" ]
```

- すべてのループデバイスを追加し、他のすべてのデバイスを削除するには、次のように入力します。

```
filter = [ "a|loop|", "r|.*|" ]
```

- すべてのループデバイスと SCSI デバイスを追加し、他のすべてのブロックデバイスを削除するには、次のように入力します。

```
filter = [ "a|loop|", "a|/dev/sd.*|", "r|.*|" ]
```

- 最初の SCSI ドライブにパーティション 8 のみを追加し、他のすべてのブロックデバイスを削除するには、次のように入力します。

```
filter = [ "a|^/dev/sda8$|", "r|.*|" ]
```

- WWID によって識別される特定のデバイスおよびすべてのマルチパスデバイスからすべてのパーティションを追加するには、次のように入力します。

```
filter = [ "a|/dev/disk/by-id/<disk-id>.|", "a|/dev/mapper/mpath.|", "r|.*|" ]
```

  また、このコマンドは、他のブロックデバイスを削除します。

関連情報

- **lvm.conf(5)** man ページ
- LVM デバイスフィルター設定の適用
- PV の重複警告を防ぐ LVM デバイスフィルターの例

14.2.3. LVM デバイスフィルター設定の適用

lvm.conf 設定ファイル内にフィルターを設定することにより、LVM がどのデバイスをスキャンするかを制御できます。

前提条件

- 使用するデバイスフィルターパターンが準備されている。

手順

1. 実際に **/etc/lvm/lvm.conf** ファイルを変更せずに、次のコマンドを使用してデバイスフィルターパターンをテストします。以下にフィルター設定の例を示します。

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|*.|" ] }'
```

2. **/etc/lvm/lvm.conf** ファイルの設定セクション **devices** にデバイスフィルターパターンを追加します。

```
filter = [ "a|/dev/emcpower.*|", "r|*.|" ]
```

3. 再起動時に必要なデバイスのみをスキャンします。

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

   このコマンドによって、LVM が必要なデバイスのみを再起動時にスキャンするように **initramfs** ファイルシステムを再構築します。

第15章 LVM の割り当ての制御

デフォルトでは、ボリュームグループは **normal** 割り当てポリシーを使用します。これにより、同じ物理ボリューム上に並行ストライプを配置しないなどの常識的な規則に従って物理エクステントが割り当てられます。**vgcreate** コマンドで **--alloc** 引数を使用すると、別の割り当てポリシー (**contiguous**、**anywhere**、または **cling**) を指定できます。一般的に、normal 以外の割り当てポリシーが必要となるのは、通常とは異なる、標準外のエクステント割り当てを必要とする特別なケースのみです。

15.1. 指定したデバイスからのエクステントの割り当て

lvcreate コマンドと **lvconvert** コマンドを使用する際に、コマンドラインの末尾でデバイス引数を使用すると、特定のデバイスからの割り当てに制限できます。より詳細に制御するために、各デバイスの実際のエクステント範囲を指定できます。このコマンドは単に、指定した物理ボリューム (PV) を引数として使用して、新しい論理ボリューム (LV) にエクステントを割り当てます。各 PV から利用可能なエクステントがなくなるまでエクステントを取得し、その後、次にリストされている PV からエクステントを取得します。リストされているすべての PV の領域が、要求された LV サイズに対して不足する場合、コマンドは失敗します。このコマンドは、指定した PV からのみ割り当てを行うことに注意してください。Raid LV は、個別の RAID イメージまたは個別のストライプにシーケンシャル PV を使用します。PV のサイズが RAID イメージ全体に対して十分でない場合、デバイスの使用量を完全に予測することはできません。

手順

1. ボリュームグループ (VG) を作成します。

```
# vgcreate <vg_name> <PV> ...
```

   ここでは、以下のようになります。

   - **<vg_name>** は VG の名前です。
   - **<PV>** は PV です。

2. PV を割り当てて、リニアや RAID などのさまざまなボリュームタイプを作成できます。

   a. エクステントを割り当ててリニアボリュームを作成します。

```
# lvcreate -n <lv_name> -L <lv_size> <vg_name> [ <PV> ... ]
```

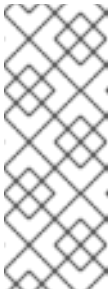
   ここでは、以下のようになります。

   - **<lv_name>** は LV の名前です。
   - **<lv_size>** は LV のサイズです。デフォルトの単位はメガバイトです。
   - **<vg_name>** は VG の名前です。
   - **[<PV ... >]**。
     コマンドラインでは、PV の 1 つを指定することも、すべての PV を指定することもできます。何も指定しないことも可能です。
     - 1 つの PV を指定すると、当該 PV から LV のエクステントが割り当てられます。

注記

PV に LV 全体に対する十分な空きエクステントがない場合、**lvcreate** は失敗します。

- 2 つの PV を指定した場合、LV のエクステントは、当該 PV のうちの 1 つ、または両方の PV の組み合わせから割り当てられます。
- PV を指定しない場合、エクステントは VG 内の PV の 1 つ、または VG 内のすべての PV の任意の組み合わせから割り当てられます。



注記

このような場合、LVM は、指定した PV または使用可能な PV のすべてを使用しない可能性があります。最初の PV に LV 全体に対して十分な空きエクステントがある場合、他の PV はおそらく使用されません。ただし、最初の PV に空きエクステントの割り当てサイズが設定されていない場合、LV の一部は最初の PV から割り当てられ、一部は 2 番目の PV から割り当てられる可能性があります。

例15.1 1つの PV からのエクステントの割り当て

この例では、**lv1** エクステントが **sda** から割り当てられます。

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

例15.2 2 つの PV からのエクステントの割り当て

この例では、**lv2** エクステントが **sda**、**sdb**、または両方の組み合わせから割り当てられます。

```
# lvcreate -n lv2 L1G vg /dev/sda /dev/sdb
```

例15.3 PV を指定しないエクステントの割り当て

この例では、**lv3** エクステントが、VG 内の PV の 1 つ、または VG 内のすべての PV の任意の組み合わせから割り当てられます。

```
# lvcreate -n lv3 -L1G vg
```

または、以下を実行します。

b. エクステントを割り当てて RAID ボリュームを作成します。

```
# lvcreate --type <segment_type> -m <mirror_images> -n <lv_name> -L <lv_size>
<vg_name> [ <PV> ... ]
```

ここでは、以下のようになります。

- **<segment_type>** は、指定したセグメントタイプ (例: **raid5**、**mirror**、**snapshot**) です。
- **<mirror_images>** は、指定した数のイメージを含む、**raid1** またはミラーリングされた LV を作成します。たとえば、**-m 1** を指定すると、2 つのイメージを含む **raid1** LV が作成されます。
- **<lv_name>** は LV の名前です。
- **<lv_size>** は LV のサイズです。デフォルトの単位はメガバイトです。
- **<vg_name>** は VG の名前です。
- **<[PV ...]>**
  最初の RAID イメージは最初の PV から割り当てられ、2 番目の RAID イメージは 2 番目の PV から割り当てられます。以後も同様です。

例15.4 2 つの PV からの RAID イメージの割り当て

この例では、**lv4** の最初の RAID イメージは **sda** から割り当てられ、2 番目のイメージは **sdb** から割り当てられます。

```
# lvcreate --type raid1 -m 1 -n lv4 -L1G vg /dev/sda /dev/sdb
```

例15.5 3 つの PV からの RAID イメージの割り当て

この例では、**lv5** の最初の RAID イメージは **sda** から割り当てられ、2 番目のイメージは **sdb** から割り当てられ、3 番目のイメージは **sdc** から割り当てられます。

```
# lvcreate --type raid1 -m 2 -n lv5 -L1G vg /dev/sda /dev/sdb /dev/sdc
```

関連情報

- **lvcreate(8)** の man ページ
- **lvconvert(8)** の man ページ
- **lvmraid(7)** の man ページ

15.2. LVM の割り当てポリシー

LVM の操作で物理エクステントを 1 つまたは複数の論理ボリューム (LV) に割り当てる必要がある場合、割り当ては以下のように行われます。

- ボリュームグループで割り当てられていない物理エクステントのセットが、割り当てのために生成されます。コマンドラインの末尾に物理エクステントの範囲を指定すると、指定した物理ボリューム (PV) の中で、その範囲内で割り当てられていない物理エクステントだけが、割り当て用エクステントとして考慮されます。
- 割り当てポリシーは順番に試行されます。最も厳格なポリシー (**contiguous**) から始まり、最後は **--alloc** オプションで指定した割り当てポリシーか、特定の LV やボリュームグループ (VG) にデフォルトとして設定されている割り当てポリシーが試行されます。各割り当てポリシーで

は、埋める必要がある空の LV 領域の最小番号の論理エクステントから始まり、割り当てポリシーによる制限に従って、できるだけ多くの領域の割り当てを行います。領域が足りなくなると、LVM は次のポリシーに移動します。

割り当てポリシーの制限は以下のとおりです。

- **contiguous** ポリシーでは、LV の最初の論理エクステントを除き、論理エクステントの物理的位置が、直前の論理エクステントの物理的位置に隣接している必要があります。
  LV がストライプ化またはミラーリングされている場合、**contiguous** の割り当て制限は、領域を必要とする各ストライプまたは RAID イメージに個別に適用されます。

- **cling** の割り当てポリシーでは、既存の LV に追加する論理エクステントに使用される PV が、その LV にある少なくとも 1 つの論理エクステントですでに使用されている必要があります。

- **normal** の割り当てポリシーでは、並行 LV 内の同じオフセットで、並行 LV (つまり、異なるストライプまたは RAID イメージ) にすでに割り当てられている論理エクステントと同じ PV を共有する物理エクステントは選択されません。

- 割り当て要求を満たすのに十分な空きエクステントがあっても、**normal** の割り当てポリシーによって使用されない場合は、たとえ同じ PV に 2 つのストライプを配置することによってパフォーマンスが低下しても、**anywhere** の割り当てポリシーがその空きエクステントを使用します。

vgchange コマンドを使用して、割り当てポリシーを変更できます。

注記

今後の更新で、定義された割り当てポリシーに基づくレイアウト操作のコードが変更される可能性があることに注意してください。たとえば、割り当て可能な空き物理エクステントの数が同じ 2 つの空の物理ボリュームをコマンドラインで指定する場合、現行バージョンの LVM では、それが表示されている順番に使用が検討されます。ただし、今後のリリースで、そのプロパティーが変更されない保証はありません。特定の LV 用に特定のレイアウトが必要な場合は、各手順に適用される割り当てポリシーに基づいて LVM がレイアウトを決定することがないように、**lvcreate** と **lvconvert** を順に使用してレイアウトを構築してください。

15.3. 物理ボリュームでの割り当て防止

pvchange コマンドを使用すると、単一または複数の物理ボリュームの空き領域で物理エクステントが割り当てられないようにすることができます。これは、ディスクエラーが発生した場合や、物理ボリュームを削除する場合に必要となる可能性があります。

手順

- **device_name** での物理エクステントの割り当てを無効にするには、次のコマンドを使用します。

```
# pvchange -x n /dev/sdk1
```

  pvchange コマンドで **-xy** 引数を使用すると、無効にされていた割り当てを許可できます。

関連情報

- **pvchange(8)** の man ページ

第16章 タグを使用した LVM オブジェクトのグループ化

論理ボリューム管理 (LVM) オブジェクトにタグを割り当てて、オブジェクトをグループ化できます。この機能を使用すると、アクティブ化など、LVM の動作の制御をグループごとに自動化できます。LVM オブジェクトのタグをコマンドとして使用することもできます。

16.1. LVM オブジェクトタグ

論理ボリューム管理 (LVM) タグは、同じタイプの LVM2 オブジェクトをグループ化するために使用する用語です。タグは、物理ボリューム、ボリュームグループ、論理ボリュームなどのオブジェクトや、クラスター設定のホストに割り当てることができます。

曖昧さを避けるために、各タグの先頭に @ を付けます。各タグは、そのタグを所有し、コマンドライン上の位置によって予期されるタイプのすべてのオブジェクトに置き換えることで拡張されます。

LVM タグは、最大 1024 文字の文字列です。LVM タグはハイフンで開始できません。

有効なタグは、限られた範囲の文字のみで設定されます。使用できる文字は、**A-Z a-z 0-9 _ + です。- / = !: # &** です。

タグを付けることができるのは、ボリュームグループ内のオブジェクトのみです。物理ボリュームは、ボリュームグループから削除されるとタグを失います。これは、タグがボリュームグループメタデータの一部として保存され、物理ボリュームが削除されると削除されるためです。

一部のコマンドは、同じタグを持つすべてのボリュームグループ (VG)、論理ボリューム (LV)、または物理ボリューム (PV) に適用できます。特定のコマンドの man ページには、**VG|Tag**、**LV|Tag**、**PV|Tag** など、VG、LV、PV 名の代わりにタグ名を使用できる場合の構文が表示されます。

16.2. LVM タグのリスト表示

以下の例は、LVM タグのリストを表示する方法を示しています。

手順

- 次のコマンドを実行すると、**database** タグを持つ論理ボリュームがすべてリスト表示されます。

```
# lvs @database
```

- 現在アクティブなホストタグのリストを表示するには、次のコマンドを使用します。

```
# lvm tags
```

16.3. LVM オブジェクトへのタグの追加

さまざまなボリューム管理コマンドで **--addtag** オプションを使用すると、LVM オブジェクトにタグを追加してグループ化できます。

前提条件

- **lvm2** パッケージがインストールされている。

手順

- 既存の PV にタグを追加するには、次を使用します。

```
# pvchange --addtag <@tag> <PV>
```

- 既存の VG にタグを追加するには、次を使用します。

```
# vgchange --addtag <@tag> <VG>
```

- 作成時に VG にタグを追加するには、次を使用します。

```
# vgcreate --addtag <@tag> <VG>
```

- 既存の LV にタグを追加するには、次を使用します。

```
# lvchange --addtag <@tag> <LV>
```

- 作成時に LV にタグを追加するには、次を使用します。

```
# lvcreate --addtag <@tag> ...
```

16.4. LVM オブジェクトからのタグの削除

LVM オブジェクトのグループ化が不要になった場合は、さまざまなボリューム管理コマンドで **--deltag** オプションを使用して、オブジェクトからタグを削除できます。

前提条件

- **lvm2** パッケージがインストールされている。
- 物理ボリューム (PV)、ボリュームグループ (VG)、または論理ボリューム (LV) にタグが作成されている。

手順

- 既存の PV からタグを削除するには、次を使用します。

```
# pvchange --deltag @tag PV
```

- 既存の VG からタグを削除するには、次を使用します。

```
# vgchange --deltag @tag VG
```

- 既存の LV からタグを削除するには、次を使用します。

```
# lvchange --deltag @tag LV
```

16.5. LVM ホストタグの定義

この手順では、クラスター設定で LVM ホストタグを定義する方法を説明します。設定ファイルにはホストタグを定義できます。

手順

- **tags** セクションに **hosttags = 1** を設定し、マシンのホスト名を使用してホストタグを自動的に定義します。
  これにより、すべてのマシンに複製できる共通の設定ファイルを使用して、ファイルのコピーを同じにすることができますが、ホスト名によって動作がマシン間で異なる場合があります。

ホストタグが存在すると、追加の設定ファイル (**lvm_hosttag.conf**) が読み込まれます。そのファイルで新しいタグが定義されている場合は、読み込むファイルのリストに設定ファイルが追加されます。

たとえば、設定ファイル内の次のエントリーは常に **tag1** を定義し、ホスト名が **host1** の場合は **tag2** を定義します。

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

16.6. タグによる論理ボリュームのアクティブ化の制御

この手順では、設定ファイルで、特定の論理ボリュームのみをそのホストでアクティブにするように指定する方法を説明します。

手順

たとえば、次のエントリーはアクティベーション要求のフィルターとして機能し (**vgchange -ay** など)、**vg1/lvol0** と、そのホストのメタデータにある **database** タグを持つ論理ボリュームまたはボリュームグループのみをアクティベートします。

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

メタデータタグがマシンのホストタグと一致する場合にのみ一致する特殊な一致 **@***。

別の例として、クラスター内のすべてのマシンに、以下のエントリーがる状況を考慮してください。

```
tags { hosttags = 1 }
```

ホストの **db2** でのみ **vg1/lvol2** をアクティベートする場合は、以下の手順を行います。

1. クラスター内の任意のホストから **lvchange --addtag @db2 vg1/lvol2** を実行します。
2. **lvchange -ay vg1/lvol2** を実行します。

このソリューションでは、ボリュームグループのメタデータ内にホスト名を保存します。

第17章 LVM のトラブルシューティング

論理ボリュームマネージャー (LVM) ツールを使用して、LVM ボリュームおよびグループのさまざまな問題のトラブルシューティングを行うことができます。

17.1. LVM での診断データの収集

LVM コマンドが想定どおりに機能しない場合は、以下の方法で診断情報を収集できます。

手順

- 以下の方法を使用して、さまざまな診断データを収集します。
  - **-v** 引数を LVM コマンドに追加して、コマンドの出力の詳細レベルを増やします。**v** を追加すると、詳細度をさらに増やすことができます。**v** は最大 4 つ許可されます (例: **-vvvv**)。
  - **/etc/lvm/lvm.conf** 設定ファイルの **log** セクションで、**level** オプションの値を増やします。これにより、LVM がシステムログにより多くの情報を提供します。
  - 問題が論理ボリュームのアクティブ化に関連する場合は、アクティブ化中に LVM がログメッセージをログに記録できるようにします。
    i. **/etc/lvm/lvm.conf** 設定ファイルの **log** セクションで **activation = 1** オプションを設定します。
    ii. LVM コマンドに **-vvvv** オプションを付けて実行します。
    iii. コマンドの出力を確認します。
    iv. **activation** オプションを **0** にリセットします。
    オプションを **0** にリセットしないと、メモリー不足の状況でシステムが応答しなくなる可能性があります。
  - 診断目的で情報ダンプを表示します。

    ```
    # lvmdump
    ```

  - 追加のシステム情報を表示します。

    ```
    # lvs -v
    ```

    ```
    # pvs --all
    ```

    ```
    # dmsetup info --columns
    ```

  - **/etc/lvm/backup/** ディレクトリーの最後の LVM メタデータのバックアップと、**/etc/lvm/archive/** ディレクトリー内のアーカイブバージョンを確認します。
  - 現在の設定情報を確認します。

    ```
    # lvmconfig
    ```

  - **/run/lvm/hints** キャッシュファイルで、物理ボリュームを持つデバイスを記録します。

関連情報

- **lvmdump(8)** の man ページ

17.2. 障害が発生した LVM デバイスに関する情報の表示

障害が発生した論理ボリュームマネージャー (LVM) ボリュームに関するトラブルシューティング情報は、障害の原因を特定するのに役立ちます。最も一般的な LVM ボリューム障害の例を以下に示します。

例17.1 障害が発生したボリュームグループ

この例では、ボリュームグループ **myvg** を設定するデバイスの1つで障害が発生しました。ボリュームグループの使用可能性は、障害の種類によって異なります。たとえば、RAID ボリュームも関係している場合、ボリュームグループは引き続き使用できます。障害が発生したデバイスに関する情報も確認できます。

```
# 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)
```

例17.2 障害が発生した論理ボリューム

この例では、デバイスの1つで障害が発生しました。このような障害が、ボリュームグループ内の論理ボリュームに障害が発生する原因となることがあります。コマンドの出力には、障害が発生した論理ボリュームが表示されます。

```
# 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)
```

例17.3 RAID 論理ボリュームのイメージの障害

次の例は、RAID 論理ボリュームのイメージに障害が発生した場合の **pvs** および **lvs** ユーティリティーからのコマンド出力を示しています。論理ボリュームは引き続き使用できます。

```
# 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. ボリュームグループから見つからない LVM 物理ボリュームの削除

物理ボリュームに障害が発生した場合は、ボリュームグループ内の残りの物理ボリュームをアクティブにし、その物理ボリュームを使用していたすべての論理ボリュームをボリュームグループから削除できます。

手順

1. ボリュームグループ内の残りの物理ボリュームをアクティベートします。

```
# vgchange --activate y --partial myvg
```

2. 削除する論理ボリュームを確認します。

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

3. ボリュームグループから、失われた物理ボリュームを使用していた論理ボリュームをすべて削除します。

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

4. 必要に応じて、保持する論理ボリュームを誤って削除した場合には、**vgreduce** 操作を元に戻すことができます。

```
# vgcfgrestore myvg
```

警告

シンプールを削除すると、LVM は操作を元に戻すことができません。

17.4. 見つからない LVM 物理ボリュームのメタデータの検索

物理ボリュームのボリュームグループメタデータ領域が誤って上書きされたり、破棄されたりする場合は、メタデータ領域が正しくないことを示すエラーメッセージか、システムが特定の UUID を持つ物理ボリュームを見つけることができないことを示すエラーメッセージが表示されます。

この手順では、物理ボリュームが見つからないか、破損している、アーカイブされた最新のメタデータを見つけます。

手順

1. 物理ボリュームを含むボリュームグループのアーカイブされたメタデータファイルを検索します。アーカイブされたメタデータファイルは、**/etc/lvm/archive/volume-group-name_backup-number.vg** パスにあります。

```
# cat /etc/lvm/archive/myvg_00000-1248998876.vg
```

00000-1248998876 を backup-number に置き換えます。ボリュームグループの番号が最も高い、既知の有効なメタデータファイルの最後のものを選択します。

2. 物理ボリュームの UUID を検索します。以下の方法のいずれかを使用します。

- 論理ボリュームをリスト表示します。

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

  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
```

- アーカイブされたメタデータファイルを確認します。ボリュームグループ設定の **physical_volumes** セクションで、**id =** のラベルが付いた値として UUID を検索します。

- **--partial** オプションを使用してボリュームグループを非アクティブにします。

```
# 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. LVM 物理ボリュームでのメタデータの復元

この手順では、破損したり、新しいデバイスに置き換えたりする物理ボリュームのメタデータを復元します。物理ボリュームのメタデータ領域を書き換えて、物理ボリュームからデータを復旧できる場合があります。

警告

作業用の LVM 論理ボリュームでこの手順を実行しないでください。誤った UUID を指定すると、データが失われることになります。

前提条件

- 見つからない物理ボリュームのメタデータを特定している。詳細は、見つからない LVM 物理ボリュームのメタデータの検索 を参照してください。

手順

1. 物理ボリュームでメタデータを復元します。

```
# pvcreate --uuid physical-volume-uuid \
        --restorefile /etc/lvm/archive/volume-group-name_backup-number.vg \
        block-device
```

注記

コマンドは、LVM メタデータ領域のみを上書きし、既存のデータ領域には影響を与えません。

例17.4 /dev/vdb1 での物理ボリュームの復元

以下の例では、以下のプロパティーで **/dev/vdb1** デバイスを物理ボリュームとしてラベル付けします。

- **FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk** の UUID

- **VG_00050.vg** に含まれるメタデータ情報 (ボリュームグループの最新のアーカイブメタデータ)

```
# 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. ボリュームグループのメタデータを復元します。

```
# vgcfgrestore myvg

 Restored volume group myvg
```

3. ボリュームグループの論理ボリュームを表示します。

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

現在、論理ボリュームは非アクティブです。以下に例を示します。

```
 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. 論理ボリュームのセグメントタイプが RAID の場合は、論理ボリュームを再同期します。

```
# lvchange --resync myvg/mylv
```

5. 論理ボリュームを非アクティブにします。

```
# lvchange --activate y myvg/mylv
```

6. ディスク上の LVM メタデータが、それを上書きしたものと同じかそれ以上のスペースを使用する場合は、この手順で物理ボリュームを回復できます。メタデータを上書きしたものがメタデータ領域を超えると、ボリューム上のデータが影響を受ける可能性があります。そのデータを復元するには、**fsck** コマンドを使用することができます。

検証手順

- アクティブな論理ボリュームを表示します。

```
# 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. LVM 出力の丸めエラー

ボリュームグループの領域使用量を報告する LVM コマンドは、報告された数を **2** 進法に切り上げ、人間が判読できる出力を提供します。これには、**vgdisplay** ユーティリティーおよび **vgs** ユーティリティーが含まれます。

丸めの結果、報告された空き領域の値は、ボリュームグループが提供する物理エクステントよりも大きくなる可能性があります。報告された空き領域のサイズの論理ボリュームを作成しようとすると、以下のエラーが発生する可能性があります。

```
Insufficient free extents
```

エラーを回避するには、ボリュームグループの空き物理エクステントの数を調べる必要があります。これは、空き領域の正確な値です。次に、エクステントの数を使用して、論理ボリュームを正常に作成できます。

17.7. LVM ボリューム作成時の丸めエラーの防止

LVM 論理ボリュームを作成する場合は、丸めエラーを防ぐために論理ボリュームの論理エクステントの数を指定できます。

手順

1. ボリュームグループの空き物理エクステントの数を検索します。

```
# vgdisplay myvg
```

例17.5 ボリュームグループの空きエクステント

たとえば、以下のボリュームグループには 8780 個のの空き物理エクステントがあります。

```
--- 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. 論理ボリュームを作成します。ボリュームサイズをバイトではなくエクステントに入力します。

例17.6 エクステントの数を指定して論理ボリュームを作成

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

例17.7 残りの領域をすべて使用する論理ボリュームの作成

または、論理ボリュームを拡張して、ボリュームグループ内の残りの空き領域の割合を使用できます。以下に例を示します。

```
# lvcreate --extents 100%FREE --name mylv myvg
```

検証手順

- ボリュームグループが使用するエクステントの数を確認します。

```
# 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 メタデータとそのディスク上の場所

LVM ヘッダーとメタデータ領域は、さまざまなオフセットとサイズで使用できます。

デフォルトの LVM ディスクヘッダー:

- **label_header** 構造と **pv_header** 構造にあります。
- ディスクの 2 番目の 512 バイトセクターにあります。なお、物理ボリューム (PV) の作成時にデフォルト以外の場所が指定された場合、ヘッダーは最初のセクターまたは 3 番目のセクターにある可能性があります。

標準の LVM メタデータ領域:

- ディスクの先頭から 4096 バイトの位置から開始します。
- ディスクの先頭から 1 MiB の位置で終了します。
- **mda_header** 構造を含む 512 バイトのセクターで開始します。

メタデータテキスト領域は、**mda_header** セクターの後に始まり、メタデータ領域の終わりまで続きます。LVM VG メタデータテキストは、メタデータテキスト領域に循環的に書き込まれます。**mda_header** は、テキスト領域内の最新の VG メタデータの場所を指します。

 # pvck --dump headers /dev/sda コマンドを使用して、ディスクから LVM ヘッダーを出力できます。このコマンドは、**label_header**、**pv_header**、**mda_header**、およびメタデータテキストが見つかった場合はその場所を出力します。正常でないフィールドは **CHECK** 接頭辞を付けて出力されます。

LVM メタデータ領域のオフセットは PV を作成したマシンのページサイズと一致するため、メタデータ領域はディスクの先頭から 8K、16K、または 64K の位置から開始する場合もあります。

PV の作成時に、より大きなメタデータ領域またはより小さなメタデータ領域を指定できます。その場合、メタデータ領域は 1 MiB 以外の位置で終了する可能性があります。**pv_header** は、メタデータ領域のサイズを指定します。

PV を作成するときに、必要に応じて 2 番目のメタデータ領域をディスクの末尾で有効にすることができます。**pv_header** にはメタデータ領域の場所が含まれます。

17.9. ディスクからの VG メタデータの抽出

状況に応じて、次のいずれかの手順を選択して、ディスクから VG メタデータを抽出します。抽出したメタデータを保存する方法については、抽出したメタデータのファイルへの保存 を参照してください。

注記

修復には、ディスクからメタデータを抽出せずに **/etc/lvm/backup/** にあるバックアップファイルを使用できます。

手順

- 有効な **mda_header** から参照される現在のメタデータテキストを出力します。

```
# pvck --dump metadata <disk>
```

例17.8 有効な mda_header からのメタデータテキスト

```
# pvck --dump metadata /dev/sdb
  metadata text at 172032 crc Oxc627522f # vgname test segno 59
  ---
  <raw metadata from disk>
  ---
```

- 有効な **mda_header** の検出に基づいて、メタデータ領域で見つかったすべてのメタデータコピーの場所を出力します。

```
# pvck --dump metadata_all <disk>
```

例17.9 メタデータ領域内のメタデータコピーの場所

```
# 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
```

- たとえば、ヘッダーが欠落しているか破損している場合は、**mda_header** を使用せずにメタデータ領域内のメタデータのすべてのコピーを検索します。

```
# pvck --dump metadata_search <disk>
```

例17.10 mda_header を使用しないメタデータ領域内のメタデータコピー

```
# 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
```

- メタデータの各コピーの説明を表示するには、**dump** コマンドに **-v** オプションを追加します。

```
# pvck --dump metadata -v <disk>
```

例17.11 各メタデータコピーの説明の表示

```
# pvck --dump metadata -v /dev/sdb
  metadata text at 199680 crc 0x628cf243 # vgname my_vg seqno 40
  ---
my_vg {
id = "dmEbPi-gsgx-VbvS-Uaia-HczM-iu32-Rb7iOf"
seqno = 40
format = "lvm2"
status = ["RESIZEABLE", "READ", "WRITE"]
flags = []
extent_size = 8192
max_lv = 0
max_pv = 0
metadata_copies = 0

physical_volumes {

pv0 {
id = "8gn0is-Hj8p-njgs-NM19-wuL9-mcB3-kUDiOQ"
device = "/dev/sda"

device_id_type = "sys_wwid"
device_id = "naa.6001405e635dbaab125476d88030a196"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}

pv1 {
id = "E9qChJ-5ElL-HVEp-rc7d-U5Fg-fHxL-2QLyID"
device = "/dev/sdb"

device_id_type = "sys_wwid"
device_id = "naa.6001405f3f9396fddcd4012a50029a90"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}
```

このファイルは修復に使用できます。最初のメタデータ領域は、デフォルトでダンプメタデータに使用されます。ディスクの末尾に 2 番目のメタデータ領域がある場合は、**--settings "mda_num=2"** オプションを使用して、代わりに 2 番目のメタデータ領域をダンプメタデータに使用できます。

17.10. 抽出したメタデータのファイルへの保存

修復のためにダンプされたメタデータを使用する必要がある場合は、**-f** オプションと **--settings** オプションを使用して、抽出したメタデータをファイルに保存する必要があります。

手順

- **-f <filename>** を **--dump metadata** に追加すると、指定されたファイルに raw メタデータが書き込まれます。このファイルは修復に使用できます。
- **-f <filename>** を **--dump metadata_all** または **--dump metadata_search** に追加すると、すべての場所の raw メタデータが指定されたファイルに書き込まれます。
- **--dump metadata_all|metadata_search** からメタデータテキストのインスタンスを 1 つ保存するには、**--settings "metadata_offset=<offset>"** を追加します。**<offset>** には、リスト表示された出力の "metadata at <offset>" の値を使用します。

  例17.12 コマンドの出力:

```
# 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. PVCREATE コマンドと VGCFGRESTORE コマンドを使用した、LVM ヘッダーとメタデータが破損したディスクの修復

破損した物理ボリューム、または新しいデバイスに置き換えられた物理ボリューム上のメタデータとヘッダーを復元できます。物理ボリュームのメタデータ領域を書き換えて、物理ボリュームからデータを復旧できる場合があります。

警告

これらの手順は、各コマンドの意味、現在のボリュームのレイアウト、実現する必要があるレイアウト、およびバックアップメタデータファイルの内容をよく理解している場合にのみ、細心の注意を払って使用する必要があります。これらのコマンドはデータを破損する可能性があるため、トラブルシューティングについては Red Hat グローバルサポートサービスに問い合わせることを推奨します。

前提条件

- 見つからない物理ボリュームのメタデータを特定している。詳細は、見つからない LVM 物理ボリュームのメタデータの検索 を参照してください。

手順

1. **pvcreate** および **vgcfgrestore** コマンドに必要な次の情報を収集します。**# pvs -o+uuid** コマンドを実行すると、ディスクと UUID に関する情報を収集できます。

- **metadata-file** は、VG の最新のメタデータバックアップファイルへのパスです (例: **/etc/lvm/backup/<vg-name>**)。
- **vg-name** は、破損または欠落している PV がある VG の名前です。
- このデバイスの破損した PV の **UUID** は、**# pvs -i+uuid** コマンドの出力から取得した値です。
- **disk** は、PV が配置されるディスクの名前です (例: **/dev/sdb**)。これが正しいディスクであることを確認するか、Red Hat サポートにお問い合わせください。正しいディスクでない場合、次の手順に従うとデータが失われる可能性があります。

2. ディスク上に LVM ヘッダーを再作成します。

```
# pvcreate --restorefile <metadata-file> --uuid <UUID> <disk>
```

必要に応じて、ヘッダーが有効であることを確認します。

```
# pvck --dump headers <disk>
```

3. ディスク上に VG メタデータを復元します。

```
# vgcfgrestore --file <metadata-file> <vg-name>
```

必要に応じて、メタデータが復元されていることを確認します。

```
# pvck --dump metadata <disk>
```

VG のメタデータバックアップファイルがない場合は、抽出したメタデータのファイルへの保存 の手順を使用して取得できます。

検証

- 新しい物理ボリュームが損傷しておらず、ボリュームグループが正しく機能していることを確認するには、次のコマンドの出力を確認します。

```
# vgs
```

関連情報

- **pvck (8) man ページ**
- 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. PVCK コマンドを使用した、LVM ヘッダーとメタデータが破損したディスクの修復

これは pvcreate コマンドと vgcfgrestore コマンドを使用した、LVM ヘッダーとメタデータが破損したディスクの修復 の代替手段です。**pvcreate** および **vgcfgrestore** コマンドが機能しない場合があります。この方法は、損傷したディスクにターゲットを絞っています。

この方法では、**pvck --dump** で抽出されたメタデータ入力ファイル、または **/etc/lvm/backup** のバックアップファイルを使用します。可能であれば、同じ VG 内の別の PV から、または PV 上の 2 番目のメタデータ領域から **pvck --dump** によって保存されたメタデータを使用します。詳細は、抽出したメタデータのファイルへの保存 を参照してください。

手順

- ディスク上のヘッダーとメタデータを修復します。

```
# pvck --repair -f <metadata-file> <disk>
```

  ここでは、以下のようになります。

  - **<metadata-file>** は、VG の最新のメタデータを含むファイルです。これは **/etc/lvm/backup/vg-name** にすることも、**pvck --dump metadata_search** コマンド出力から取得した、raw メタデータテキストを含むファイルにすることもできます。
  - **<disk>** は、PV が配置されるディスクの名前です (例: **/dev/sdb**)。データの損失を防ぐために、それが正しいディスクであることを確認してください。ディスクが正しいかどうかわからない場合は、Red Hat サポートにお問い合わせください。

注記

メタデータファイルがバックアップファイルの場合、VG にメタデータを保持する各 PV で **pvck --repair** を実行する必要があります。メタデータファイルが別の PV から抽出された raw メタデータである場合、**pvck --repair** は破損した PV でのみ実行する必要があります。

検証

- 新しい物理ボリュームが損傷しておらず、ボリュームグループが正しく機能していることを確認するには、次のコマンドの出力を確認します。

```
# vgs <vgname>
```

```
# pvs <pvname>
```

```
# lvs <lvname>
```

関連情報

- **pvck (8) man ページ**
- 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. LVM RAID のトラブルシューティング

LVM RAID デバイスのさまざまな問題のトラブルシューティングを実行して、データエラーの修正、デバイスの復旧、障害が発生したデバイスの置き換えを行うことができます。

17.13.1. RAID 論理ボリュームでのデータ整合性の確認

LVM は、RAID 論理ボリュームのスクラビングに対応します。RAID スクラビングは、アレイ内のデータおよびパリティーブロックをすべて読み込み、それが一貫しているかどうかを確認するプロセスです。**lvchange --syncactionrepair** コマンドは、アレイでバックグラウンドの同期アクションを開始します。次の属性は、データの整合性に関する詳細を提供します。

- **raid_sync_action** フィールドには、RAID 論理ボリュームが実行している現在の同期アクションが表示されます。値は次のいずれかです。

  idle

  すべての **sync** アクションが完了しました (何も実行していません)。

  resync

  マシンの不完全なシャットダウン後にアレイを初期化または再同期しています。

  recover

  アレイ内のデバイスを交換しています。

  check

  アレイの不一致を検索しています。

  repair

  不一致を検索して修復しています。

- **raid_mismatch_count** フィールドには、**check** アクション中に検出された不一致の数が表示されます。

- **Cpy%Sync** フィールドには、**sync** アクションの進行状況が表示されます。

- **lv_attr** フィールドは、追加のインジケーターを提供します。このフィールドのビット 9 は、論理ボリュームの正常性を示し、以下のインジケーターに対応しています。

  m または mismatches

  RAID 論理ボリュームに不一致があることを示します。この文字は、スクラビング操作によって RAID の一貫性のない部分が検出された後に表示されます。

  r または refresh

  LVM がデバイスラベルを読み取ることができ、デバイスが動作しているとみなされる場合でも、RAID アレイ内に障害が発生したデバイスがあることを示します。デバイスが利用可能になったことをカーネルに通知するように論理ボリュームを更新するか、デバイスに障害が発生したと思われる場合はデバイスを交換します。

手順

1. オプション: スクラビングプロセスが使用する I/O 帯域幅を制限します。RAID スクラビング操作を実行すると、**sync** アクションに必要なバックグラウンド I/O が、LVM デバイスへの他の I/O (ボリュームグループメタデータの更新など) よりも優先される可能性があります。これにより、他の LVM 操作が遅くなる可能性があります。
   リカバリースロットルを実装してスクラビング操作のレートを制御できます。**lvchange --syncaction** コマンドで **--maxrecoveryrate Rate[bBsSkKmMgG]** または **--minrecoveryrate Rate[bBsSkKmMgG]** を使用して復旧速度を設定できます。詳細は、最小/最大 I/O レートオプ

ション を参照してください。

Rate 値は、アレイ内の各デバイスに対する 1 秒あたりのデータ通信量を指定します。接尾辞を指定しないと、オプションはデバイスごとの 1 秒あたらりの kiB を想定します。

2. アレイ内の不一致数を修復せずに、アレイ内の不一致の数を表示します。

```
# lvchange --syncaction check my_vg/my_lv
```

このコマンドは、アレイでバックグラウンドの同期アクションを開始します。

3. オプション: **var/log/syslog** ファイルでカーネルメッセージを確認します。

4. アレイ内の不一致を修正します。

```
# lvchange --syncaction repair my_vg/my_lv
```

このコマンドは、RAID 論理ボリューム内の障害が発生したデバイスを修復または交換します。このコマンドを実行したら、**var/log/syslog** ファイルでカーネルメッセージを確認できます。

検証

1. スクラビング操作に関する情報を表示します。

```
# 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
```

関連情報

- **lvchange(8)** および **lvmraid(7)** man ページ
- 最小/最大 I/O レートオプション

17.13.2. 論理ボリュームに障害が発生した RAID デバイスの交換

RAID は従来の LVM ミラーリングとは異なります。LVM ミラーリングの場合は、障害が発生したデバイスを削除します。そうしないと、障害が発生したデバイスで RAID アレイが動作し続ける間、ミラーリングされた論理ボリュームがハングします。RAID1 以外の RAID レベルの場合、デバイスを削除すると、デバイスはより低いレベルの RAID に変換されます (たとえば、RAID6 から RAID5 へ、または RAID4 または RAID5 から RAID0 への変換)。

LVM では、障害が発生したデバイスを取り外して代替デバイスを割り当てる代わりに、**lvconvert** コマンドの **--repair** 引数を使用して、RAID 論理ボリューム内で物理ボリュームとして機能する障害が発生したデバイスを交換できます。

前提条件

- ボリュームグループには、障害が発生したデバイスを置き換えるのに十分な空き容量を提供する物理ボリュームが含まれています。
  ボリュームグループに十分な空きエクステントがある物理ボリュームがない場合は、**vgextend** ユーティリティーを使用して、十分なサイズの物理ボリュームを新たに追加します。

手順

手順

1. RAID 論理ボリュームを表示します。

```
# 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. **/dev/sdc** デバイスに障害が発生したら、RAID 論理ボリュームを表示します。

```
# 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. 障害が発生したデバイスを交換します。

```
# 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. オプション: 障害が発生したデバイスを置き換える物理ボリュームを手動で指定します。

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 代替の論理ボリュームを調べます。

```
# 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)
```

障害が発生したデバイスをボリュームグループから削除するまで、LVM ユーティリティーは、障害が発生したデバイスが見つけられないことを示しています。

6. 障害が発生したデバイスをボリュームグループから削除します。

```
# vgreduce --removemissing my_vg
```

検証

1. 障害が発生したデバイスを取り外した後、利用可能な物理ボリュームを表示します。

```
# 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. 障害が発生したデバイスを交換した後、論理ボリュームを調べます。

```
# 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)
```

関連情報

- **lvconvert(8)** および **vgreduce(8)** man ページ

17.14. マルチパス化された LVM デバイスに対する重複した物理ボリューム警告のトラブルシューティング

マルチパスストレージで LVM を使用する場合は、ボリュームグループまたは論理ボリュームのリストを表示する LVM コマンドを実行すると、以下のようなメッセージが表示される場合があります。

```
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
```

これらの警告のトラブルシューティングにより、LVM が警告を表示する理由を理解し、または警告を非表示にできます。

17.14.1. 重複した PV 警告の原因

Device Mapper Multipath (DM Multipath)、EMC PowerPath、または Hitachi Dynamic Link Manager (HDLM) などのマルチパスソフトウェアがシステム上のストレージデバイスを管理すると、特定の論理ユニット (LUN) への各パスが異なる SCSI デバイスとして登録されます。

マルチパスソフトウェアは、各パスにマップする新しいデバイスを作成します。各 LUN には、同じ基礎となるデータを参照する **/dev** ディレクトリーに複数のデバイスノードがあるため、すべてのデバイスノードには同じ LVM メタデータが含まれます。

表17.1 異なるマルチパスソフトウェアでのデバイスマッピングの例

マルチパスソフトウェア	LUN への SCSI パス	マルチパスデバイスパスへのマッピング
DM Multipath	**/dev/sdb** および **/dev/sdc**	**/dev/mapper/mpath1** または **/dev/mapper/mpatha**
EMC PowerPath		**/dev/emcpowera**
HDLM		**/dev/sddlmab**

複数のデバイスノードが原因で、LVM ツールは同じメタデータを複数回検出し、複製として報告します。

17.14.2. PV の重複警告が発生した場合

LVM は、以下のいずれかのケースで重複した PV 警告を表示します。

同じデバイスへの単一パス

出力に表示される 2 つデバイスは、両方とも同じデバイスへの単一パスです。
以下の例は、重複デバイスが、同じデバイスへの両方の単一パスである、重複した PV の警告を示しています。

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sdd not /dev/sdf
```

multipath -ll コマンドを使用して現在の DM Multipath トポロジーをリスト表示すると、同じマルチパスマップの下に **/dev/sdd** と **/dev/sdf** の両方を確認できます。

これらの重複メッセージは警告のみで、LVM 操作が失敗しているわけではありません。代わりに、LVM が物理ボリュームとしてデバイスのいずれかのみを使用して他を無視していることを警告します。

メッセージは、LVM が誤ったデバイスを選択するか、ユーザーが警告を中断していることを示す場合は、フィルターを適用できます。フィルターは、物理ボリュームに必要なデバイスのみを検索し、マルチパスデバイスへの基礎となるパスを省略するように LVM を設定します。その結果、警告が表示されなくなりました。

マルチパスマップ

出力に表示される 2 つのデバイスは、両方ともマルチパスマップです。

以下の例は、両方のマルチパスマップである 2 つのデバイスに対する重複した物理ボリューム警告を示しています。重複した物理ボリュームは、同じデバイスへの異なるパスではなく、2 つのデバイスに置かれます。

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/mapper/mpatha not
/dev/mapper/mpathc

Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowera not
/dev/emcpowerh
```

この状況は、同じデバイスへの両方の単一パスであるデバイスに対する重複する警告よりも複雑です。これらの警告は、多くの場合、マシンがアクセスできないデバイス (LUN クローンやミラーなど) にアクセスしていることを意味します。

マシンから削除するデバイスが分からないと、この状況は復旧できない可能性があります。Red Hat は、この問題に対処するために Red Hat テクニカルサポートにお問い合わせください。

17.14.3. PV の重複警告を防ぐ LVM デバイスフィルターの例

以下の例は、1 つの論理ユニット (LUN) への複数のストレージパスによって引き起こされる、重複した物理ボリュームの警告を回避する LVM デバイスフィルターを示しています。

論理ボリュームマネージャー (LVM) のフィルターを設定して、すべてのデバイスのメタデータをチェックできます。メタデータには、ルートボリュームグループを含むローカルハードディスクドライブとマルチパスデバイスが含まれます。マルチパスデバイスへの基礎となるパス (**/dev/sdb**、**/dev/sdd** など) を拒否すると、マルチパスデバイス自体で一意の各メタデータ領域が一度検出されるため、重複した物理ボリュームの警告を回避できます。

- 最初のハードディスクドライブ上の 2 番目のパーティションとデバイスマッパー (DM) マルチパスデバイスを許可し、それ以外をすべて拒否するには、次のように入力します。

  ```
  filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
  ```

- すべての HP SmartArray コントローラーと EMC PowerPath デバイスを許可するには、次のように入力します。

  ```
  filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|" ]
  ```

- 最初の IDE ドライブおよびマルチパスデバイス上のパーティションを許可するには、次のように入力します。

  ```
  filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
  ```

関連情報

- LVM デバイスフィルター設定の例

17.14.4. 関連情報

- LVM デバイスの可視性および使用を制限する
- LVM デバイスフィルター