Red Hat Ceph Storage 6

ブロックデバイスガイド

Red Hat Ceph Storage ブロックデバイスの管理、作成、設定、および使用

Last Updated: 2024-08-30

Red Hat Ceph Storage 6 ブロックデバイスガイド

Red Hat Ceph Storage ブロックデバイスの管理、作成、設定、および使用

法律上の通知

概要

本書では、Red Hat Ceph Storage ブロックデバイスを管理、作成、設定、および使用する方法を説明します。 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。

目次

第1章 CEPH ブロックデバイスの概要

ブロックは、シーケンスでのデータの長さ (例: 512 バイトのデータブロック) をバイト単位で設定したものです。多くのブロックを 1 つのファイルに統合すると、読み取り/書き込みが可能なストレージデバイスとして使用できます。以下のような回転メディアを使用してデータを保存する最も一般的な方法として、ブロックベースのストレージインターフェイスが挙げられます。

- ハードドライブ
- CD/DVD ディスク
- フロッピーディスク
- 従来の 9 トラックテープ

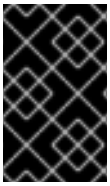
ブロックデバイスインターフェイスは偏在するので、Red Hat Ceph Storage などのマーシャルデータストレージシステムの操作には仮想ブロックデバイスが理想的です。

Ceph ブロックデバイスは、シンプロビジョニングされ、サイズ変更が可能で、Ceph Storage クラスター内の複数の Object Storage Devices (OSD) にストライプ化したストアデータです。Ceph ブロックデバイスは、Reliable Autonomic Distributed Object Store (RADOS) ブロックデバイス (RBD) としても知られています。Ceph ブロックデバイスは、以下のような RADOS 機能を利用します。

- スナップショット
- レプリケーション
- データの整合性

Ceph ブロックデバイスは、**librbd** ライブラリーを使用して OSD と対話します。

Ceph ブロックデバイスは、**libvirt** ユーティリティーおよび QEMU ユーティリティーに依存して Ceph ブロックデバイスと統合するために、Quick Emulator (QEMU) などの Kernel Virtual Machines (KVM) や OpenStack などのクラウドベースのコンピューティングシステムに、無限のスケーラビリティーと、高いパフォーマンスをもたらします。同じストレージクラスターを使用して、Ceph Object Gateway および Ceph ブロックデバイスを同時に運用できます。



重要

Ceph ブロックデバイスを使用するには、実行中の Ceph Storage クラスターにアクセスできる必要があります。Red Hat Ceph Storage クラスターのインストールの詳細は、**Red Hat Ceph Storage インストールガイド**を参照してください。

第2章 CEPH ブロックデバイス

ストレージ管理者は、Ceph のブロックデバイスコマンドについて理解しておくと、Red Hat Ceph Storage クラスターを効果的に管理しやすくなります。Ceph ブロックデバイスのさまざまな機能を有効または無効にしたり、ブロックデバイスのプールとイメージを作成および管理したりできます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

2.1. コマンドのヘルプの表示

コマンドラインインターフェイスから、コマンドとサブコマンドのヘルプを表示します。

注記

-h オプションは引き続き、使用できるすべてのコマンドのヘルプを表示します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. **rbd help** コマンドを使用して、特定の **rbd** コマンドとそのサブコマンドのヘルプを表示します。

   構文

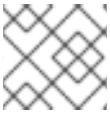
   ```
   rbd help COMMAND SUBCOMMAND
   ```

2. **snap list** コマンドのヘルプを表示するには、次のコマンドを実行します。

   ```
   [root@rbd-client ~]# rbd help snap list
   ```

2.2. ブロックデバイスプールの作成

ブロックデバイスクライアントを使用する前に、**rbd** のプールが存在し、初期化されていることを確認します。



注記

最初にプールを作成してから、これをソースとして指定する**必要があります**。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. **rbd** プールを作成するには、以下を実行します。

   構文

   ```
   ceph osd pool create POOL_NAME PG_NUM
   ceph osd pool application enable POOL_NAME rbd
   rbd pool init -p POOL_NAME
   ```

   例

   ```
   [root@rbd-client ~]# ceph osd pool create pool1
   [root@rbd-client ~]# ceph osd pool application enable pool1 rbd
   [root@rbd-client ~]# rbd pool init -p pool1
   ```

関連情報

- 詳細は、**Red Hat Ceph Storage ストラテジーガイド**の プール の章を参照してください。

2.3. ブロックデバイスイメージの作成

ブロックデバイスをノードに追加する前に、Ceph Storage クラスターにそのイメージを作成します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. ブロックデバイスイメージを作成するには、以下のコマンドを実行します。

   構文

   ```
   rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd create image1 --size 1024 --pool pool1
   ```

   以下の例では、**pool1** という名前のプールに情報を格納する **image1** という名前のイメージが 1 GB のサイズで作成されます。

   注記

   イメージを作成する前に、プールが存在することを確認します。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **ブロックデバイスプールの作成** セクションを参照してください。

2.4. ブロックデバイスイメージのリスト表示

ブロックデバイスイメージをリスト表示します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. **rbd** プールのブロックデバイスをリスト表示するには、以下を実行します。

注記

RBD はデフォルトのプール名です。

例

```
[root@rbd-client ~]# rbd ls
```

2. 特定のプールのブロックデバイスをリスト表示するには、以下を実行します。

構文

```
rbd ls POOL_NAME
```

例

```
[root@rbd-client ~]# rbd ls pool1
```

2.5. ブロックデバイスイメージ情報の取得

ブロックデバイスイメージに関する情報を取得します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. デフォルトの **rbd** プールの特定イメージから情報を取得するには、以下のコマンドを実行します。

構文

```
```

```
rbd --image IMAGE_NAME info
```

例

```
[root@rbd-client ~]# rbd --image image1 info
```

2. プール内のイメージから情報を取得するには、以下を実行します。

構文

```
rbd --image IMAGE_NAME -p POOL_NAME info
```

例

```
[root@rbd-client ~]# rbd --image image1 -p pool1 info
```

2.6. ブロックデバイスイメージのサイズ変更

Ceph ブロックデバイスイメージはシンプロビジョニングされています。データの保存を開始する前に、実際には物理ストレージを使用しません。ただし、**--size** オプションでは、設定する最大容量があります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

- デフォルトの **rbd** プールの Ceph ブロックデバイスイメージの最大サイズを増やすには、以下を実行します。

構文

```
rbd resize --image IMAGE_NAME --size SIZE
```

例

```
[root@rbd-client ~]# rbd resize --image image1 --size 1024
```

- デフォルトの **rbd** プールの Ceph ブロックデバイスイメージの最大サイズを減らすには、以下を実行します。

構文

```
rbd resize --image IMAGE_NAME --size SIZE --allow-shrink
```

例

```
[root@rbd-client ~]# rbd resize --image image1 --size 1024 --allow-shrink
```

- 特定のプールの Ceph ブロックデバイスイメージの最大サイズを増やすには、以下を実行します。

  構文

  ```
  rbd resize --image POOL_NAME/IMAGE_NAME --size SIZE
  ```

  例

  ```
  [root@rbd-client ~]# rbd resize --image pool1/image1 --size 1024
  ```

- 特定のプールの Ceph ブロックデバイスイメージの最大サイズを減らすには、以下を実行します。

  構文

  ```
  rbd resize --image POOL_NAME/IMAGE_NAME --size SIZE --allow-shrink
  ```

  例

  ```
  [root@rbd-client ~]# rbd resize --image pool1/image1 --size 1024 --allow-shrink
  ```

2.7. ブロックデバイスイメージの削除

ブロックデバイスイメージを削除します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. デフォルトの **rbd** プールからブロックデバイスを削除するには、次のコマンドを実行します。

   構文

   ```
   rbd rm IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd rm image1
   ```

2. 特定のプールからブロックデバイスを削除するには、次のコマンドを実行します。

   構文

   ```
   rbd rm IMAGE_NAME -p POOL_NAME
   ```

   例

```
[root@rbd-client ~]# rbd rm image1 -p pool1
```

2.8. ブロックデバイスイメージのゴミ箱への移行

RADOS Block Device (RBD) イメージは、**rbd trash** コマンドを使用してゴミ箱に移動できます。このコマンドは、**rbd rm** コマンドよりも多くのオプションがあります。

イメージをゴミ箱に移動すると、後でゴミ箱から取り除くこともできます。この機能により、誤って削除されるのを回避できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. イメージをゴミ箱に移動するには、以下のコマンドを実行します。

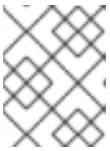
   構文

   ```
   rbd trash mv [POOL_NAME/] IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd trash mv pool1/image1
   ```

   イメージがゴミ箱に入ると、一意のイメージ ID が割り当てられます。

   注記

   ゴミ箱オプションのいずれかを使用する必要がある場合は、後でこのイメージを指定するのにこのイメージ ID が必要です。

2. ゴミ箱にあるイメージ ID のリストに対して **rbd trash list POOL_NAME** を実行します。このコマンドは、イメージの削除前の名前も返します。さらに、**rbd info** および **rbd snap** コマンドで使用可能な **--image-id** 引数 (任意) があります。**rbd info** コマンドに **--image-id** を使用し、ごみ箱の中にあるイメージのプロパティーを表示し、**rbd snap** で、イメージのスナップショットをゴミ箱から削除します。

3. ゴミ箱からイメージを削除するには、以下のコマンドを実行します。

   構文

   ```
   rbd trash rm [POOL_NAME/] IMAGE_ID
   ```

   例

   ```
   [root@rbd-client ~]# rbd trash rm pool1/d35ed01706a0
   ```

重要

- イメージがゴミ箱から削除されると、そのイメージは復元できません。
- ミラーリングが有効になっているイメージをゴミ箱に移動すると、セカンダリーサイトのイメージも削除されます。イメージをプライマリーサイトで復元した場合は、ミラーリングを再度有効にする必要があります。セカンダリーサイトはイメージを再度プルします。

4. **rbd trash restore** コマンドを実行して、イメージを復元します。

   構文

   ```
   rbd trash restore [POOL_NAME/] IMAGE_ID
   ```

   例

   ```
   [root@rbd-client ~]# rbd trash restore pool1/d35ed01706a0
   ```

5. ゴミ箱から期限切れのイメージをすべて削除するには、以下のコマンドを実行します。

   構文

   ```
   rbd trash purge POOL_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd trash purge pool1
   Removing images: 100% complete...done.
   ```

2.9. ゴミ箱の自動パージスケジュールの定義

プールでゴミ箱のパージ操作を定期的にスケジュールできます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. ゴミ箱のパージのスケジュールを追加するには、以下のコマンドを実行します。

   構文

   ```
   rbd trash purge schedule add --pool POOL_NAME INTERVAL
   ```

   例

   ```
   [ceph: root@host01 /]# rbd trash purge schedule add --pool pool1 10m
   ```

2. ゴミ箱のパージのスケジュールをリスト表示するには、以下のコマンドを実行します。

   構文

   ```
   rbd trash purge schedule ls --pool POOL_NAME
   ```

   例

   ```
   [ceph: root@host01 /]# rbd trash purge schedule ls --pool pool1
   every 10m
   ```

3. ゴミ箱のパージスケジュールの状態を把握するには、以下を実行します。

   例

   ```
   [ceph: root@host01 /]# rbd trash purge schedule status
   POOL   NAMESPACE   SCHEDULE  TIME
   pool1              2021-08-02 11:50:00
   ```

4. ゴミ箱のパージスケジュールを削除するには、以下のコマンドを実行します。

   構文

   ```
   rbd trash purge schedule remove --pool POOL_NAME INTERVAL
   ```

   例

   ```
   [ceph: root@host01 /]# rbd trash purge schedule remove --pool pool1 10m
   ```

2.10. イメージ機能の有効化および無効化

fast-diff、**exclusive-lock**、**object-map**、**deep-flatten** などのブロックデバイスイメージはデフォルトで有効です。これらのイメージ機能は、すでに存在するイメージに対して有効/無効を設定することができます。

注記

ディープフラット化 機能は、既存のイメージでのみ無効にできますが、有効化できません。**ディープフラット化** を使用するには、イメージ作成時に有効化します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. プール内の特定のイメージから情報を取得します。

   構文

   ■

```
rbd --image POOL_NAME/IMAGE_NAME info
```

例

```
[ceph: root@host01 /]# rbd --image pool1/image1 info
```

2. 機能を有効にします。

構文

```
rbd feature enable POOL_NAME/IMAGE_NAME FEATURE_NAME
```

a. **pool1** プールの **image1** イメージで **exclusive-lock** 機能を有効にするには、以下を実行します。

例

```
[ceph: root@host01 /]# rbd feature enable pool1/image1 exclusive-lock
```

重要

fast-diff および **object-map** 機能を有効にする場合には、オブジェクトマップを再構築します。

+ 構文

```
rbd object-map rebuild POOL_NAME/IMAGE_NAME
```

3. 機能を無効にします。

構文

```
rbd feature disable POOL_NAME/IMAGE_NAME FEATURE_NAME
```

a. **pool1** プールの **image1** イメージで **fast-diff** 機能を無効にするには、以下を実行します。

例

```
[ceph: root@host01 /]# rbd feature disable pool1/image1 fast-diff
```

2.11. イメージメタデータの使用

Ceph は、カスタムイメージメタデータをキーと値のペアとして追加することをサポートしています。ペアには厳密な形式がありません。

また、メタデータを使用して特定のイメージの RADOS Block Device (RBD) 設定パラメーターを設定することもできます。

rbd image-meta コマンドを使用して、メタデータと連携します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. 新しいメタデータのキー/値のペアを設定するには、以下を行います。

   構文

   ```
   rbd image-meta set POOL_NAME/IMAGE_NAME KEY VALUE
   ```

   例

   ```
   [ceph: root@host01 /]# rbd image-meta set pool1/image1 last_update 2021-06-06
   ```

   この例では、**pool1** プールの **image1** イメージの **last_update** キーを、**2021-06-06** 値に設定します。

2. キーの値を表示するには、次のコマンドを実行します。

   構文

   ```
   rbd image-meta get POOL_NAME/IMAGE_NAME KEY
   ```

   例

   ```
   [ceph: root@host01 /]# rbd image-meta get pool1/image1 last_update
   ```

   この例では、**last_update** キーの値を確認します。

3. イメージの全メタデータを表示するには、以下のコマンドを実行します。

   構文

   ```
   rbd image-meta list POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [ceph: root@host01 /]# rbd image-meta list pool1/image1
   ```

   この例では、**pool1** プールの **image1** イメージに設定されたメタデータをリスト表示しています。

4. メタデータのキー/値のペアを削除するには、以下を実行します。

   構文

   ```
   rbd image-meta remove POOL_NAME/IMAGE_NAME KEY
   ```

   例

```
[ceph: root@host01 /]# rbd image-meta remove pool1/image1 last_update
```

この例では、**pool1** プール内の **image1** イメージから **last_update** のキーと値のペアを削除します。

5. 特定のイメージの Ceph 設定ファイルに設定されている RBD イメージ設定を上書きするには、以下を実行します。

構文

```
rbd config image set POOL_NAME/IMAGE_NAME  PARAMETER VALUE
```

例

```
[ceph: root@host01 /]# rbd config image set pool1/image1 rbd_cache false
```

この例では、**pool1** プールの **image1** イメージの RBD キャッシュを無効にします。

関連情報

- 指定可能な設定オプションの一覧は、**Red Hat Ceph Storage ブロックデバイスガイド**の **ブロックデバイスの一般的なオプション** セクションを参照してください。

2.12. プール間のイメージの移動

同じクラスターにある異なるプール間で RADOS Block Device (RBD) イメージを移動できます。

このプロセスで、ソースイメージはすべてのスナップショット履歴を含めて、ターゲットイメージにコピーされます。また、オプションでスパースの保持に役立つソースイメージの親へのリンクもコピーされます。ソースイメージは読み取り専用で、ターゲットイメージは書き込み可能です。移行時に、ターゲットイメージがソースイメージにリンクされます。

このプロセスは、新規ターゲットイメージの使用中に、バックグラウンドで安全に実行できます。ただし、新規ターゲットのイメージが、イメージを使用するクライアントの参照先として更新されるように、準備手順前にターゲットイメージを使用してすべてのクライアントを停止してください。

重要

現時点では、**krbd** カーネルモジュールはライブ移行に対応していません。

前提条件

- ソースイメージを使用するすべてのクライアントを停止しておく。
- クライアントノードへの root レベルのアクセス。

手順

1. ソースおよびターゲットイメージをクロスリンクする新規ターゲットイメージを作成して、移行を準備します。

構文

```
rbd migration prepare SOURCE_IMAGE TARGET_IMAGE
```

以下を置き換えます。

- SOURCE_IMAGE: 移動するイメージの名前に置き換えます。POOL/IMAGE_NAME 形式を使用します。
- TARGET_IMAGE: 新しいイメージの名前。POOL/IMAGE_NAME 形式を使用します。

例

```
[root@rbd-client ~]# rbd migration prepare pool1/image1 pool2/image2
```

2. **作成** 予定の新しいターゲットイメージの状態を確認します。

構文

```
rbd status TARGET_IMAGE
```

例

```
[root@rbd-client ~]# rbd status pool2/image2
Watchers: none
Migration:
        source: pool1/image1 (5e2cba2f62e)
        destination: pool2/image2 (5e2ed95ed806)
        state: prepared
```

3. 必要に応じて、新規ターゲットイメージ名を使用してクライアントを再起動します。
4. ソースイメージをターゲットイメージにコピーします。

構文

```
rbd migration execute TARGET_IMAGE
```

例

```
[root@rbd-client ~]# rbd migration execute pool2/image2
```

5. 移行が完了したことを確認します。

例

```
[root@rbd-client ~]# rbd status pool2/image2
Watchers:
   watcher=1.2.3.4:0/3695551461 client.123 cookie=123
Migration:
        source: pool1/image1 (5e2cba2f62e)
        destination: pool2/image2 (5e2ed95ed806)
        state: executed
```

6. ソースとターゲットイメージ間のクロスリンクを削除して移行をコミットします。これにより、ソースイメージも削除されます。

   構文

   ```
   rbd migration commit TARGET_IMAGE
   ```

   例

   ```
   [root@rbd-client ~]# rbd migration commit pool2/image2
   ```

   ソースイメージが 1 つ以上のクローンの親である場合は、クローンイメージが使用されていないことを確認した後に **--force** オプションを使用します。

   例

   ```
   [root@rbd-client ~]# rbd migration commit pool2/image2 --force
   ```

7. 準備手順の後にクライアントを再起動しなかった場合は、新規ターゲットイメージ名を使用してクライアントを再起動します。

2.13. プールの移行

RADOS ブロックデバイス (RBD) イメージを移行またはコピーできます。

このプロセスでは、ソースイメージがエクスポートされてからインポートされます。

重要

ワークロードに RBD イメージ **のみ** が含まれている場合は、この移行プロセスを使用します。ワークロード内に **rados cppool** イメージを存在させることはできません。ワークロードに rados cppool イメージが存在する場合は、**ストレージストラテジーガイド** の プールの移行 を参照してください。

重要

エクスポートおよびインポートコマンドを実行するときは、関連する RBD イメージにアクティブな I/O がないことを確認してください。このプールの移行期間中は、運用を停止することを推奨します。

前提条件

- エクスポートおよびインポート中の RBD イメージ内のすべてのアクティブな I/O を停止する。
- クライアントノードへの root レベルのアクセス。

手順

- ボリュームを移行します。

  構文

```
rbd export volumes/VOLUME_NAME - | rbd import --image-format 2 -
volumes_new/VOLUME_NAME
```

例

```
[root@rbd-client ~]# rbd export volumes/volume-3c4c63e3-3208-436f-9585-fee4e2a3de16 - |
rbd import --image-format 2 - volumes_new/volume-3c4c63e3-3208-436f-9585-
fee4e2a3de16
```

- インポートまたはエクスポートにローカルドライブを使用する必要がある場合は、コマンドを分割して、最初にローカルドライブにエクスポートし、次にファイルを新しいプールにインポートできます。

  構文

  ```
  rbd export volume/VOLUME_NAME FILE_PATH
  rbd import --image-format 2 FILE_PATH volumes_new/VOLUME_NAME
  ```

  例

  ```
  [root@rbd-client ~]# rbd export volumes/volume-3c4c63e3-3208-436f-9585-fee4e2a3de16
  <path of export file>
  [root@rbd-client ~]# rbd import --image-format 2 <path> volumes_new/volume-3c4c63e3-
  3208-436f-9585-fee4e2a3de16
  ```

2.14. RBDMAP サービス

systemd ユニットファイル、**rbdmap.service** は、**ceph-common** パッケージに含まれています。**rbdmap.service** ユニットは、**rbdmap** シェルスクリプトを実行します。

このスクリプトは、1つ以上の RBD イメージの RADOS Block Device (RBD) のマッピングと解除を自動化しています。スクリプトはいつでも手動で実行できますが、通常のユースケースでは、システムの起動時に RBD イメージを自動的にマウントし、シャットダウン時にアンマウントします。スクリプトでは、RBD イメージをマウントする **map** またはマウントを解除する **unmap** のいずれか1つの引数を使用できます。。スクリプトは設定ファイルを解析します。デフォルトは **/etc/ceph/rbdmap** ですが、**RBDMAPFILE** という環境変数を使用して上書きできます。設定ファイルの各行は RBD イメージに対応します。

設定ファイルの形式は以下のようになります。

IMAGE_SPEC RBD_OPTS

ここで、IMAGE_SPEC は POOL_NAME / IMAGE_NAME、または IMAGE_NAME だけを指定します。IMAGE_NAME だけを指定する場合は、POOL_NAME は **rbd** に設定されます。RBD_OPTS は、基礎となる **rbd map** コマンドに渡すオプションのリストです。以下のパラメーターとその値は、コンマ区切りの文字列で指定する必要があります。

OPT1=VAL1,OPT2=VAL2,...,OPT_N=VAL_N

これにより、スクリプトは以下のような **rbd map** コマンドを実行します。

構文

```
rbd map POOLNAME/IMAGE_NAME --OPT1 VAL1 --OPT2 VAL2
```

注記

コンマまたは等価記号など、オプションおよび値の場合には、これらの値が置き換えられないように、単純にアポストロフィーを使用することができます。

成功すると、**rbd map** の操作はイメージを **/dev/rbd/rbdX** デバイスにマッピングします。この時点で、**udev** ルールがトリガーされ、分かりやすいデバイス名のシンボリックリンク (例: **/dev/rbd/POOL_NAME/IMAGE_NAME**) を作成し、実際のマップされたデバイスを参照します。マウントまたはマウント解除を行うには、わかりやすいデバイス名に対応するエントリーを **/etc/fstab** ファイルに指定する必要があります。RBD イメージの **/etc/fstab** エントリーを作成する場合は、**noauto** または **nofail** マウントオプションを指定することが推奨されます。これにより、init システムが、デバイスの作成前に、先にマウントするのを防ぎます。

関連情報

- 使用可能なオプションの全リストは、**rbd** の man ページを参照してください。

2.15. RBDMAP サービスの設定

起動時に RADOS Block Device (RBD)、またはシャットダウン時に RADOS Block Device (RBD) を自動的にマップしてマウントするか、マップとマウントを解除します。

前提条件

- マウントを実行するノードへの Root レベルのアクセス。
- **ceph-common** パッケージのインストール。

手順

1. **/etc/ceph/rbdmap** 設定ファイルを開いて編集します。
2. RBD イメージを設定ファイルに追加します。

   例

   ```
   foo/bar1    id=admin,keyring=/etc/ceph/ceph.client.admin.keyring
   foo/bar2
   id=admin,keyring=/etc/ceph/ceph.client.admin.keyring,options='lock_on_read,queue_depth=102
   4'
   ```

3. 設定ファイルに加えた変更を保存します。
4. RBD マッピングサービスを有効にします。

   例

   ```
   [root@client ~]# systemctl enable rbdmap.service
   ```

関連情報

- RBD システムサービスの詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **rbdmap サービス** セクションを参照してください。

2.16. 永続的な書き込みログキャッシュ

Red Hat Ceph Storage クラスターでは、永続的な書き込みログ (PWL) キャッシュが librbd ベースの RBD クライアントに対して永続的で耐障害性のあるライトバックキャッシュを提供します。

PWL キャッシュは、ログ順のライトバックデザインを使用して、チェックポイントを内部で維持し、クラスターにフラッシュされる書き込みが常にクラッシュします。クライアントキャッシュが完全になくなった場合、ディスクイメージには整合性がありますが、データが古くなったように見えます。キャッシュデバイスとして、PMEM(永続メモリー) や SSD(Solid State Disk) を使用して PWL キャッシュを使用することができます。

PMEM の場合、キャッシュモードはレプリカ書き込みログ (RWL) で、SSD の場合はキャッシュモードが SSD になります。現在、PWL キャッシュは RWL および SSD モードに対応しており、デフォルトでは無効になっています。

PWL キャッシュの主な利点は次のとおりです。

- PWL キャッシュは、キャッシュが満杯でない場合は、高パフォーマンスを提供できます。キャッシュが大きいほど、高パフォーマンスの期間が長くなります。
- PWL キャッシュは永続性を提供し、RBD キャッシュと比較してそれほど遅くはありません。RBD キャッシュは高速ですが揮発性で、データの順番と永続性を保証することはできません。
- キャッシュが満杯の定常状態では、性能はインフライト I/O の数に影響されます。たとえば、PWL は io_depth が低い場合には高い性能を発揮しますが、I/O 数が 32 を超えるような高い io_depth では、キャッシュがない場合よりも性能が悪くなることが多いです。

PMEM キャッシュのユースケースは以下のとおりです。

- RBD キャッシュとは異なり、PWL キャッシュには不揮発性の特性があり、データ損失を避けたいが高パフォーマンスを必要とするシナリオで使用されます。
- RWL モードは、低レイテンシーを提供します。バースト I/O の場合安定した低レイテンシーを実現し、安定した低レイテンシーへの要求が高いシナリオに適しています。
- また、RWL モードでは、I/O 深度が低い、またはインフライト I/O が多すぎない状況において、連続的に安定した高い性能向上を実現しています。

SSD キャッシュのユースケースは以下のとおりです。

- SSD モードの利点は RWL モードと似ています。SSD ハードウェアは比較的安価で一般的ですが、そのパフォーマンスは PMEM よりも若干低くなります。

2.17. 永続的な書き込みログキャッシュの制約

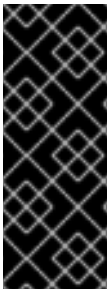
永続的な書き込みログ (PWL) キャッシュを使用する場合は、考慮する必要のあるいくつかの制約があります。

- 永続メモリー (PMEM) とソリッドステートディスク (SSD) では、根本的な実装が異なり、PMEM の方が高性能です。現在、PMEM は "書き込み時の永続性" を、SSD は "フラッシュまたはチェックポイントの永続性" を提供することができます。今後のリリースでは、この 2 つのモードが設定できるようになる予定です。

- ユーザーがイメージを頻繁に切り替え、開閉を繰り返すと、Ceph のパフォーマンスが低下します。PWL キャッシュが有効な場合には、パフォーマンスは悪化します。Flexible I/O (fio) テストで **num_jobs** を設定することは推奨されません。その代わりに、複数のジョブが異なるイメージを書き込むように設定します。

2.18. 永続的な書き込みログキャッシュの有効化

Ceph RADOS ブロックデバイス (RBD) の **rbd_persistent_cache_mode** および **rbd_plugins** オプションを設定すると、Red Hat Ceph Storage クラスターで永続的な書き込みログキャッシュ (PWL) を有効にすることができます。



重要

永続的な書き込みログキャッシュを有効にするには、排他的ロック機能を有効にする必要があります。キャッシュは、排他的ロックを取得した後にのみ読み込むことができます。排他的ロックは、**rbd_default_features** 設定オプションまたは **rbd create** コマンドの **--image-feature** フラグで上書きされない限り、新規に作成されたイメージでデフォルトで有効にされます。**exclusive-lock** 機能の詳細は、**イメージ機能の有効化および無効化** セクションを参照してください。

ceph config set コマンドを使用して、ホストレベルで永続的な書き込みログキャッシュオプションを設定します。プールまたはイメージレベルで永続的な書き込みログキャッシュオプションを設定するには、**rbd config pool set** または **rbd config image set** コマンドを使用します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- monitor ノードへのルートレベルのアクセス。
- 排他的ロック機能が有効になっている。
- クライアント側のディスクが永続メモリー (PMEM) またはソリッドステートディスク (SSD) である。
- RBD キャッシュが無効になっている。

手順

1. PWL キャッシュを有効にします。
   a. ホストレベルで、**ceph config set** コマンドを使用します。

      構文

      ```
      ceph config set client rbd_persistent_cache_mode CACHE_MODE
      ceph config set client rbd_plugins pwl_cache
      ```

      CACHE_MODE は **rwl** または **ssd** に置き換えます。

      例

      ```
      [ceph: root@host01 /]# ceph config set client rbd_persistent_cache_mode ssd
      [ceph: root@host01 /]# ceph config set client rbd_plugins pwl_cache
      ```

b. プールレベルで、**rbd config pool set** コマンドを使用します。

構文

```
rbd config pool set POOL_NAME rbd_persistent_cache_mode CACHE_MODE
rbd config pool set POOL_NAME rbd_plugins pwl_cache
```

CACHE_MODE は **rwl** または **ssd** に置き換えます。

例

```
[ceph: root@host01 /]# rbd config pool set pool1 rbd_persistent_cache_mode ssd
[ceph: root@host01 /]# rbd config pool set pool1 rbd_plugins pwl_cache
```

c. イメージレベルで、**rbd config image set** コマンドを使用します。

構文

```
rbd config image set POOL_NAME/IMAGE_NAME rbd_persistent_cache_mode
CACHE_MODE
rbd config image set POOL_NAME/IMAGE_NAME rbd_plugins pwl_cache
```

CACHE_MODE は **rwl** または **ssd** に置き換えます。

例

```
[ceph: root@host01 /]# rbd config image set pool1/image1 rbd_persistent_cache_mode
ssd
[ceph: root@host01 /]# rbd config image set pool1/image1 rbd_plugins pwl_cache
```

2. 必要に応じて、ホスト、プール、またはイメージレベルで追加の RBD オプションを設定します。

構文

```
rbd_persistent_cache_mode CACHE_MODE
rbd_plugins pwl_cache
rbd_persistent_cache_path /PATH_TO_CACHE_DIRECTORY (1)
rbd_persistent_cache_size PERSISTENT_CACHE_SIZE (2)
```

(1) **rbd_persistent_cache_path**: パフォーマンスの低下を回避するために **rwl** モードを使用する際に、DAX(ダイレクトアクセス) が有効にされている必要のあるデータをキャッシュするファイルフォルダー。

(2) **rbd_persistent_cache_size**: イメージごとのキャッシュサイズ。最小キャッシュサイズは 1 GB です。キャッシュサイズが大きいほど、パフォーマンスが向上します。

a. **rwl** モードの追加 RBD オプションの設定:

例

```
rbd_cache false
rbd_persistent_cache_mode rwl
```

```
rbd_plugins pwl_cache
rbd_persistent_cache_path /mnt/pmem/cache/
rbd_persistent_cache_size 1073741824
```

b. **ssd** モードの追加 RBD オプションの設定:

例

```
rbd_cache false
rbd_persistent_cache_mode ssd
rbd_plugins pwl_cache
rbd_persistent_cache_path /mnt/nvme/cache
rbd_persistent_cache_size 1073741824
```

関連情報

- DAX の使用の詳細は、**kernel.org** の Direct Access for files を参照してください。

2.19. 永続的な書き込みログキャッシュのステータス確認

永続的な書き込みログ (PWL) キャッシュのステータスを確認できます。キャッシュは、排他的ロックの取得時に使用され、排他的ロックの解放時に永続的な書き込みログキャッシュが終了します。キャッシュのステータスには、キャッシュサイズ、場所、タイプ、およびその他のキャッシュ関連情報に関する情報が表示されます。キャッシュの開放および終了時に、キャッシュステータスが更新されます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- monitor ノードへのルートレベルのアクセス。
- PWL キャッシュが有効になっている実行中のプロセス。

手順

- PWL キャッシュスのテータスを表示します。

構文

```
rbd status POOL_NAME/IMAGE_NAME
```

例

```
[ceph: root@host01 /]# rbd status pool1/image1
Watchers:
        watcher=10.10.0.102:0/1061883624 client.25496 cookie=140338056493088
Persistent cache state:
        host: host02
        path: /mnt/nvme0/rbd-pwl.rbd.101e5824ad9a.pool
        size: 1 GiB
        mode: ssd
        stats_timestamp: Mon Apr 18 13:26:32 2022
```

```
present: true    empty: false    clean: false
allocated: 509 MiB
cached: 501 MiB
dirty: 338 MiB
free: 515 MiB
hits_full: 1450 / 61%
hits_partial: 0 / 0%
misses: 924
hit_bytes: 192 MiB / 66%
miss_bytes: 97 MiB
```

2.20. 永続的な書き込みログキャッシュのフラッシュ

永続的な書き込みログ (PWL) キャッシュを破棄する前に、**rbd** コマンドを使用して、**persistent-cache flush**、プール名、およびイメージ名を指定して、キャッシュファイルをフラッシュできます。**flush** コマンドは、キャッシュファイルを OSD に明示的に書き戻すことができます。キャッシュの中断が発生した場合、またはアプリケーションが予期せず終了した場合、キャッシュ内のすべてのエントリーが OSD にフラッシュされるため、データを手動でフラッシュしてからキャッシュを **無効** にすることができます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- monitor ノードへのルートレベルのアクセス。
- PWL キャッシュが有効である。

手順

- PWL キャッシュをフラッシュします。

  構文

  ```
  rbd persistent-cache flush POOL_NAME/IMAGE_NAME
  ```

  例

  ```
  [ceph: root@host01 /]# rbd persistent-cache flush pool1/image1
  ```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **永続的な書き込みログキャッシュの破棄** セクションを参照してください。

2.21. 永続的な書き込みログキャッシュの破棄

キャッシュのデータの有効期限が切れている場合など、永続的な書き込みログ (PWL) キャッシュを手動で破棄する必要がある場合があります。**rbd image-cache invalidate** コマンドを使用すると、イメージのキャッシュファイルを破棄することができます。このコマンドは、指定されたイメージのキャッシュメタデータを削除し、キャッシュ機能を無効にし、ローカルキャッシュファイルが存在する場合は削除します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- monitor ノードへのルートレベルのアクセス。
- PWL キャッシュが有効である。

手順

- PWL キャッシュを破棄します。

  構文

  ```
  rbd persistent-cache invalidate POOL_NAME/IMAGE_NAME
  ```

  例

  ```
  [ceph: root@host01 /]# rbd persistent-cache invalidate pool1/image1
  ```

2.22. コマンドラインインターフェイスを使用した CEPH ブロックデバイスのパフォーマンスの監視

Red Hat Ceph Storage 4.1 以降、パフォーマンスメトリック収集フレームワークは、Ceph OSD および Manager コンポーネントに統合されます。このフレームワークには、他の Ceph ブロックデバイスのパフォーマンス監視ソリューションの構築時にパフォーマンスメトリックを生成して処理するための手段が含まれます。

新しい Ceph Manager モジュール **rbd_support** は、有効になっている場合にパフォーマンスメトリックを集約します。**rbd** コマンドには、**iotop** と **iostat** の新しい 2 つのアクションがあります。

注記

これらのアクションの初回使用じには、データフィールドの設定に約 30 秒かかります。

前提条件

- Ceph Monitor ノードへのユーザーレベルのアクセス。

手順

1. **rbd_support** Ceph Manager モジュールが有効であることを確認します。

   例

   ```
   [ceph: root@host01 /]# ceph mgr module ls

   {
       "always_on_modules": [
           "balancer",
           "crash",
           "devicehealth",
           "orchestrator",
   ```

```
            "pg_autoscaler",
            "progress",
            "rbd_support",   <--
            "status",
            "telemetry",
            "volumes"
    }
```

2. "iotop" スタイルのイメージを表示するには、以下のコマンドを実行します。

例

```
[user@mon ~]$ rbd perf image iotop
```

注記

ops、read-ops、write-bytes、read-bytes、write-latency、および read-latency の列は、右と左矢印キーを使用して動的にソートできます。

3. "iostat" スタイルのイメージを表示するには、以下を実行します。

例

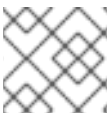
```
[user@mon ~]$ rbd perf image iostat
```

注記

このコマンドは JSON または XML 形式で出力でき、他のコマンドラインツールを使用してソートできます。

2.23. CEPH ユーザーおよびキーリング

cephx が有効な場合には、ユーザー名または ID とユーザーに対応するキーが含まれるキーリングへのパスを指定する必要があります。



注記

Cephx はデフォルトで有効化されています。

以下のパラメーターのエントリーを再追加しなくてもいいように、**CEPH_ARGS** 環境変数を追加することもできます。

構文

```
rbd --id USER_ID --keyring=/path/to/secret [commands]
rbd --name USERNAME --keyring=/path/to/secret [commands]
```

例

```
[root@rbd-client ~]# rbd --id admin --keyring=/etc/ceph/ceph.keyring [commands]
[root@rbd-client ~]# rbd --name client.admin --keyring=/etc/ceph/ceph.keyring [commands]
```

ヒント

ユーザーとシークレットを **CEPH_ARGS** 環境変数に追加して、毎回入力する必要がないようにします。

第3章 イメージのライブマイグレーション

ストレージ管理者は、RBD イメージのライブマイグレーションを、異なるプール間で行うことも、同じストレージクラスター内の同じプールで行うことも可能です。異なるイメージ形式やレイアウトの間や、外部データソースからも移行することができます。ライブマイグレーションが開始されると、ソースイメージは宛先イメージにディープコピーされ、可能な限りデータのスパース割り当てを維持しつつ、すべてのスナップショット履歴をプルします。

重要

現時点では、**krbd** カーネルモジュールはライブ移行に対応していません。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

3.1. ライブマイグレーションプロセス

デフォルトでは、同じストレージクラスターから RBD イメージのライブマイグレーション中には、ソースイメージに読み取り専用のマークが付けられます。すべてのクライアントは、Input/Output (I/O) を新規ターゲットイメージにリダイレクトします。また、このモードは、ソースイメージの親へのリンクを保存してスパース性を保持するか、移行中にイメージをフラット化してソースイメージの親の依存関係を削除することもできます。import-only モードでライブマイグレーションプロセスを使用できます。このモードでは、ソースイメージは変更されません。ターゲットイメージをバックアップファイル、HTTP (s) ファイル、または S3 オブジェクトなどの外部データソースにリンクできます。ライブマイグレーションのコピープロセスは、新規ターゲットイメージの使用中に、バックグラウンドで安全に実行できます。

ライブマイグレーションプロセスは、以下の 3 つのステップで構成されます。

移行の準備: 最初のステップでは、新規ターゲットイメージを作成し、ターゲットイメージをソースイメージにリンクします。import-only モードが設定されていない場合には、ソースイメージはターゲットイメージにもリンクされ、読み取り専用とマークされます。ターゲットイメージ内に初期化されていないデータエクステントの読み取りを試みると、そのソースイメージへの読み込みを内部的にリダイレクトし、ターゲットイメージ内で初期化されていないエクステントへの書き込みが行われ、重複するソースイメージエクステントをターゲットイメージに書き込みます。

Execute Migration: これはバックグラウンドで実行される操作で、ソースイメージからターゲットへの初期化されたすべてのブロックをディープコピーします。クライアントが新規ターゲットイメージをアクティブに使用している場合に、このステップを実行できます。

Finish Migration: バックグラウンドの移行プロセスが完了したら、移行をコミットまたは中止できます。移行をコミットすると、ソースとターゲットイメージ間の相互リンクが削除され、import-only モードで設定されていない場合にはソースイメージが削除されます。移行を中断すると、クロスリンクが削除され、その結果、ターゲットイメージが削除されます。

3.2. 形式

native 形式を使用して、Red Hat Ceph Storage クラスター内のネイティブ RBD イメージをソースイメージとして記述することができます。**source-spec** JSON ドキュメントは以下のようにエンコードされます。

構文

```
{
    "type": "native",
    "pool_name": "POOL_NAME",
    ["pool_id": "POOL_ID",] (optional, alternative to "POOL_NAME" key)
    ["pool_namespace": "POOL_NAMESPACE",] (optional)
    "image_name": "IMAGE_NAME>",
    ["image_id": "IMAGE_ID",] (optional, useful if image is in trash)
    "snap_name": "SNAP_NAME",
    ["snap_id": "SNAP_ID",] (optional, alternative to "SNAP_NAME" key)
}
```

ネイティブ Ceph 操作を使用するため、**native** フォーマットにはストリームオブジェクトは含まれません。たとえば、イメージの **rbd/ns1/image1@snap1** からインポートするには、**source-spec** を以下のようにエンコードできます。

例

```
{
"type": "native",
"pool_name": "rbd",
"pool_namespace": "ns1",
"image_name": "image1",
"snap_name": "snap1"
}
```

qcow 形式を使用して、QEMU コピーオンライト (QCOW) ブロックデバイスを記述できます。QCOW v1 および v2 形式はいずれも、圧縮、暗号化、バッキングファイル、外部データファイルなどの高度な機能を除き、現在サポートされています。**qcow** 形式データは、サポート対象のストリームソースにリンクできます。

例

```
{
    "type": "qcow",
    "stream": {
      "type": "file",
  "file_path": "/mnt/image.qcow"
    }
}
```

raw 形式を使用して、**rbd export –export-format 1 SNAP_SPEC** であるシックプロビジョニングされた、raw ブロックデバイスのエクスポートを記述できます。**raw** 形式データは、サポート対象のストリームソースにリンクできます。

例

```
{
    "type": "raw",
    "stream": {
      "type": "file",
      "file_path": "/mnt/image-head.raw"
    },
    "snapshots": [
        {
```

```
            "type": "raw",
            "name": "snap1",
            "stream": {
              "type": "file",
        "file_path": "/mnt/image-snap1.raw"
            }
         },
    ] (optional oldest to newest ordering of snapshots)
}
```

snapshots 配列の追加はオプションで、現在、シックプロビジョニングの raw スナップショットのエクスポートのみをサポートします。

3.3. ストリーム

ファイルストリーム

ファイル ストリームを使用して、ローカルでアクセス可能な POSIX ファイルソースからインポートできます。

構文

```
{
    <format unique parameters>
    "stream": {
        "type": "file",
        "file_path": "FILE_PATH"
    }
}
```

たとえば、**/mnt/image.raw** にあるファイルから raw 形式のイメージをインポートするには、**source-spec** JSON ファイルは以下のようになります。

例

```
{
    "type": "raw",
    "stream": {
        "type": "file",
        "file_path": "/mnt/image.raw"
    }
}
```

HTTP ストリーム

HTTP ストリームを使用して、リモートの HTTP または HTTPS Web サーバーからインポートできます。

構文

```
{
    <format unique parameters>
    "stream": {
        "type": "http",
```

```
        "url": "URL_PATH"
    }
}
```

たとえば、**http://download.ceph.com/image.raw** にあるファイルから raw 形式のイメージをインポートするには、**source-spec** JSON ファイルは以下のようになります。

例

```
{
    "type": "raw",
    "stream": {
        "type": "http",
        "url": "http://download.ceph.com/image.raw"
    }
}
```

S3 ストリーム

s3 ストリームを使用して、リモート S3 バケットからインポートできます。

構文

```
{
    <format unique parameters>
    "stream": {
        "type": "s3",
        "url": "URL_PATH",
        "access_key": "ACCESS_KEY",
        "secret_key": "SECRET_KEY"
    }
}
```

たとえば、**http://s3.ceph.com/bucket/image.raw** にあるファイルから raw 形式のイメージをインポートするには、以下のように source-spec JSON をエンコードします。

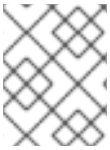
例

```
{
    "type": "raw",
    "stream": {
        "type": "s3",
        "url": "http://s3.ceph.com/bucket/image.raw",
        "access_key": "NX5QOQKC6BH2IDN8HC7A",
        "secret_key": "LnEsqNNqZIpkzauboDcLXLcYaWwLQ3Kop0zAnKIn"
    }
}
```

3.4. ライブマイグレーションプロセスの準備

同じ Red Hat Ceph Storage クラスター内にある RBD イメージのデフォルトのライブマイグレーションプロセスを作成できます。**rbd migration prepare** コマンドでは、**rbd create** コマンドと同じレイアウトオプションをすべて使用できます。**rbd create** コマンドでは、イミュータブルイメージのオンディスクレイアウトに変更を加えることができます。ディスク上のレイアウトのみを変更し、元のイメージ名

を維持する場合は、**migration_target** 引数を省略します。ライブマイグレーションを準備する前に、ソースイメージを使用するクライアントをすべて停止する必要があります。読み取り/書き込みモードでイメージが開いている稼働中のクライアントが検出された場合には、**prepare** の手順は失敗します。**prepare** 手順が完了したら、新しいターゲットイメージを使用してクライアントを再起動することができます。



注記

ソースイメージを使用してクライアントは再起動できないため、結果は失敗となります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ブロックデバイスプール 2 つ。
- ブロックデバイスイメージ 1 つ。

手順

1. ストレージクラスター内でライブマイグレーションを準備します。

   構文

   ```
   rbd migration prepare SOURCE_POOL_NAME/SOURCE_IMAGE_NAME
   TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   例

   ```
   [ceph: root@rbd-client /]# rbd migration prepare sourcepool1/sourceimage1
   targetpool1/sourceimage1
   ```

   OR

   ソースイメージの名前を変更する場合は、以下のコマンドを実行します。

   構文

   ```
   rbd migration prepare SOURCE_POOL_NAME/SOURCE_IMAGE_NAME
   TARGET_POOL_NAME/NEW_SOURCE_IMAGE_NAME
   ```

   例

   ```
   [ceph: root@rbd-client /]# rbd migration prepare sourcepool1/sourceimage1
   targetpool1/newsourceimage1
   ```

   この例では、**newsourceimage1** は名前が変更されたソースイメージです。

2. 次のコマンドを使用すると、ライブマイグレーションプロセスの現在の状態を確認できます。

   構文

```
rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

例

```
[ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
Watchers: none
Migration:
source: sourcepool1/sourceimage1 (adb429cb769a)
destination: targetpool2/testimage1 (add299966c63)
state: prepared
```

重要

移行プロセスで、ソースイメージは RBD ゴミ箱に移動され、誤用を回避します。

例

```
[ceph: root@rbd-client /]# rbd info sourceimage1
rbd: error opening image sourceimage1: (2) No such file or directory
```

例

```
[ceph: root@rbd-client /]# rbd trash ls --all sourcepool1
adb429cb769a sourceimage1
```

3.5. IMPORT-ONLY 移行の準備

--import-only オプションと、**--source-spec** か、**--source-spec-path** のオプションを指定して、**rbd migration prepare** コマンドを実行して、**import-only** のライブマイグレーションプロセスを開始し、コマンドラインまたはファイルから直接ソースイメージデータにアクセスする方法を記述した JSON ドキュメントを渡します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- バケットおよび S3 オブジェクトが作成されます。

手順

1. JSON ファイルを作成します。

例

```
[ceph: root@rbd-client /]# cat testspec.json
 {
   "type": "raw",
    "stream": {
        "type": "s3",
        "url": "http:10.74.253.18:80/testbucket1/image.raw",
```

```
        "access_key": "RLJOCP6345BGB38YQXI5",
        "secret_key": "oahWRB2ote2rnLy4dojYjDrsvaBADriDDgtSfk6o"
    }
```

2. **import-only** ライブマイグレーションプロセスを準備します。

   構文

   ```
   rbd migration prepare --import-only --source-spec-path "JSON_FILE"
   TARGET_POOL_NAME
   ```

   例

   ```
   [ceph: root@rbd-client /]# rbd migration prepare --import-only --source-spec-path
   "testspec.json" targetpool1
   ```

   注記

   rbd migration prepare コマンドでは、**rbd create** コマンドと同じイメージオプションをすべて使用できます。

3. **import-only** ライブマイグレーションのステータスを確認できます。

   例

   ```
   [ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
   Watchers: none
   Migration:
   source: {"stream":
   {"access_key":"RLJOCP6345BGB38YQXI5","secret_key":"oahWRB2ote2rnLy4dojYjDrsvaBAD
   riDDgtSfk6o","type":"s3","url":"http://10.74.253.18:80/testbucket1/image.raw"},"type":"raw"}
   destination: targetpool1/sourceimage1 (b13865345e66)
   state: prepared
   ```

3.6. ライブマイグレーションプロセスの実行

ライブマイグレーションを準備したら、イメージブロックをソースイメージからターゲットイメージにコピーする必要があります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ブロックデバイスプール 2 つ。
- ブロックデバイスイメージ 1 つ。

手順

1. ライブマイグレーションを実行します。

   構文

```
rbd migration execute TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

例

```
[ceph: root@rbd-client /]# rbd migration execute targetpool1/sourceimage1
Image migration: 100% complete...done.
```

2. 移行ブロックのディープコピーの進捗に関するフィードバックを確認できます。

構文

```
rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

例

```
[ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
Watchers: none
Migration:
source: sourcepool1/testimage1 (adb429cb769a)
destination: targetpool1/testimage1 (add299966c63)
state: executed
```

3.7. ライブマイグレーションプロセスのコミット

ライブマイグレーションで、ソースイメージからターゲットイメージへのディープコピーを完了したら、移行をコミットできます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ブロックデバイスプール 2 つ。
- ブロックデバイスイメージ 1 つ。

手順

1. ディープコピーが完了したら、移行をコミットします。

構文

```
rbd migration commit TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

例

```
[ceph: root@rbd-client /]# rbd migration commit targetpool1/sourceimage1
Commit image migration: 100% complete...done.
```

検証

ライブマイグレーションをコミットすると、ソースとターゲットイメージ間のクロスリンクが削除され、ソースプールからソースイメージも削除されます。

例

```
[ceph: root@rbd-client /]# rbd trash list --all sourcepool1
```

3.8. ライブマイグレーションプロセスの中断

ライブマイグレーションプロセスは、元に戻すことができます。ライブマイグレーションを中断すると、準備と実行の手順を元に戻します。

注記

ライブマイグレーションがコミットされていない場合に限り、中止できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ブロックデバイスプール 2 つ。
- ブロックデバイスイメージ 1 つ。

手順

1. ライブマイグレーションプロセスを中断します。

   構文

   ```
   rbd migration abort TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   例

   ```
   [ceph: root@rbd-client /]# rbd migration abort targetpool1/sourceimage1
   Abort image migration: 100% complete...done.
   ```

検証

ライブマイグレーションプロセスを中断すると、ターゲットイメージが削除され、元のソースイメージへのアクセスがソースプールで復元されます。

例

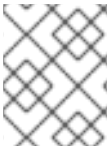
```
[ceph: root@rbd-client /]# rbd ls sourcepool1
sourceimage1
```

第4章 イメージの暗号化

ストレージ管理者は、特定の RBD イメージの暗号化に使用されるシークレットキーを設定できます。イメージレベルの暗号化は、RBD クライアントによって内部に処理されます。

注記

krbd モジュールはイメージレベルの暗号化に対応していません。



注記

dm-crypt または **QEMU** などの外部ツールを使用して、RBD イメージを暗号化できます。

前提条件

- Red Hat Ceph Storage 6 クラスターが実行されている。
- **ルート** レベルのパーミッション。

4.1. 暗号化形式

RBD イメージは、デフォルトでは暗号化されません。RBD イメージは、サポート対象の暗号化形式の 1 つにフォーマットすることで暗号化できます。フォーマット操作を行うと、暗号化メタデータを RBD イメージに永続化します。暗号化メタデータには、暗号化形式、バージョン、暗号アルゴリズム、モードの仕様などの情報や、暗号化キーのセキュリティー保護に使用する情報が含まれます。

暗号化鍵は、ユーザーが保存するシークレット (パスフレーズ) で保護されますが、RBD イメージには永続データとして保存されません。暗号化形式の操作では、暗号化形式、暗号アルゴリズム、およびモードの仕様、パスフレーズの指定が必要になります。暗号化メタデータは RBD イメージに保存されます。現在、raw イメージの起動時に書き込まれる暗号化ヘッダーとして保存されます。つまり、暗号化されたイメージの有効なイメージサイズは、raw イメージのサイズよりも小さくなります。

注記

明示的に (再) フォーマットされない限り、暗号化されたイメージのクローンは、同じ形式とシークレットを使用して本質的に暗号化されます。

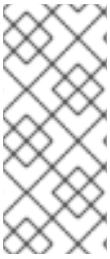
注記

フォーマット前の RBD イメージへの書き込みデータはいずれも、ストレージリソースを引き続き使用している場合でも、読み取りできなくなる可能性があります。ジャーナル機能が有効になっている RBD イメージは、暗号化できません。

4.2. 暗号化ロード

デフォルトでは、すべての RBD API は、暗号化されていない RBD イメージと同じように、暗号化 RBD イメージを処理します。イメージ内にある raw データはどこにある場合でも読み取りまたは書き込みが可能です。Raw データをイメージに書き込むと、暗号化形式の整合性が確保できなくなる可能性があります。たとえば、raw データは、イメージの最初にある暗号化メタデータを上書きする可能性があります。暗号化された RBD イメージで暗号化された入力/Outout (I/O) またはメンテナンス操作を安全に実行するには、イメージを開いてからすぐに、追加の暗号化ロード操作を適用する必要があります。

暗号化ロード操作では、暗号化形式と、イメージ自体の暗号化キーのロックを解除するためのパスフレーズと、明示的にフォーマットされた各先祖イメージを指定する必要があります。開いた RBD イメージの I/O はすべて、クローン作成された RBD イメージ用に暗号化または復号化されます (親イメージの IO を含む)。暗号化キーは、イメージを終了するまで RBD クライアントによってメモリーに保存されます。



注記

暗号化が RBD イメージに読み込まれたら、他の暗号化ロードやフォーマットの操作は適用できません。さらに、開いているイメージコンテキストを使用して RBD イメージサイズと親のオーバーラップを取得する API コールは、それぞれ有効イメージサイズと有効な親のオーバーラップを返します。**rbd-nbd** を介して RBD イメージをブロックデバイスとしてマッピングするときに、暗号化が自動的に読み込まれます。

注記

開いているイメージコンテキストを使用してイメージサイズと親のオーバーラップを取得する API コールは、有効なイメージサイズと有効な親のオーバーラップを返します。

注記

暗号化されたイメージのクローンが明示的にフォーマットされた場合、親スナップショットから親データをコピーする際にクローンイメージのフォーマットに従って親データを再暗号化する必要があるため、クローン作成されたイメージのフラット化または縮小は透過的でなくなります。フラット化操作を実行する前に暗号化がロードされていないと、クローン作成されたイメージで以前アクセス可能だった親データが読み取れなくなる可能性があります。

注記

暗号化されたイメージのクローンが明示的にフォーマットされた場合、クローン作成されたイメージを縮小する操作は透過的でなくなります。これは、クローン作成されたイメージにスナップショットが含まれていたり、クローン作成されたイメージがオブジェクトサイズに合わないサイズに縮小されていたりするようなシナリオでは、フラット化と同様に、親スナップショットから一部のデータをコピーするアクションが発生するためです。縮小操作を実行する前に暗号化がロードされていないと、クローン作成されたイメージで以前アクセス可能だった親データが読み取れなくなる可能性があります。

4.3. サポート対象の形式

LUKS (Linux Unified Key Setup) 1 と 2 の両方がサポートされます。データレイアウトは、LUKS 仕様に完全に準拠しています。**dm-crypt** または **QEMU** などの外部の LUKS 互換ツールは、暗号化された RBD イメージ上で、暗号化された Input/Outout (I/O) を安全に実行できます。さらに、raw LUKS データを RBD イメージにコピーして、外部ツールが作成した既存の LUKS イメージをインポートすることもできます。

現在、Advanced Encryption Standards (AES) 128 および 256 暗号化アルゴリズムのみがサポートされています。暗号化モードで唯一サポートされているのは現時点では、xts-plain64 のみです。

LUKS 形式を使用するには、以下のコマンドで RBD イメージをフォーマットします。

注記

passphrase.txt という名前のファイルを作成し、パスフレーズを入力する必要があります。パスフレーズをランダムに生成することができます。これには NULL 文字が含まれる可能性があります。パスフレーズの末尾が改行文字の場合、その改行文字は削除されます。

構文

```
rbd encryption format POOL_NAME/LUKS_IMAGE luks1|luks2 PASSPHRASE_FILE
```

例

```
[ceph: root@host01 /]# rbd encryption format pool1/luksimage1 luks1 passphrase.bin
```

注記

luks1 または **luks** の暗号化形式のいずれかを選択できます。

暗号化形式の操作では LUKS ヘッダーを生成し、RBD イメージの最初に作成します。キースロットが 1 つ、ヘッダーに追加されます。キースロットには無作為に生成される暗号鍵が格納され、このキースロットはパスフレーズファイルから読み込むパスフレーズで保護されます。デフォルトでは、xts-plain64 モードの AES-256 (現在の推奨モード) および他の LUKS ツールのデフォルトが使用されます。現在、別のパスフレーズの追加または削除はネイティブにはサポートされていませんが、**cryptsetup** などの LUKS ツールを使用して実現できます。LUKS ヘッダーのサイズは、LUKS で最大 136MiB によって異なりますが、通常最大 16MiB です (インストールされている **libcryptsetup** のバージョンにより異なる)。暗号化フォーマットは、イメージオブジェクトサイズに合わせてデータオフセットを設定し、パフォーマンスを最適化します。たとえば、8MiB オブジェクトサイズで設定されたイメージを使用する場合には、オーバーヘッドが最低でも 8MiB 必要です。

LUKS1 では、最小暗号化ユニットであるセクターが 512 バイトに固定されています。LUKS2 はサイズの大きいセクターに対応しており、デフォルトのセクターサイズは最大 4KiB に設定され、パフォーマンスの向上を図ります。セクターよりも小さい書き込み、またはセクターの開始位置が揃っていない書き込みは、クライアント上で保護された **read-modify-write** チェーンをトリガーします。この際、レイテンシーのペナルティーが大きくなります。書き込みのバッチが整列されていない場合には、I/O 競合が発生し、さらにパフォーマンスが低下する可能性があります。Red Hat は、受信書き込みが LUKS セクターに合わせて確保できない場合に、RBD 暗号化の使用を回避することを推奨します。

LUKS 暗号化イメージをマッピングするには、次のコマンドを実行します。

構文

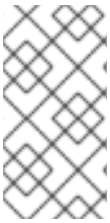
```
rbd device map -t nbd -o encryption-format=luks1|luks2,encryption-passphrase-file=passphrase.txt
POOL_NAME/LUKS_IMAGE
```

例

```
[ceph: root@host01 /]# rbd device map -t nbd -o encryption-format=luks1,encryption-passphrase-
file=passphrase.txt pool1/luksimage1
```

注記

luks1 または **luks2** の暗号化形式のいずれかを選択できます。



注記

セキュリティー上の理由から、暗号化フォーマットと暗号化ロード操作は CPU に負荷がかかるので、完了するまでに数秒かかることがあります。I/O が暗号化されている場合には、AES-NI が有効になっていると、マイクロ秒単位のレイテンシーが追加され、CPU 使用率が若干増加する可能性があります。

4.4. イメージおよびクローンへの暗号化形式の追加

階層化されたクライアント側の暗号化がサポートされています。クローン作成されたイメージは、親イメージとは異なる独自の形式とパスフレーズで暗号化できます。

rbd encryption format コマンドを使用して、暗号化形式をイメージおよびクローンに追加します。LUKS2 形式のイメージを使用すると、LUKS2 形式のクローンと LUKS1 形式のクローンの両方を作成できます。

前提条件

- ブロックデバイス (RBD) が設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

1. LUKS2 形式のイメージを作成します。

   構文

   ```
   rbd create --size SIZE POOL_NAME/LUKS_IMAGE
   rbd encryption format POOL_NAME/LUKS_IMAGE luks1|luks2 PASSPHRASE_FILE
   rbd resize --size 50G --encryption-passphrase-file PASSPHRASE_FILE
   POOL_NAME/LUKS_IMAGE
   ```

   例

   ```
   [ceph: root@host01 /]# rbd create --size 50G mypool/myimage
   [ceph: root@host01 /]# rbd encryption format mypool/myimage luks2 passphrase.txt
   [ceph: root@host01 /]# rbd resize --size 50G --encryption-passphrase-file passphrase.txt
   mypool/myimage
   ```

   rbd resize コマンドは、イメージを拡張して LUKS2 ヘッダーに関連するオーバーヘッドを補います。

2. LUKS2 形式のイメージを使用して、同じ有効サイズの LUKS2 形式のクローンを作成します。

   構文

   ```
   rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
   rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
   rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
   ```

```
rbd encryption format POOL_NAME/CLONE_NAME luks1 CLONE_PASSPHRASE_FILE
```

例

```
[ceph: root@host01 /]# rbd snap create mypool/myimage@snap
[ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
[ceph: root@host01 /]# rbd clone mypool/myimage@snap mypool/myclone
[ceph: root@host01 /]# rbd encryption format mypool/myclone luks1 clone-passphrase.bin
```

3. LUKS2 形式のイメージを使用して、同じ有効サイズの LUKS1 形式のクローンを作成します。

構文

```
rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
rbd encryption format POOL_NAME/CLONE_NAME luks1 CLONE_PASSPHRASE_FILE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file
CLONE_PASSPHRASE_FILE --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME/CLONE_NAME
```

例

```
[ceph: root@host01 /]# rbd snap create mypool/myimage@snap
[ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
[ceph: root@host01 /]# rbd clone mypool/myimage@snap mypool/myclone
[ceph: root@host01 /]# rbd encryption format mypool/myclone luks1 clone-passphrase.bin
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
clone-passphrase.bin --encryption-passphrase-file passphrase.bin mypool/myclone
```

LUKS1 のヘッダーは通常 LUKS2 のヘッダーよりも小さいため、最後の **rbd resize** コマンドにより、クローン作成したイメージを縮小し、不要な余裕領域を取り除きます。

4. LUKS1 形式のイメージを使用して、同じ有効サイズの LUKS2 形式のクローンを作成します。

構文

```
rbd resize --size SIZE POOL_NAME/LUKS_IMAGE
rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
rbd encryption format POOL_NAME/CLONE_NAME luks2 CLONE_PASSPHRASE_FILE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME/LUKS_IMAGE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file
CLONE_PASSPHRASE_FILE --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME_/CLONE_NAME
```

例

```
[ceph: root@host01 /]# rbd resize --size 51G mypool/myimage
[ceph: root@host01 /]# rbd snap create mypool/myimage@snap
[ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
```

```
[ceph: root@host01 /]# rbd clone mypool/my-image@snap mypool/myclone
[ceph: root@host01 /]# rbd encryption format mypool/myclone luks2 clone-passphrase.bin
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
passphrase.bin mypool/myimage
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
clone-passphrase.bin --encryption-passphrase-file passphrase.bin mypool/myclone
```

LUKS2 のヘッダーは通常 LUKS1 のヘッダーよりも大きいため、最初の **rbd resize** コマンドにより、親イメージを一時的に拡張し、親スナップショットおよびクローン作成されたイメージに追加領域を予約します。これは、クローン作成されたイメージのすべての親データにアクセスできるようにするために必要です。最後の **rbd resize** コマンドは、親イメージを元のサイズに縮小し、未使用の予約領域を取り除きます。親スナップショットとクローン作成されたイメージには影響を与えません。

フォーマットされていないイメージにはヘッダーがないため、フォーマットされていないイメージのフォーマットされたクローンの作成にも同様のことが適用されます。

関連情報

- クライアントを **cephadm-ansible** インベントリーに追加する方法は、**Red Hat Ceph Storage インストールガイド** の Ansible インベントリー場所の設定 セクションを参照してください。

第5章 スナップショット管理

ストレージ管理者は、Ceph のスナップショット機能を十分に理解している場合には、Red Hat Ceph Storage クラスターに保存されているイメージのスナップショットの管理や、クローン作成に役立ちます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

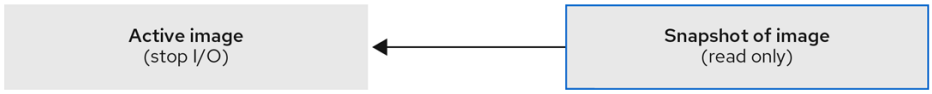
5.1. CEPH ブロックデバイスのスナップショット

スナップショットは、特定の時点におけるイメージの状態の読み取り専用コピーです。Ceph ブロックデバイスの高度な機能の 1 つとして、イメージのスナップショットを作成して、イメージの状態の履歴を保持できることが挙げられます。Ceph は、スナップショット階層機能もサポートしており、仮想マシンイメージなどのイメージのクローンをすばやく簡単に作成できます。Ceph は、**QEMU**、**libvirt**、OpenStack、および CloudStack など、**rbd** コマンドと、より上層レベルのインターフェイスを使用するブロックデバイススナップショットをサポートします。

注記

I/O の発生中にスナップショットが作成された場合、スナップショットは正確なイメージデータまたは最新のイメージデータを取得できず、マウントできる新規イメージに、スナップショットをクローンする必要がある場合があります。Red Hat は、イメージのスナップショットを作成する前に **I/O** を停止することを推奨します。イメージにファイルシステムが含まれる場合に、ファイルシステムはスナップショットの作成前に整合性のある状態でなければなりません。**I/O** を停止するには、**fsfreeze** コマンドを使用します。仮想マシンの場合には、**qemu-guest-agent** を使用してスナップショットの作成時にファイルシステムを自動的にフリーズできます。

図5.1 Ceph Block デバイスのスナップショット



関連情報

- 詳細は、 **fsfreeze(8)** の man ページを参照してください。

5.2. ブロックデバイススナップショットの作成

Ceph ブロックデバイスのスナップショットを作成します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. **snap create** オプション、プール名、およびイメージ名を指定します。

   - 方法 1:

     構文

     ```
     rbd --pool POOL_NAME snap create --snap SNAP_NAME IMAGE_NAME
     ```

     例

     ```
     [root@rbd-client ~]# rbd --pool pool1 snap create --snap snap1 image1
     ```

   - 方法 2:

     構文

     ```
     rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
     ```

     例

     ```
     [root@rbd-client ~]# rbd snap create pool1/image1@snap1
     ```

5.3. ブロックデバイススナップショットのリスト表示

ブロックデバイスのスナップショットをリスト表示します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. プール名とイメージ名を指定します。

   構文

   ```
   rbd --pool POOL_NAME --image IMAGE_NAME snap ls
   rbd snap ls POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 --image image1 snap ls
   [root@rbd-client ~]# rbd snap ls pool1/image1
   ```

5.4. ブロックデバイススナップショットのロールバック

ブロックデバイスのスナップショットをロールバックします。

注記

イメージをスナップショットにロールバックすると、イメージの現行バージョンがスナップショットからのデータで上書きされます。ロールバックの実行にかかる時間は、イメージのサイズとともに増加します。スナップショットにイメージを **ロールバック** するよりも、**クローンするほうが短時間ででき**、既存の状態戻す方法として推奨の方法です。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. **snap rollback** オプション、プール名、イメージ名、および snap 名を指定します。

   構文

   ```
   rbd --pool POOL_NAME snap rollback --snap SNAP_NAME IMAGE_NAME
   rbd snap rollback POOL_NAME/IMAGE_NAME@SNAP_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap rollback --snap snap1 image1
   [root@rbd-client ~]# rbd snap rollback pool1/image1@snap1
   ```

5.5. ブロックデバイススナップショットの削除

Ceph ブロックデバイスのスナップショットを削除します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. ブロックデバイスのスナップショットを削除するには、**snap rm** オプション、プール名、イメージ名、およびスナップショット名を指定します。

   構文

   ```
   rbd --pool POOL_NAME snap rm --snap SNAP_NAME IMAGE_NAME
   rbd snap rm POOL_NAME-/IMAGE_NAME@SNAP_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap rm --snap snap2 image1
   [root@rbd-client ~]# rbd snap rm pool1/image1@snap1
   ```

重要

イメージにクローンがある場合には、クローン作成されたイメージは、親イメージのスナップショットへの参照を保持します。親イメージのスナップショットを削除するには、最初に子イメージをフラット化する必要があります。

注記

Ceph OSD デーモンはデータを非同期的に削除するため、スナップショットを削除してもディスク領域がすぐに解放されません。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の クローンしたイメージのフラット化 を参照してください。

5.6. ブロックデバイススナップショットのパージ

ブロックデバイススナップショットをパージします。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. **snap purge** オプションと、特定のプールにイメージ名を指定します。

   構文

   ```
   rbd --pool POOL_NAME snap purge IMAGE_NAME
   rbd snap purge POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap purge image1
   [root@rbd-client ~]# rbd snap purge pool1/image1
   ```

5.7. ブロックデバイススナップショットの名前変更

ブロックデバイスのスナップショットの名前を変更します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. スナップショットの名前を変更するには、以下のコマンドを実行します。

   構文

   ```
   rbd snap rename POOL_NAME/IMAGE_NAME@ORIGINAL_SNAPSHOT_NAME
   POOL_NAME/IMAGE_NAME@NEW_SNAPSHOT_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd snap rename data/dataset@snap1 data/dataset@snap2
   ```

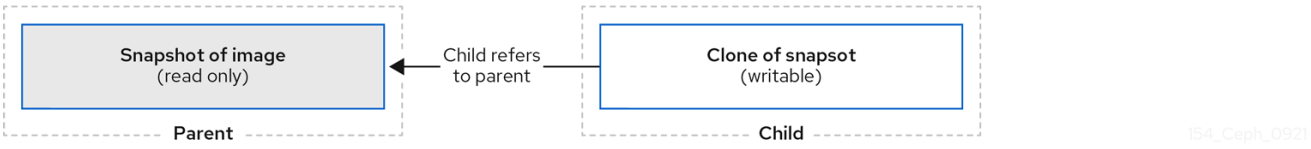
   これにより、**data** プールにある **データセット** イメージの **snap1** スナップショットの名前が **snap2** に変更されます。

2. **rbd help snap rename** コマンドを実行して、スナップショットの名前変更に関する追加情報を表示します。

5.8. CEPH ブロックデバイスの階層化

Ceph は、ブロックデバイススナップショットの多数のコピーオンライト (COW) またはコピーオンリード (COR) のクローンを作成する機能をサポートしています。スナップショットの階層化により、Ceph ブロックデバイスクライアントはイメージを非常に迅速に作成できます。たとえば、Linux 仮想マシンで書き込んだブロックデバイスイメージを作成できます。次に、イメージのスナップショットを作成して、スナップショットを保護し、必要な数のクローンを作成します。スナップショットは読み取り専用であるため、スナップショットのクローンを作成するとセマンティクスが簡素化され、クローンの作成時間を短縮できます。

図5.2 Ceph Block デバイスの階層化



注記

親 および **子** という用語は、Ceph ブロックデバイスのスナップショット (親)、およびスナップショットからクローン作成された対応のイメージ (子) を意味します。以下のコマンドラインを使用する場合に、これらの用語が重要です。

クローン作成された各イメージ (子) は、親イメージへの参照を保存し、クローン作成されたイメージで親スナップショットを開き、読み取ることができるようになります。この参照は、クローンが **フラット化** (スナップショットからの情報が完全にクローンにコピー) されると、削除されます。

スナップショットのクローン作成は、他の Ceph ブロックデバイスイメージのように動作します。クローン作成されたイメージを読み取り、書き込み、クローンし、サイズを変更できます。クローン作成されたイメージには、特別な制限はありません。ただし、スナップショットのクローンはスナップショットを参照するので、クローンを作成する前にスナップショットを保護する **必要があります**。

スナップショットのクローンは、コピーオンライト (COW) またはコピーオンリード (COR) のいずれかです。クローンではコピーオンライト (COW) は常に有効で、コピーオンリード (COR) は明示的に有効化する必要があります。コピーオンライト (COW) は、クローン内の未割り当てのオブジェクトへの書き込み時に、親からクローンにデータをコピーします。コピーオンリード (COR) は、クローン内の未

割り当てのオブジェクトから読み取る時に、親からクローンにデータをコピーします。クローンからデータの読み取りは、オブジェクトがクローンに存在しない場合、親からのデータのみを読み取ります。RADOS ブロックデバイスは、サイズの大きいイメージを複数のオブジェクトに分割します。デフォルトは 4 MB に設定され、すべてのコピーオンライト (COW) およびすべてのコピーオンリード (COR) 操作が完全なオブジェクトで行われます。つまり、クローンに 1 バイトが書き込まれると、4 MB オブジェクトが親から読み取られ、まだ以前の COW/COR 操作から宛先オブジェクトがクローンに存在しない場合には、クローンに書き込まれます。

コピーオンリード (COR) が有効になっているかどうか。クローンから下層にあるオブジェクトを読み取ることができない場合には、親に再ルーティングされます。実質的に親の数に制限が特にないため、クローンのクローンを作成できます。これは、オブジェクトが見つかるまで、またはベースの親イメージに到達するまで、この再ルーティングが続行されます。コピーオンリード (COR) が有効になっている場合には、クローンから直接読み取ることができない場合には、親からすべてのオブジェクトを読み取り、そのデータをクローンに書き込むことで、今後、親から読み取る必要なく、同じエクステントの読み取りがクローン自体で行われるようにします。

これは基本的に、オンデマンドのオブジェクトごとのフラット化操作です。これは、クローンが親から離れた高遅延接続の場所 (別の地理的場所の別のプールにある親など) にある場合に特に便利です。コピーオンリード (COR) では、読み取りのならし遅延が短縮されます。最初の数回読み取りは、親から追加のデータが読み取られるため、レイテンシーが高くなっています。たとえば、クローンから 1 バイトを読み取る場合に、4 MB を親から読み取り、クローンに書き込みする必要がありますが、それ以降はクローン自体からすべての読み取りが行われます。

スナップショットからコピーオンリード (COR) のクローンを作成するには、**ceph.conf** ファイルの **[global]** セクションまたは **[client]** セクションに **rbd_clone_copy_on_read = true** を追加してこの機能を明示的に有効にする必要があります。

関連情報

- **フラット化** の詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の クローンしたイメージのフラット化 セクションを参照してください。

5.9. ブロックデバイススナップショットの保護

親スナップショットのクローン作成は、親スナップショットにアクセスします。ユーザーが親のスナップショットを誤って削除した場合に、クローンはすべて破損します。

set-require-min-compat-client パラメーターは、Ceph の mimic バージョン以上に設定できます。

例

```
ceph osd set-require-min-compat-client mimic
```

これにより、デフォルトでクローン v2 が作成されます。ただし、mimic よりも古いクライアントは、これらのブロックデバイスイメージにアクセスできません。

注記

クローン v2 では、スナップショットの保護は必要ありません。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. 以下のコマンドで **POOL_NAME**、**IMAGE_NAME**、および **SNAP_SHOT_NAME** を指定します。

   構文

   ```
   rbd --pool POOL_NAME snap protect --image IMAGE_NAME --snap SNAPSHOT_NAME
   rbd snap protect POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap protect --image image1 --snap snap1
   [root@rbd-client ~]# rbd snap protect pool1/image1@snap1
   ```

   注記

   保護されたスナップショットは削除できません。

5.10. ブロックデバイススナップショットのクローン作成

ブロックデバイスのスナップショットのクローンを作成して、同じプール内または別のプール内に、スナップショットの子イメージ (読み取りまたは書き込みイメージ) を作成します。ユースケースの例として、読み取り専用のイメージおよびスナップショットをプールでテンプレートとして維持し、別のプールで書き込み可能なクローンとして維持します。

注記

クローン v2 では、スナップショットの保護は必要ありません。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. スナップショットのクローンを作成するには、親プール、スナップショット、子プール、およびイメージ名を指定する必要があります。

   構文

   ```
   rbd snap --pool POOL_NAME --image PARENT_IMAGE --snap SNAP_NAME --dest-pool
   POOL_NAME --dest CHILD_IMAGE_NAME
   rbd clone POOL_NAME/PARENT_IMAGE@SNAP_NAME
   POOL_NAME/CHILD_IMAGE_NAME
   ```

   例

```
[root@rbd-client ~]# rbd clone --pool pool1 --image image1 --snap snap2 --dest-pool pool2 --
dest childimage1
[root@rbd-client ~]# rbd clone pool1/image1@snap1 pool1/childimage1
```

5.11. ブロックデバイススナップショットの保護解除

スナップショットを削除する前に、そのスナップショットを保護解除する必要があります。さらに、クローンからの参照があるスナップショットは、**削除できません**。スナップショットを削除する前に、スナップショットの各クローンをフラット化する必要があります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. 以下のコマンドを実行します。

   構文

   ```
   rbd --pool POOL_NAME snap unprotect --image IMAGE_NAME --snap SNAPSHOT_NAME
   rbd snap unprotect POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap unprotect --image image1 --snap snap1

   [root@rbd-client ~]# rbd snap unprotect pool1/image1@snap1
   ```

5.12. スナップショットの子のリスト表示

スナップショットの子をリスト表示します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. スナップショットの子をリスト表示するには、以下のコマンドを実行します。

   構文

   ```
   rbd --pool POOL_NAME children --image IMAGE_NAME --snap SNAP_NAME
   rbd children POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

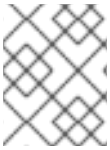
   例

   -

```
[root@rbd-client ~]# rbd --pool pool1 children --image image1 --snap snap1
[root@rbd-client ~]# rbd children pool1/image1@snap1
```

5.13. クローンしたイメージのフラット化

クローン作成されたイメージは、親スナップショットへの参照を保持します。親スナップショットへの参照を子クローンから削除すると、実質的に、その情報をスナップショットからクローンにコピーしてイメージを "フラット化" できます。クローンのフラット化にかかる時間は、スナップショットのサイズとともに増加します。フラット化イメージにはスナップショットからのすべての情報が含まれるため、フラット化されるイメージは階層化されたクローンよりも多くのストレージ領域を使用します。



注記

イメージで **ディープフラット** 機能が有効になっている場合には、イメージのクローンは、デフォルトで親から分離されます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. 子イメージに関連付けられた親イメージのスナップショットを削除するには、最初に子イメージをフラット化する必要があります。

   構文

   ```
   rbd --pool POOL_NAME flatten --image IMAGE_NAME
   rbd flatten POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd --pool pool1 flatten --image childimage1
   [root@rbd-client ~]# rbd flatten pool1/childimage1
   ```

第6章 CEPH ブロックデバイスのミラーリング

ストレージ管理者は、Red Hat Ceph Storage クラスター間でデータイメージをミラーリングして、冗長性向けに別の階層を Ceph ブロックデバイスに追加できます。Ceph ブロックデバイスのミラーリングについて理解して使用すると、サイト障害など、データ損失から守ることができます。Ceph ブロックデバイスのミラーリングには、一方向ミラーリングまたは双方向ミラーリングの 2 つの設定があり、プールと個別のイメージにミラーリングを設定できます。

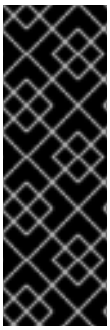
前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- 2 つのストレージクラスター間のネットワーク接続。
- 各 Red Hat Ceph Storage クラスターの Ceph クライアントノードへのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。

6.1. CEPH ブロックデバイスのミラーリング

RADOS Block Device (RBD) ミラーリングとは、2 つ以上の Ceph Storage クラスター間で Ceph ブロックデバイスイメージを非同期にレプリケーションするプロセスのことです。異なる地理的な場所にある Ceph Storage クラスターを配置することで、RBD ミラーリングはサイトの障害からの復旧に役立ちます。ジャーナルベースの Ceph ブロックデバイスのミラーリングにより、読み取りと書き込み、ブロックデバイスのサイズ調整、スナップショット、クローンおよびフラット化など、イメージに対する全変更を含む、ある時点の一貫したレプリカが作成されるようにします。

RBD ミラーリングは排他的ロックとジャーナリング機能を使用して、イメージに対するすべての変更を順番に記録します。これにより、イメージのクラッシュ整合性のあるミラーが利用できるようになりました。



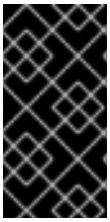
重要

ブロックデバイスイメージをミラーリングするプライマリーおよびセカンダリープールに対応する CRUSH 階層には、容量とパフォーマンスの特性が同じである必要があり、また、追加のレイテンシーなしにミラーリングを行うために十分な帯域幅が必要になります。たとえば、プライマリーストレージクラスター内のイメージへの平均書き込みスループットが X MB/s である場合に、ネットワークはセカンダリーサイトへのネットワーク接続で N * X スループットと、N イメージをミラーリングする安全係数 Y% に対応している必要があります。

rbd-mirror デーモンは、リモートプライマリーイメージから変更を取得し、プライマリーイメージ以外のローカルイメージにそれらの変更を書き込むことで、別の Ceph Storage クラスターにイメージを同期します。**rbd-mirror** デーモンは、Ceph Storage クラスター 1 台では一方向ミラーリング、Ceph Storage クラスター 2 台ではミラーリング関係に参加する双方向ミラーリングを実行します。

一方向または双方向レプリケーションのどちらかを使用して RBD ミラーリングを機能させる場合に、いくつかの前提条件があります。

- 同じ名前のプールが両方のストレージクラスターに存在する。
- プールには、ジャーナルが有効化された、ミラーリングするイメージが含まれている。



重要

一方向または双方向レプリケーションでは、**rbd-mirror** の各インスタンスは他の Ceph Storage クラスターを同時に接続できる必要があります。また、ミラーリングを処理するために、ネットワークには 2 つのデータセンターサイトの間で十分な帯域幅が必要です。

一方向レプリケーション

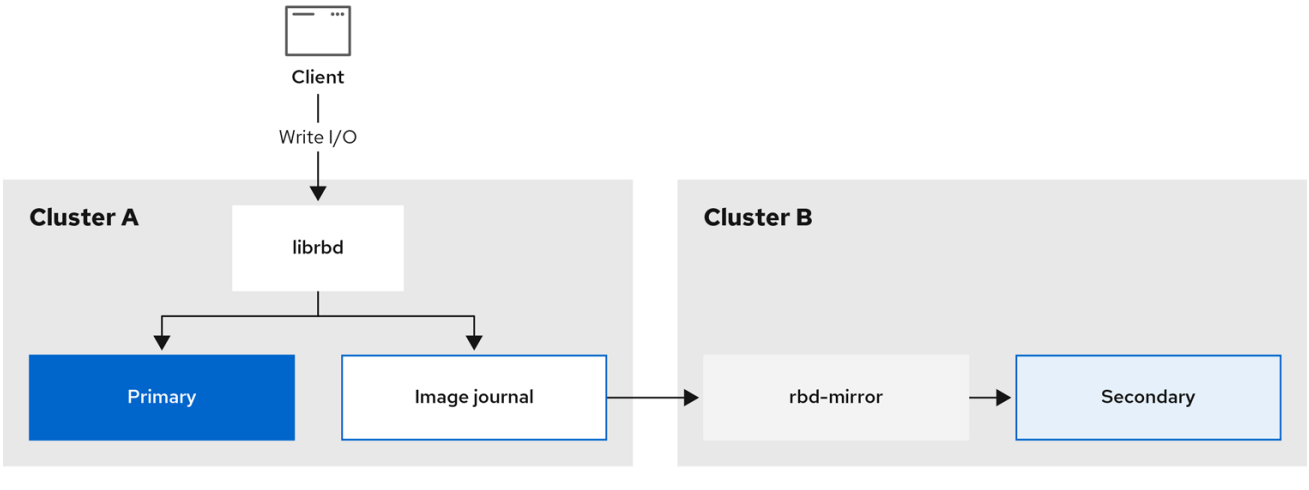
一方向ミラーリングは、ストレージクラスター内のプライマリーイメージまたはプールのイメージがセカンダリーのストレージクラスターにレプリケートされることを意味します。一方向ミラーリングは、複数のセカンダリーストレージクラスターへのレプリケーションにも対応します。

セカンダリーストレージクラスターでは、イメージはプライマリー以外のレプリケーションであるため、Ceph クライアントはイメージに書き込むことができません。データがプライマリーストレージクラスターからセカンダリーストレージクラスターにミラーリングされると、**rbd-mirror** はセカンダリーストレージクラスター上でのみを実行します。

一方向のミラーリングを機能させるには、いくつかの前提条件があります。

- 2 つの Ceph Storage クラスターがあり、プライマリーストレージクラスターからセカンダリーストレージクラスターにイメージをレプリケートする必要がある。
- セカンダリーストレージクラスターには、**rbd-mirror** デーモンを実行する Ceph クライアントノードがアタッチされている。**rbd-mirror** デーモンは、プライマリーストレージクラスターに接続して、イメージをセカンダリーストレージクラスターに同期します。

図6.1 一方向ミラーリング



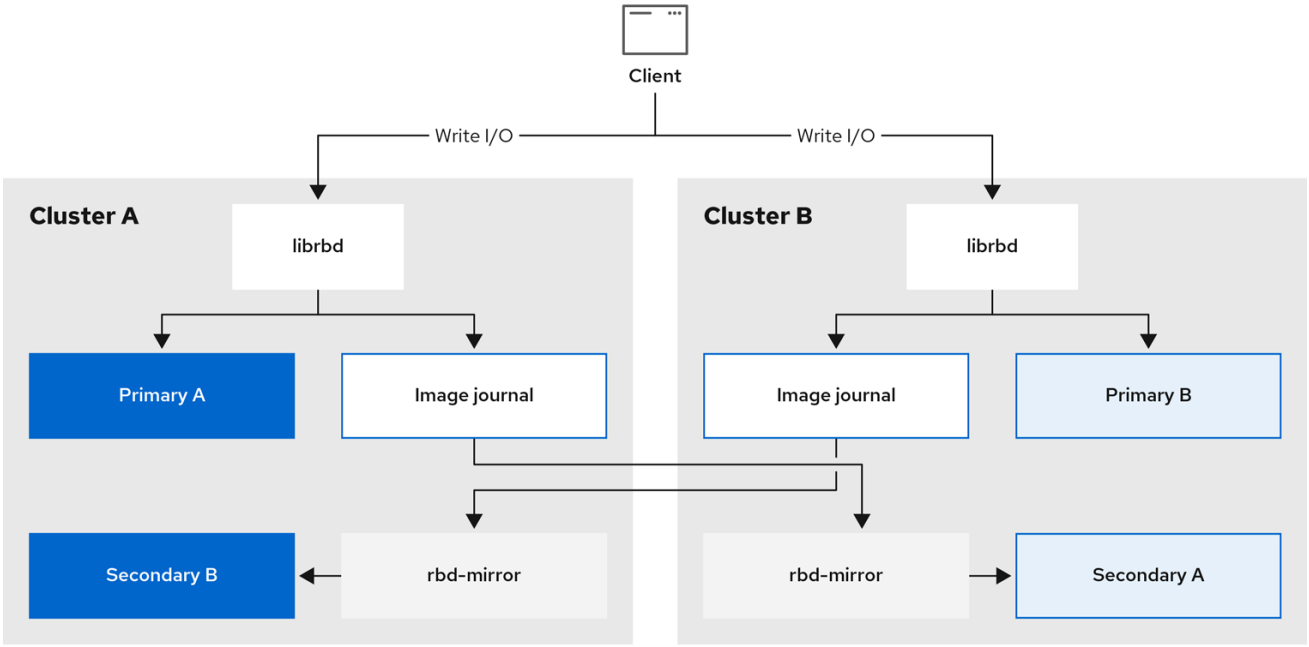
双方向レプリケーション

双方向レプリケーションは、プライマリークラスターに **rbd-mirror** デーモンを追加して、そのクラスターでイメージをデモートし、セカンダリークラスターでプロモートできるようにします。その後、セカンダリークラスターのイメージに対して変更が行われ、セカンダリーからプライマリーに逆方向にレプリケートされます。どちらかのクラスターでのイメージのプロモートとデモートを可能にするには、両方のクラスターで **rbd-mirror** が実行されている必要があります。現在、双方向レプリケーションは 2 つのサイトの間でのみサポートされています。

双方向のミラーリングを機能させるには、いくつかの前提条件があります。

- ストレージクラスターが 2 台あり、それらのクラスター間でイメージをどちらの方向にでも複製できる。
- 両方のストレージクラスターには、**rbd-mirror** デーモンを実行するクライアントノードが割り当てられている。セカンダリーストレージクラスターで実行される **rbd-mirror** デーモンは、プライマリーストレージクラスターに接続してイメージをセカンダリーに同期し、プライマリーストレージクラスターで実行されている **rbd-mirror** デーモンは、セカンダリーストレージクラスターに接続し、イメージをプライマリーに同期します。

図6.2 双方向ミラーリング



ミラーリングモード

ミラーリングは、ストレージクラスターのミラーリングを使用して、プールごとに設定されます。Ceph は、プールのイメージの種類に応じて、2 つのミラーリングモードをサポートします。

プールモード

ジャーナリング機能が有効になっているプール内のイメージはすべてミラーリングされます。

イメージモード

プール内の特定のイメージのサブセットのみがミラーリングされます。各イメージのミラーリングを別々に有効にする必要があります。

イメージの状態

イメージの変更が可能かどうかは、その状態により異なります。

- プライマリー状態のイメージを変更できます。
- プライマリー状態以外のイメージは変更できません。

イメージでミラーリングが最初に有効化された時点で、イメージはプライマリーに自動的にプロモートされます。以下でプロモートが可能です。

- プールモードでミラーリングを暗黙的に有効にする。

- 特定のイメージのミラーリングを明示的に有効にする。

プライマリーイメージをデモートし、プライマリー以外のイメージをプロモートすることができます。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **プールでのミラーリングの有効化** セクションを参照してください。
- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **イメージミラーリングの有効化** セクションを参照してください。
- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **イメージのプロモートおよびデモート** セクションを参照してください。

6.1.1. ジャーナルベースおよびスナップショットベースのミラーリングの概要

RADOS Block Device (RBD) イメージは、2 つのモードを使用して 2 つの Red Hat Ceph Storage クラスター間で非同期にミラーリングできます。

ジャーナルベースのミラーリング

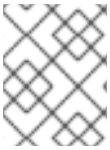
このモードでは、RBD ジャーナリングイメージ機能を使用して、2 つの Red Hat Ceph Storage クラスター間のある時点でのレプリケーションと、クラッシュ整合性のあるレプリケーションを行えるようにします。実際のイメージは、RBD イメージへのすべての書き込みが最初に関連付けられたジャーナルに記録されるまで変更されません。リモートクラスターはこのジャーナルから読み取り、イメージのローカルコピーへの更新をリプレイします。RBD イメージへの書き込みごとに Ceph クラスターへの書き込みが 2 回発生するため、RBD ジャーナリングイメージ機能を使用すると、書き込みレイテンシーがほぼ 2 倍になります。

スナップショットベースのミラーリング

このモードでは、定期的なスケジュール済みまたは手動で作成された RBD イメージミラースナップショットを使用して、2 つの Red Hat Ceph Storage クラスター間にクラッシュの整合性のある RBD イメージを複製します。リモートクラスターは、2 つのミラースナップショット間のデータまたはメタデータの更新を判断して、差異をイメージのローカルコピーにコピーします。RBD の **fast-diff** イメージ機能により、完全な RBD イメージをスキャンしなくても、更新されたデータブロックをすばやく判断できます。フェイルオーバーのシナリオで使用する前に、2 つのスナップショットの間にある差異をすべて同期する必要があります。部分的に適用されている差異については、フェイルオーバー時にロールバックされます。

6.2. コマンドラインインターフェイスを使用した一方向ミラーリングの設定

この手順では、プライマリーストレージクラスターからセカンダリーストレージクラスターへのプールの一方向レプリケーションを設定します。



注記

一方向レプリケーションを使用する場合は、複数のセカンダリーストレージクラスターにミラーリングできます。

注記

このセクションの例には、プライマリーイメージでプライマリーストレージクラスターを **site-a** として、そのイメージをレプリケートするセカンダリーストレージクラスターを **site-b** として参照し、2 つのストレージクラスターを区別します。これらの例で使用されるプール名は **data** と呼ばれます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- 各ストレージクラスターの Ceph クライアントノードへの Root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。

手順

1. 両方のサイトで **cephadm** シェルにログインします。

   例

   ```
   [root@site-a ~]# cephadm shell
   [root@site-b ~]# cephadm shell
   ```

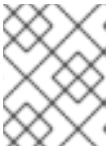
2. **site-b** で、セカンダリークラスターでミラーデーモンのデプロイメントをスケジュールします。

   構文

   ```
   ceph orch apply rbd-mirror --placement=NODENAME
   ```

   例

   ```
   [ceph: root@site-b /]# ceph orch apply rbd-mirror --placement=host04
   ```

   注記

   nodename は、セカンダリークラスターでミラーリングを設定するホストです。

3. **site-a** のイメージのジャーナリング機能を有効にします。

   a. **新規イメージ**の場合は、**--image-feature** オプションを使用します。

      構文

      ```
      rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME --image-feature
      FEATURE FEATURE
      ```

      例

      ```
      [ceph: root@site-a /]# rbd create image1 --size 1024 --pool data --image-feature
      exclusive-lock,journaling
      ```

-

注記

exclusive-lock がすでに有効にされている場合は、**ジャーナリング** のみを引数として使用します。それ以外の場合は、以下のエラーが返されます。

```
one or more requested features are already enabled
(22) Invalid argument
```

b. **既存のイメージ** の場合は、**rbd feature enable** コマンドを使用します。

構文

```
rbd feature enable POOL_NAME/IMAGE_NAME FEATURE, FEATURE
```

例

```
[ceph: root@site-a /]# rbd feature enable data/image1 exclusive-lock, journaling
```

c. デフォルトですべての新規イメージのジャーナリングを有効にするには、**ceph config set** コマンドを使用して設定パラメーターを設定します。

例

```
[ceph: root@site-a /]# ceph config set global rbd_default_features 125
[ceph: root@site-a /]# ceph config show mon.host01 rbd_default_features
```

4. 両方のストレージクラスターで、ミラーリングモード (pool または image モード) を選択します。

a. **プールモード** の有効化:

構文

```
rbd mirror pool enable POOL_NAME MODE
```

例

```
[ceph: root@site-a /]# rbd mirror pool enable data pool
[ceph: root@site-b /]# rbd mirror pool enable data pool
```

この例では、**data** という名前のプール全体のミラーリングを有効にします。

b. **イメージモード** の有効化:

構文

```
rbd mirror pool enable POOL_NAME MODE
```

例

```
[ceph: root@site-a /]# rbd mirror pool enable data image
[ceph: root@site-b /]# rbd mirror pool enable data image
```

この例では、**data** という名前のプールでイメージモードのミラーリングを有効にします。

注記

プールの特定イメージのミラーリングを有効にするには、**Red Hat Ceph Storage ブロックデバイスガイド** の イメージミラーリングの有効化 セクションを参照してください。

c. 両方のサイトでミラーリングが正常に有効になっていることを確認します。

構文

```
rbd mirror pool info POOL_NAME
```

例

```
[ceph: root@site-a /]# rbd mirror pool info data
Mode: pool
Site Name: c13d8065-b33d-4cb5-b35f-127a02768e7f

Peer Sites: none

[ceph: root@site-b /]# rbd mirror pool info data
Mode: pool
Site Name: a4c667e2-b635-47ad-b462-6faeeee78df7

Peer Sites: none
```

5. Ceph クライアントノードで、ストレージクラスターのピアをブートストラップします。

a. Ceph ユーザーアカウントを作成し、ストレージクラスターのピアをプールに登録します。

構文

```
rbd mirror pool peer bootstrap create --site-name PRIMARY_LOCAL_SITE_NAME POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
```

例

```
[ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a data > /root/bootstrap_token_site-a
```

注記

以下の bootstrap コマンド例では、**client.rbd-mirror.site-a** および **client.rbd-mirror-peer** Ceph ユーザーを作成します。

b. ブートストラップトークンファイルを **site-b** ストレージクラスターにコピーします。

c. **site-b** ストレージクラスターでブートストラップトークンをインポートします。

構文

```
rbd mirror pool peer bootstrap import --site-name SECONDARY_LOCAL_SITE_NAME --direction rx-only POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
```

例

```
[ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-only data /root/bootstrap_token_site-a
```

注記

一方向 RBD ミラーリングでは、ピアのブートストラップ時に双方向のミラーリングがデフォルトであるため **--direction rx-only** 引数を使用する必要があります。

6. ミラーリングのステータスを確認するには、プライマリーサイトおよびセカンダリーサイトの Ceph Monitor ノードから以下のコマンドを実行します。

構文

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

例

```
[ceph: root@mon-site-a /]# rbd mirror image status data/image1
image1:
  global_id:   c13d8065-b33d-4cb5-b35f-127a02768e7f
  state:       up+stopped
  description: remote image is non-primary
  service:     host03.yuoosv on host03
  last_update: 2021-10-06 09:13:58
```

ここでは **up** は **rbd-mirror** デーモンが実行中で、**stopped** は、このイメージが別のストレージクラスターからのレプリケーション先ではないことを意味します。これは、イメージがこのストレージクラスターのプライマリーであるためです。

例

```
[ceph: root@mon-site-b /]# rbd mirror image status data/image1
image1:
  global_id:   c13d8065-b33d-4cb5-b35f-127a02768e7f
```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。
- Ceph ユーザーの詳細は、**Red Hat Ceph Storage 管理ガイド**の ユーザー管理 セクションを参照してください。

6.3. コマンドラインインターフェイスを使用した双方向ミラーリングの設定

この手順では、プライマリーストレージクラスターとセカンダリーストレージクラスターとの間に、プールの双方向レプリケーションを設定します。

注記

双方向レプリケーションを使用する場合にミラーリングできるのは、2 つのストレージクラスター間だけです。

注記

このセクションの例には、プライマリーイメージでプライマリーストレージクラスターを **site-a** として、そのイメージをレプリケートするセカンダリーストレージクラスターを **site-b** として参照し、2 つのストレージクラスターを区別します。これらの例で使用されるプール名は **data** と呼ばれます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- 各ストレージクラスターの Ceph クライアントノードへの Root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。

手順

1. 両方のサイトで **cephadm** シェルにログインします。

   例

   ```
   [root@site-a ~]# cephadm shell
   [root@site-b ~]# cephadm shell
   ```

2. **site-a** プライマリークラスターで、以下のコマンドを実行します。

   例

   ```
   [ceph: root@site-a /]# ceph orch apply rbd-mirror --placement=host01
   ```

   注記

   nodename は、ミラーリングを設定するホストです。

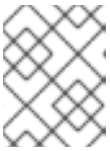
3. **site-b** で、セカンダリークラスターでミラーデーモンのデプロイメントをスケジュールします。

   構文

   ```
   ceph orch apply rbd-mirror --placement=NODENAME
   ```

   例

```
[ceph: root@site-b /]# ceph orch apply rbd-mirror --placement=host04
```



注記

nodename は、セカンダリークラスターでミラーリングを設定するホストです。

4. **site-a** のイメージのジャーナリング機能を有効にします。

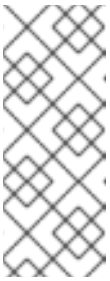
   a. **新規イメージ**の場合は、**--image-feature** オプションを使用します。

   構文

   ```
   rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME --image-feature
   FEATURE FEATURE
   ```

   例

   ```
   [ceph: root@site-a /]# rbd create image1 --size 1024 --pool data --image-feature
   exclusive-lock,journaling
   ```

   注記

   exclusive-lock がすでに有効にされている場合は、**ジャーナリング** のみを引数として使用します。それ以外の場合は、以下のエラーが返されます。

   ```
   one or more requested features are already enabled
   (22) Invalid argument
   ```

   b. **既存のイメージ** の場合は、**rbd feature enable** コマンドを使用します。

   構文

   ```
   rbd feature enable POOL_NAME/IMAGE_NAME FEATURE, FEATURE
   ```

   例

   ```
   [ceph: root@site-a /]# rbd feature enable data/image1 exclusive-lock, journaling
   ```

   c. デフォルトですべての新規イメージのジャーナリングを有効にするには、**ceph config set** コマンドを使用して設定パラメーターを設定します。

   例

   ```
   [ceph: root@site-a /]# ceph config set global rbd_default_features 125
   [ceph: root@site-a /]# ceph config show mon.host01 rbd_default_features
   ```

5. 両方のストレージクラスターで、ミラーリングモード (pool または image モード) を選択します。

   a. **プールモード** の有効化:

構文

```
rbd mirror pool enable POOL_NAME MODE
```

例

```
[ceph: root@site-a /]# rbd mirror pool enable data pool
[ceph: root@site-b /]# rbd mirror pool enable data pool
```

この例では、**data** という名前のプール全体のミラーリングを有効にします。

b. **イメージモード** の有効化:

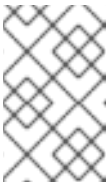
構文

```
rbd mirror pool enable POOL_NAME MODE
```

例

```
[ceph: root@site-a /]# rbd mirror pool enable data image
[ceph: root@site-b /]# rbd mirror pool enable data image
```

この例では、**data** という名前のプールでイメージモードのミラーリングを有効にします。



注記

プールの特定イメージのミラーリングを有効にするには、**Red Hat Ceph Storage ブロックデバイスガイド** の **イメージミラーリングの有効化** セクションを参照してください。

c. 両方のサイトでミラーリングが正常に有効になっていることを確認します。

構文

```
rbd mirror pool info POOL_NAME
```

例

```
[ceph: root@site-a /]# rbd mirror pool info data
Mode: pool
Site Name: c13d8065-b33d-4cb5-b35f-127a02768e7f

Peer Sites: none

[ceph: root@site-b /]# rbd mirror pool info data
Mode: pool
Site Name: a4c667e2-b635-47ad-b462-6faeeee78df7

Peer Sites: none
```

6. Ceph クライアントノードで、ストレージクラスターのピアをブートストラップします。

a. Ceph ユーザーアカウントを作成し、ストレージクラスターのピアをプールに登録します

a. Ceph ユーザーアカウントを作成し、ストレージクラスターのピアをプールに登録します。

構文

```
rbd mirror pool peer bootstrap create --site-name PRIMARY_LOCAL_SITE_NAME POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
```

例

```
[ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a data > /root/bootstrap_token_site-a
```

注記

以下の bootstrap コマンド例では、**client.rbd-mirror.site-a** および **client.rbd-mirror-peer** Ceph ユーザーを作成します。

b. ブートストラップトークンファイルを **site-b** ストレージクラスターにコピーします。

c. **site-b** ストレージクラスターでブートストラップトークンをインポートします。

構文

```
rbd mirror pool peer bootstrap import --site-name SECONDARY_LOCAL_SITE_NAME --direction rx-tx POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
```

例

```
[ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-tx data /root/bootstrap_token_site-a
```

注記

ピアのブートストラップ時には双方向ミラーリングがデフォルトであるため、**--direction** 引数はオプションです。

7. ミラーリングのステータスを確認するには、プライマリーサイトおよびセカンダリーサイトの Ceph Monitor ノードから以下のコマンドを実行します。

構文

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

例

```
[ceph: root@mon-site-a /]# rbd mirror image status data/image1
image1:
  global_id:   a4c667e2-b635-47ad-b462-6faeeee78df7
  state:       up+stopped
  description: local image is primary
  service:     host03.glsdbv on host03.ceph.redhat.com
  last_update: 2021-09-16 10:55:58
```

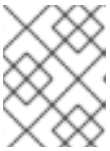
```
  peer_sites:
    name: a
    state: up+stopped
    description: replaying,
{"bytes_per_second":0.0,"entries_behind_primary":0,"entries_per_second":0.0,"non_primary_p
osition":{"entry_tid":3,"object_number":3,"tag_tid":1},"primary_position":
{"entry_tid":3,"object_number":3,"tag_tid":1}}
    last_update: 2021-09-16 10:55:50
```

ここでは **up** は **rbd-mirror** デーモンが実行中で、**stopped** は、このイメージが別のストレージクラスターからのレプリケーション先ではないことを意味します。これは、イメージがこのストレージクラスターのプライマリーであるためです。

例

```
[ceph: root@mon-site-b /]# rbd mirror image status data/image1
image1:
  global_id:   a4c667e2-b635-47ad-b462-6faeeee78df7
  state:       up+replaying
  description: replaying,
{"bytes_per_second":0.0,"entries_behind_primary":0,"entries_per_second":0.0,"non_primary_p
osition":{"entry_tid":3,"object_number":3,"tag_tid":1},"primary_position":
{"entry_tid":3,"object_number":3,"tag_tid":1}}
  service:     host05.dtisty on host05
  last_update: 2021-09-16 10:57:20
  peer_sites:
    name: b
    state: up+stopped
    description: local image is primary
    last_update: 2021-09-16 10:57:28
```

イメージが **up+replaying** の場合には、ミラーリングが正常に機能します。ここでは **up** は **rbd-mirror** デーモンが実行中で、**replaying** は、このイメージが別のストレージクラスターからのレプリケーション先であることを意味します。



注記

サイト間の接続によって、ミラーリングでイメージの同期に時間がかかる場合があります。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **Ceph ブロックデバイスのミラーリング** セクションを参照してください。
- Ceph ユーザーの詳細は、**Red Hat Ceph Storage 管理ガイド**の **ユーザー管理** セクションを参照してください。

6.4. CEPH ブロックデバイスのミラーリングの管理

ストレージ管理者は、Ceph ブロックデバイスのミラーリング環境の管理に役立つさまざまなタスクを実行できます。次のタスクを実行できます。

- ストレージクラスターピアの情報を表示する。

- ストレージクラスターピアを追加または削除する。
- プールまたはイメージのミラーリングステータスを取得する。
- プールまたはイメージでのミラーリングを有効化する。
- プールまたはイメージでのミラーリングを無効化する。
- ブロックデバイスのレプリケーションを遅延する。
- イメージをプロモートおよびデモートする。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- Ceph クライアントノードへの root レベルのアクセス。
- 一方向または双方向 Ceph ブロックデバイスのミラーリング関係。
- 管理者レベル権限が割り当てられた CephX ユーザー。

6.4.1. ピアに関する情報の表示

ストレージクラスターピアの情報を表示します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. ピアの情報を表示するには、以下を実行します。

   構文

   ```
   rbd mirror pool info POOL_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror pool info data
   Mode: pool
   Site Name: a

   Peer Sites:

   UUID: 950ddadf-f995-47b7-9416-b9bb233f66e3
   Name: b
   Mirror UUID: 4696cd9d-1466-4f98-a97a-3748b6b722b3
   Direction: rx-tx
   Client: client.rbd-mirror-peer
   ```

6.4.2. プールでのミラーリングの有効化

両方のピアクラスターで以下のコマンドを実行して、プールのミラーリングを有効にします。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. プールのミラーリングを有効にするには、以下を実行します。

   構文

   ```
   rbd mirror pool enable POOL_NAME MODE
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror pool enable data pool
   ```

   この例では、**data** という名前のプール全体のミラーリングを有効にします。

   例

   ```
   [root@rbd-client ~]# rbd mirror pool enable data image
   ```

   この例では、**data** という名前のプールでイメージモードのミラーリングを有効にします。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.4.3. プールでのミラーリングの無効化

ミラーリングを無効にする前に、ピアクラスターを削除します。

注記

プールのミラーリングを無効にすると、ミラーリングを別に有効化していたプール内にあるイメージに対するミラーリングも無効化されます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. プールのミラーリングを無効にするには、以下を実行します。

構文

```
rbd mirror pool disable POOL_NAME
```

例

```
[root@rbd-client ~]# rbd mirror pool disable data
```

この例では、**data** という名前のプールのミラーリングを無効にします。

6.4.4. イメージミラーリングの有効化

両方のピアストレージクラスターで、イメージモードのプール全体のミラーリングを有効にします。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. プール内の特定イメージのミラーリングを有効にします。

   構文

   ```
   rbd mirror image enable POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image enable data/image2
   ```

   この例では、**data** プールの **image2** イメージのミラーリングを有効にします。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **プールでのミラーリングの有効化** セクションを参照してください。

6.4.5. イメージミラーリングの無効化

イメージの Ceph Block Device ミラーリングを無効にすることができます。

前提条件

- スナップショットベースのミラーリングが設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

1. 特定のイメージのミラーリングを無効にするには、以下を実行します。

   構文

   ```
   rbd mirror image disable POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image disable data/image2
   ```

   この例では、**data** プールの **image2** イメージのミラーリングを無効にします。

関連情報

- クライアントを **cephadm-ansible** インベントリーに追加する方法は、**Red Hat Ceph Storage インストールガイド** の Ansible インベントリー場所の設定 セクションを参照してください。

6.4.6. イメージのプロモートおよびデモート

プールのイメージをプロモートまたはデモートできます。

注記

プロモート後にイメージは有効にならないので、プライマリー以外の同期中のイメージを強制的にプロモートしないでください。

前提条件

- スナップショットベースのミラーリングが設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

1. プライマリー以外にイメージをデモートするには、以下のコマンドを実行します。

   構文

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image2
   ```

   この例では、**data** プールの **image2** イメージをデモートします。

2. イメージをプライマリーにプロモートするには、以下のコマンドを実行します。

   構文

   ```
   rbd mirror image promote POOL_NAME/IMAGE_NAME
   ```

例

```
[root@rbd-client ~]# rbd mirror image promote data/image2
```

この例では、**data** プールの **image2** をプロモートします。

使用しているミラーリングのタイプに応じて、**一方向ミラーリングを使用した障害からの復旧** または **双方向ミラーリングを使用した障害からの復旧** を参照してください。

構文

```
rbd mirror image promote --force POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image promote --force data/image2
```

ピア Ceph Storage クラスターに伝播できない場合には、強制プロモートを使用します。伝播できない理由として、クラスターの障害や通信の停止などが挙げられます。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **正常でないシャットダウン後のフェイルオーバー** セクションを参照してください。

6.4.7. イメージの再同期

イメージを再同期して、一貫した状態を復元できます。ピアクラスター間で不整合な状態がある場合、**rbd-mirror** デーモンはイメージのミラーリングを試行しません。

再同期では、クラスターのプライマリーイメージの完全なコピーを使用してイメージが再作成されます。

警告

再同期により、イメージの既存のミラースナップショットスケジュールが削除されます。

前提条件

- スナップショットベースのミラーリングが設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

- プライマリーイメージに再同期を要求するには、以下を実行します。

構文

```
rbd mirror image resync POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image resync data/image2
```

この例では、**data** プールの **image2** の再同期を要求します。

関連情報

- 障害による不整合状態から回復するには、**一方向ミラーリングを使用した障害からの復旧** または **双方向ミラーリングを使用した障害からの復旧** を参照してください。

6.4.8. ストレージクラスターピアの追加

rbd-mirror デーモンのストレージクラスターピアを追加して、ピアストレージクラスターを検出します。たとえば、**site-a** ストレージクラスターをピアとして **site-b** ストレージクラスターに追加するには、**site-b** ストレージクラスターのクライアントノードから以下の手順を実行します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. ピアをプールに登録します。

構文

```
rbd --cluster CLUSTER_NAME mirror pool peer add POOL_NAME
PEER_CLIENT_NAME@PEER_CLUSTER_NAME -n CLIENT_NAME
```

例

```
[root@rbd-client ~]# rbd --cluster site-b mirror pool peer add data client.site-a@site-a -n
client.site-b
```

6.4.9. ストレージクラスターピアの削除

ピア UUID を指定してストレージクラスターピアを削除します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. プール名とピアー意識別子 (UUID) を指定します。

構文

```
rbd mirror pool peer remove POOL_NAME PEER_UUID
```

例

```
[root@rbd-client ~]# rbd mirror pool peer remove data 7e90b4ce-e36d-4f07-8cbc-42050896825d
```

ヒント

ピア UUID を表示するには、**rbd mirror pool info** コマンドを使用します。

6.4.10. プールのミラーリングステータスの取得

ストレージクラスターのプールのミラーステータスを取得できます。

前提条件

- スナップショットベースのミラーリングが設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

1. ミラーリングプールの概要を取得するには、以下を実行します。

構文

```
rbd mirror pool status POOL_NAME
```

例

```
[root@site-a ~]# rbd mirror pool status data
health: OK
daemon health: OK
image health: OK
images: 1 total
    1 replaying
```

ヒント

プールのすべてのミラーリングイメージのステータス詳細を出力するには、**--verbose** オプションを使用します。

6.4.11. 単一イメージのミラーリングステータスの取得

mirror image status コマンドを実行して、イメージのミラーステータスを取得できます。

前提条件

- スナップショットベースのミラーリングが設定された実行中の Red Hat Ceph Storage クラスター。
- ノードへのルートレベルのアクセス。

手順

1. ミラーリングされたイメージのステータスを取得するには、以下を実行します。

   構文

   ```
   rbd mirror image status POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@site-a ~]# rbd mirror image status data/image2
   image2:
     global_id:   1e3422a2-433e-4316-9e43-1827f8dbe0ef
     state:       up+unknown
     description: remote image is non-primary
     service:     pluto008.yuoosv on pluto008
     last_update: 2021-10-06 09:37:58
   ```

   この例では、**data** プールの **image2** イメージのステータスを取得します。

6.4.12. ブロックデバイスレプリケーションの遅延

一方向レプリケーションを使用する場合でも、RADOS Block Device (RBD) ミラーリングイメージ間でレプリケーションを遅延させることができます。セカンダリーイメージにレプリケーションされる前に、プライマリーイメージへの不要な変更を元に戻せるように、猶予の期間が必要な場合には、遅延レプリケーションを実装することができます。

注記

ブロックデバイスレプリケーションの遅延は、ジャーナルベースのミラーリングでのみ適用されます。

遅延レプリケーションを実装するには、宛先ストレージクラスター内の **rbd-mirror** デーモンで **rbd_mirroring_replay_delay = MINIMUM_DELAY_IN_SECONDS** 設定オプションを指定する必要があります。この設定は、**rbd-mirror** デーモンが使用する **ceph.conf** ファイル内でグローバルに適用することも、個別のイメージベースで適用することも可能です。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. 特定のイメージで遅延レプリケーションを使用するには、プライマリーイメージで以下の **rbd** CLI コマンドを実行します。

   構文

   ```
   rbd image-meta set POOL_NAME/IMAGE_NAME conf_rbd_mirroring_replay_delay MINIMUM_DELAY_IN_SECONDS
   ```

   例

   ```
   [root@rbd-client ~]# rbd image-meta set vms/vm-1 conf_rbd_mirroring_replay_delay 600
   ```

   この例では、**vms** プールのイメージ **vm-1** に、最小レプリケーション遅延を 10 分に設定します。

6.4.13. ジャーナルベースのミラーリングからスナップショットベースのミラーリングへの変換

ミラーリングを無効にして、スナップショットを有効にすることで、ジャーナルベースのミラーリングから、スナップショットベースのミラーリングに変換できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. Cephadm シェルにログインします。

   例

   ```
   [root@rbd-client ~]# cephadm shell
   ```

2. プール内の特定イメージのミラーリングを無効にします。

   構文

   ```
   rbd mirror image disable POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [ceph: root@rbd-client /]# rbd mirror image disable mirror_pool/mirror_image
   Mirroring disabled
   ```

3. イメージのスナップショットベースのミラーリングを有効にします。

   構文

   ```
   rbd mirror image enable POOL_NAME/IMAGE_NAME snapshot
   ```

例

```
[ceph: root@rbd-client /]# rbd mirror image enable mirror_pool/mirror_image snapshot
Mirroring enabled
```

この例では、**mirror_pool** プールの **mirror_image** イメージのスナップショットベースのミラーリングを有効にします。

6.4.14. イメージのミラーリングスナップショットの作成

スナップショットベースのミラーリングの使用時に RBD イメージの変更をミラーリングする必要がある場合には、イメージのミラーリングスナップショットを作成します。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- Red Hat Ceph Storage クラスターの Ceph クライアントノードへの root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。
- スナップショットのミラーリングの作成先の Red Hat Ceph Storage クラスターへのアクセス。

重要

デフォルトでは、最大 5 つのイメージのミラーリングスナップショットが保持されます。上限に達すると、最新のイメージのミラーリングスナップショットが自動的に削除されます。必要な場合は、**rbd_mirroring_max_mirroring_snapshots** 設定で制限を上書きできます。イメージのミラーリングスナップショットは、イメージが削除された場合、ミラーリングが無効になっている場合に自動的に削除されます。

手順

- イメージのミラーリングスナップショットを作成するには、以下を実行します。

  構文

  ```
  rbd --cluster CLUSTER_NAME mirror image snapshot POOL_NAME/IMAGE_NAME
  ```

  例

  ```
  [root@site-a ~]# rbd mirror image snapshot data/image1
  ```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.4.15. ミラーリングスナップショットのスケジューリング

ミラーリングスナップショットは、ミラーリングスナップショットのスケジュールが定義されると自動的に作成されます。ミラーリングスナップショットは、グローバルに、プールごとに、またはイメージレベルで、スケジュールできます。複数のミラーリングスナップショットのスケジュールはどのレベル

でも定義できますが、個別のミラーリングイメージに一致する最も具体的なスナップショットスケジュールのみが実行されます。

6.4.15.1. ミラーリングスナップショットのスケジュールの作成

snapshot schedule コマンドを使用して、ミラースナップショットのスケジュールを作成できます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- Red Hat Ceph Storage クラスターの Ceph クライアントノードへの root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。
- ミラーイメージをスケジュールする必要がある Red Hat Ceph Storage クラスターへのアクセス。

手順

1. ミラーリングスナップショットのスケジュールを作成するには、以下を実行します。

   構文

   ```
   rbd --cluster CLUSTER_NAME mirror snapshot schedule add --pool POOL_NAME --image IMAGE_NAME INTERVAL [START_TIME]
   ```

   CLUSTER_NAME は、クラスター名がデフォルト名 **ceph** とは異なる場合にのみ使用してください。間隔は、d、h、または m の接尾辞を使用して、日、時間、または分単位で指定できます。オプションで ISO 8601 の時間形式を使用する START_TIME を指定できます。

   例

   ```
   [root@site-a ~]# rbd mirror snapshot schedule add --pool data --image image1 6h
   ```

   例

   ```
   [root@site-a ~]# rbd mirror snapshot schedule add --pool data --image image1 24h 14:00:00-05:00
   ```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.4.15.2. 特定のレベルでの全スナップショットスケジュールのリスト表示

特定のレベルで、すべてのスナップショットスケジュールをリスト表示できます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。

- Red Hat Ceph Storage クラスターの Ceph クライアントノードへの root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。
- ミラーイメージをスケジュールする必要がある Red Hat Ceph Storage クラスターへのアクセス。

手順

1. プールまたはイメージ名を任意で指定して、グローバル、プール、またはイメージレベルごとにすべてのスナップショットスケジュールをリスト表示するには、以下を実行します。

   構文

   ```
   rbd --cluster site-a mirror snapshot schedule ls --pool POOL_NAME --recursive
   ```

   また、以下のように **--recursive** オプションを指定して、指定したレベルですべてのスケジュールをリスト表示することもできます。

   例

   ```
   [root@rbd-client ~]# rbd mirror snapshot schedule ls --pool data --recursive
   POOL        NAMESPACE IMAGE  SCHEDULE
   data        -         -      every 1d starting at 14:00:00-05:00
   data        -         image1   every 6h
   ```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.4.15.3. ミラーリングスナップショットのスケジュールの削除

snapshot schedule remove コマンドを使用して、ミラースナップショットのスケジュールを削除できます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- Red Hat Ceph Storage クラスターの Ceph クライアントノードへの root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。
- ミラーイメージをスケジュールする必要がある Red Hat Ceph Storage クラスターへのアクセス。

手順

1. ミラーリングスナップショットのスケジュールを削除するには、以下を実行します。

   構文

```
rbd --cluster CLUSTER_NAME mirror snapshot schedule remove --pool POOL_NAME --
image IMAGE_NAME INTERVAL START_TIME
```

間隔は、d、h、m の接尾辞を使用して、日、時間、または分単位で指定できます。オプションで ISO 8601 の時間形式を使用する START_TIME を指定できます。

例

```
[root@site-a ~]# rbd mirror snapshot schedule remove --pool data --image image1 6h
```

例

```
[root@site-a ~]# rbd mirror snapshot schedule remove --pool data --image image1 24h
14:00:00-05:00
```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.4.15.4. 次に作成するスナップショットのステータスの表示

スナップショットベースのミラーリング RBD イメージとして次に作成されるスナップショットのステータスを表示できます。

前提条件

- 少なくとも、正常に実行されている Red Hat Ceph Storage クラスター 2 台。
- Red Hat Ceph Storage クラスターの Ceph クライアントノードへの root レベルのアクセス。
- 管理者レベル権限が割り当てられた CephX ユーザー。
- ミラーイメージをスケジュールする必要がある Red Hat Ceph Storage クラスターへのアクセス。

手順

1. 次の作成されるスナップショットの状態を表示するには、以下を実行します。

   構文

   ```
   rbd --cluster site-a mirror snapshot schedule status [--pool POOL_NAME] [--image
   IMAGE_NAME]
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror snapshot schedule status
   SCHEDULE    TIME       IMAGE
   2021-09-21 18:00:00 data/image1
   ```

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング セクションを参照してください。

6.5. 障害からの復旧

ストレージ管理者は、ミラーリングが設定された別のストレージクラスターからデータを回復する方法を理解することで、致命的なハードウェアの障害に備えることができます。

この例では、プライマリーストレージクラスターは **site-a** と呼ばれ、セカンダリーストレージクラスターは **site-b** と呼ばれます。また、ストレージクラスターにはどちらも **image1** と **image2** の 2 つのイメージが含まれる **data** プールがあります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- 一方向または双方向ミラーリングが設定されている。

6.5.1. 障害回復

2 つ以上の Red Hat Ceph Storage クラスターの間でブロックデータを非同期にレプリケーションすることで、データセンターで大規模な障害が発生した場合にデータの損失を防ぎ、ダウンタイムが削減されます。このような障害の影響は **大規模な爆発半径** とも呼ばれ、広範囲にわたります。また、送配電網への影響や、天然災害などが原因となります。

顧客データはこれらのシナリオに備え、保護する必要があります。ボリュームは、Micre Recovery Pointive (RPO) および Recovery Time Objective (RTO) ターゲット内で、一貫性と効率性を使用してレプリケーションする必要があります。このソリューションは、Dnaly Area Network- Disaster Recovery (WAN-DR) と呼ばれます。

このようなシナリオでは、プライマリーシステムとデータセンターを復元することが困難です。最も簡単に復元する方法として、別の Red Hat Ceph Storage クラスター (障害回復サイト) にアプリケーションをフェイルオーバーして、利用可能な最新のデータのコピーでクラスターを稼働させることなどが挙げられます。このような障害シナリオから回復するのに使用されるソリューションは、アプリケーションによりガイドされます。

- **Recovery Point Objective (RPO)**: 最悪の場合にアプリケーションが許容するデータ損失量。
- **Recovery Time Objective (RTO)**: 利用可能なデータの最新コピーで、アプリケーションをオンラインに戻すのにかかる時間。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の Ceph ブロックデバイスのミラーリング の章を参照してください。
- 暗号化された状態のデータ転送の詳細は、**Red Hat Ceph Storage データのセキュリティーおよび強化ガイド** の **転送中での暗号化** セクションを参照してください。

6.5.2. 一方向ミラーリングを使用した障害からの復旧

一方向のミラーリングで障害から回復するには、以下の手順を使用します。以下で、プライマリークラスターを終了してからセカンダリークラスターにフェイルオーバーする方法、およびフェイルバックする方法が紹介します。シャットダウンは規定の順序で行うことも、順序関係なく行うこともできます。

重要

一方向ミラーリングは、複数のセカンダリーサイトをサポートします。追加のセカンダリークラスターを使用している場合は、セカンダリークラスターの中から1つ選択してフェイルオーバーします。フェイルバック中に同じクラスターから同期します。

6.5.3. 双方向ミラーリングを使用した障害からの復旧

双方向ミラーリングで障害から回復するには、以下の手順を使用します。以下で、プライマリークラスターを終了してからセカンダリークラスターのミラーリングデータにフェイルオーバーする方法、およびフェイルバックする方法が紹介します。シャットダウンは規定の順序で行うことも、順序関係なく行うこともできます。

6.5.4. 正常なシャットダウン後のフェイルオーバー

正常にシャットダウンした後にセカンダリーストレージクラスターにフェイルオーバーします。

前提条件

- 少なくとも実行中の Red Hat Ceph Storage クラスターが 2 台ある。
- ノードへのルートレベルのアクセス。
- 一方向ミラーリングを使用して設定されるプールのミラーリングまたはイメージミラーリング。

手順

1. プライマリーイメージを使用するクライアントをすべて停止します。この手順は、どのクライアントがイメージを使用するかにより異なります。たとえば、イメージを使用する OpenStack インスタンスからボリュームの割り当てを解除します。
2. **site-a** クラスターのモニターノードで以下のコマンドを実行して、**site-a** クラスターにあるプライマリーイメージをデモートします。

   構文

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image1
   [root@rbd-client ~]# rbd mirror image demote data/image2
   ```

3. **site-b** クラスターにあるプライマリー以外のイメージをプロモートするには、**site-b** クラスターのモニターノードで以下のコマンドを実行します。

   構文

   ```
   rbd mirror image promote POOL_NAME/IMAGE_NAME
   ```

   例

```
[root@rbd-client ~]# rbd mirror image promote data/image1
[root@rbd-client ~]# rbd mirror image promote data/image2
```

4. しばらくすると、**site-b** クラスターのモニターノードからイメージのステータスを確認します。イメージのステータスは、**up+stopped** の状態を表示し、プライマリーとしてリストされているはずです。

```
[root@rbd-client ~]# rbd mirror image status data/image1
image1:
  global_id:   08027096-d267-47f8-b52e-59de1353a034
  state:       up+stopped
  description: local image is primary
  last_update: 2019-04-17 16:04:37
[root@rbd-client ~]# rbd mirror image status data/image2
image2:
  global_id:   596f41bc-874b-4cd4-aefe-4929578cc834
  state:       up+stopped
  description: local image is primary
  last_update: 2019-04-17 16:04:37
```

5. イメージへのアクセスを再開します。この手順は、どのクライアントがイメージを使用するかにより異なります。

関連情報

- **Red Hat OpenStack Platform ストレージガイド**の **ブロックストレージおよびボリューム** の章を参照してください。

6.5.5. 正常にシャットダウンされなかった場合のフェイルオーバー

正常でないシャットダウン後にセカンダリーストレージクラスターにフェイルオーバーします。

前提条件

- 少なくとも実行中の Red Hat Ceph Storage クラスターが 2 台ある。
- ノードへのルートレベルのアクセス。
- 一方向ミラーリングを使用して設定されるプールのミラーリングまたはイメージミラーリング。

手順

1. プライマリーストレージクラスターが停止していることを確認します。
2. プライマリーイメージを使用するクライアントをすべて停止します。この手順は、どのクライアントがイメージを使用するかにより異なります。たとえば、イメージを使用する OpenStack インスタンスからボリュームの割り当てを解除します。
3. **site-b** ストレージクラスターの Ceph Monitor ノードからプライマリー以外のイメージをプロモートします。**site-a** ストレージクラスターにデモートが伝播されないので、**--force** オプションを使用します。

   構文

```
rbd mirror image promote --force POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image promote --force data/image1
[root@rbd-client ~]# rbd mirror image promote --force data/image2
```

4. **site-b** ストレージクラスターの Ceph Monitor ノードからイメージのステータスを確認します。**up+stopping_replay** の状態が表示されるはずです。説明には、**force promoted** と書かれている必要があります。これは、断続的な状態にあることを意味します。状態が **up+stopped** になるまで待って、サイトが正常に昇格されたことを確認します。

   例

```
[root@rbd-client ~]# rbd mirror image status data/image1
image1:
  global_id:   08027096-d267-47f8-b52e-59de1353a034
  state:       up+stopping_replay
  description: force promoted
  last_update: 2023-04-17 13:25:06

[root@rbd-client ~]# rbd mirror image status data/image1
image1:
  global_id:   08027096-d267-47f8-b52e-59de1353a034
  state:       up+stopped
  description: force promoted
  last_update: 2023-04-17 13:25:06
```

関連情報

- Red Hat OpenStack Platform ストレージガイドの ブロックストレージおよびボリューム の章を参照してください。

6.5.6. フェイルバックの準備

2 つのストレージクラスターが元々、一方向ミラーリングだけ設定されていた場合に、フェイルバックするには、プライマリーストレージクラスターのミラーリングを設定して、反対方向にイメージをレプリケートできるようにします。

フェイルバックシナリオ中は、既存のクラスターに新しいピアを追加する前に、アクセスできない既存のピアを削除する必要があります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- クライアントノードへの root レベルのアクセス。

手順

1. Cephadm シェルにログインします。

   例

```
[root@rbd-client ~]# cephadm shell
```

2. **site-a** ストレージクラスターで、以下のコマンドを実行します。

   例

   ```
   [ceph: root@rbd-client /]# ceph orch apply rbd-mirror --placement=host01
   ```

3. アクセスできないピアをすべて削除します。

   重要

   この手順は、稼働中のピアサイトで実行する必要があります。

   注記

   複数のピアは一方向のミラーリングでのみサポートされます。

   a. プールの UUID を取得します。

      構文

      ```
      rbd mirror pool info POOL_NAME
      ```

      例

      ```
      [ceph: root@host01 /]# rbd mirror pool info pool_failback
      ```

   b. アクセスできないピアを削除します。

      構文

      ```
      rbd mirror pool peer remove POOL_NAME PEER_UUID
      ```

      例

      ```
      [ceph: root@host01 /]# rbd mirror pool peer remove pool_failback f055bb88-6253-4041-923d-08c4ecbe799a
      ```

4. ピアミラープールと同じ名前でブロックデバイスプールを作成します。

   a. rbd プールを作成するには、以下を実行します。

      構文

      ```
      ceph osd pool create POOL_NAME PG_NUM
      ceph osd pool application enable POOL_NAME rbd
      rbd pool init -p POOL_NAME
      ```

      例

```
[root@rbd-client ~]# ceph osd pool create pool1
[root@rbd-client ~]# ceph osd pool application enable pool1 rbd
[root@rbd-client ~]# rbd pool init -p pool1
```

5. Ceph クライアントノードで、ストレージクラスターのピアをブートストラップします。

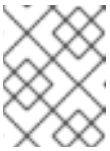
   a. Ceph ユーザーアカウントを作成し、ストレージクラスターのピアをプールに登録します。

      構文

      ```
      rbd mirror pool peer bootstrap create --site-name LOCAL_SITE_NAME POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
      ```

      例

      ```
      [ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a data > /root/bootstrap_token_site-a
      ```

      注記

      以下の bootstrap コマンド例では、**client.rbd-mirror.site-a** および **client.rbd-mirror-peer** Ceph ユーザーを作成します。

   b. ブートストラップトークンファイルを **site-b** ストレージクラスターにコピーします。

   c. **site-b** ストレージクラスターでブートストラップトークンをインポートします。

      構文

      ```
      rbd mirror pool peer bootstrap import --site-name LOCAL_SITE_NAME --direction rx-only POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
      ```

      例

      ```
      [ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-only data /root/bootstrap_token_site-a
      ```

      注記

      一方向 RBD ミラーリングでは、ピアのブートストラップ時に双方向のミラーリングがデフォルトであるため **--direction rx-only** 引数を使用する必要があります。

6. **site-a** ストレージクラスターのモニターノードから、**site-b** ストレージクラスターがピアとして正常に追加されたことを確認します。

   例

   ```
   [ceph: root@rbd-client /]# rbd mirror pool info -p data
   Mode: image
   Peers:
   ```

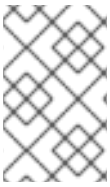
```
UUID                                 NAME   CLIENT
d2ae0594-a43b-4c67-a167-a36c646e8643 site-b client.site-b
```

関連情報

- 詳細は、**Red Hat Ceph Storage 管理ガイド**の **ユーザー管理** の章を参照してください。

6.5.6.1. プライマリーストレージクラスターへのフェイルバック

以前のプライマリーストレージクラスターが復元されたら、そのクラスターがプライマリーストレージクラスターにフェイルバックされます。



注記

イメージレベルでスナップショットをスケジュールしている場合は、イメージの再同期操作によって RBD イメージ ID が変更され、以前のスケジュールが廃止されるため、スケジュールを再追加する必要があります。

前提条件

- 少なくとも実行中の Red Hat Ceph Storage クラスターが 2 台ある。
- ノードへのルートレベルのアクセス。
- 一方向ミラーリングを使用して設定されるプールのミラーリングまたはイメージミラーリング。

手順

1. もう一度、**site-b** クラスターのモニターノードからイメージのステータスを確認します。状態として **up-stopped**、説明として **local image is primary** と表示されるはずです。

   例

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 17:37:48
   [root@rbd-client ~]# rbd mirror image status data/image2
   image2:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 17:38:18
   ```

2. **site-a** ストレージクラスターの Ceph Monitor ノードから、イメージがプライマリーかどうかを確認します。

   構文

   ```
   rbd mirror pool info POOL_NAME/IMAGE_NAME
   ```

例

```
[root@rbd-client ~]# rbd info data/image1
[root@rbd-client ~]# rbd info data/image2
```

コマンドの出力で、**mirroring primary: true** または **mirroring primary: false** を検索し、状態を判断します。

3. **site-a** ストレージクラスターの Ceph Monitor ノードから以下のようなコマンドを実行して、プライマリーとして表示されているイメージをデモートします。

   構文

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image1
   ```

4. 異常なシャットダウンがあった場合は、イメージを再同期します。
   クラスター内の他のノードと一致しないイメージは、クラスターのプライマリーイメージの完全なコピーを使用して再作成されます。

   site-a ストレージクラスターのモニターノードで以下のコマンドを実行し、イメージを **site-b** から **site-a** に再同期します。

   構文

   ```
   rbd mirror image resync POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image resync data/image1
   Flagged image for resync from primary
   [root@rbd-client ~]# rbd mirror image resync data/image2
   Flagged image for resync from primary
   ```

5. しばらくしたら、状態が **up+replaying** かをチェックして、イメージの最同期が完了していることを確認します。**site-a** ストレージクラスターのモニターノードで以下のコマンドを実行して、イメージの状態を確認します。

   構文

   ```
   rbd mirror image status POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   [root@rbd-client ~]# rbd mirror image status data/image2
   ```

6. **site-b** ストレージクラスターの Ceph Monitor ノードで以下のコマンドを実行して、 **site-b** ストレージクラスターのイメージをデモートします。

構文

```
rbd mirror image demote POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image demote data/image1
[root@rbd-client ~]# rbd mirror image demote data/image2
```

注記

複数のセカンダリーストレージクラスターがある場合に、上記の実行は、プロモートされたセカンダリーストレージクラスターからだけで結構です。

7. **site-a** ストレージクラスターの Ceph Monitor ノードで以下のコマンドを実行して、 **site-a** ストレージクラスターに配置されていた、以前のプライマリーイメージをプロモートします。

構文

```
rbd mirror image promote POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image promote data/image1
[root@rbd-client ~]# rbd mirror image promote data/image2
```

8. **site-a** ストレージクラスターの Ceph Monitor ノードからイメージのステータスを確認します。状態として **up+stopped**、説明として **local image is primary** と表示されるはずです。

構文

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

例

```
[root@rbd-client ~]# rbd mirror image status data/image1
image1:
  global_id:   08027096-d267-47f8-b52e-59de1353a034
  state:       up+stopped
  description: local image is primary
  last_update: 2019-04-22 11:14:51
[root@rbd-client ~]# rbd mirror image status data/image2
image2:
  global_id:   596f41bc-874b-4cd4-aefe-4929578cc834
  state:       up+stopped
  description: local image is primary
  last_update: 2019-04-22 11:14:51
```

6.5.7. 双方向ミラーリングの削除

フェイルバックが完了したら、双方向ミラーリングを削除し、Ceph ブロックデバイスのミラーリングサービスを無効にできます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. **site-a** ストレージクラスターから、ピアとしての **site-b** ストレージクラスターを削除します。

   例

   ```
   [root@rbd-client ~]# rbd mirror pool peer remove data client.remote@remote --cluster local
   [root@rbd-client ~]# rbd --cluster site-a mirror pool peer remove data client.site-b@site-b -n
   client.site-a
   ```

2. **site-a** クライアントで **rbd-mirror** デーモンを停止して無効にします。

   構文

   ```
   systemctl stop ceph-rbd-mirror@CLIENT_ID
   systemctl disable ceph-rbd-mirror@CLIENT_ID
   systemctl disable ceph-rbd-mirror.target
   ```

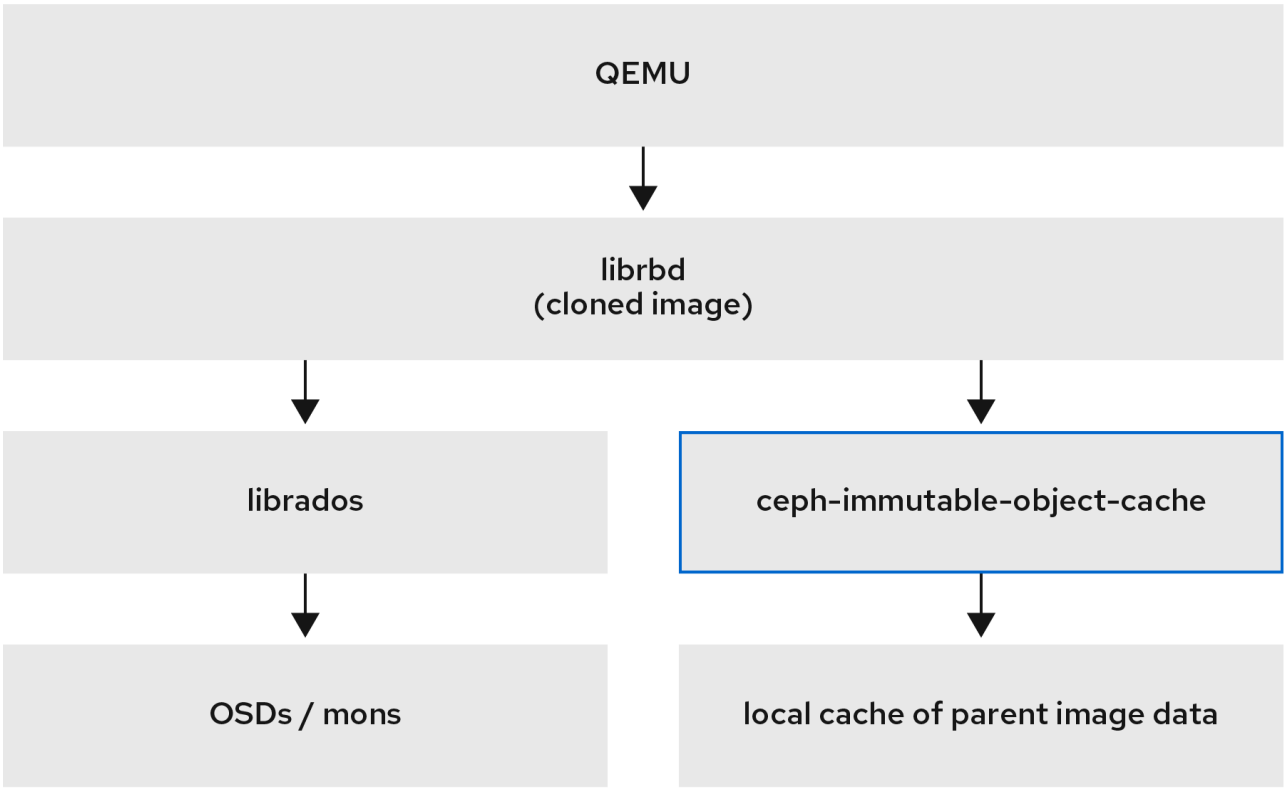
   例

   ```
   [root@rbd-client ~]# systemctl stop ceph-rbd-mirror@site-a
   [root@rbd-client ~]# systemctl disable ceph-rbd-mirror@site-a
   [root@rbd-client ~]# systemctl disable ceph-rbd-mirror.target
   ```

第7章 CEPH-IMMUTABLE-OBJECT-CACHE デーモンの管理

ストレージ管理者は、**ceph-immutable-object-cache** デーモンを使用してローカルディスクの親イメージコンテンツをキャッシュします。このキャッシュはローカルのキャッシュディレクトリーに保存されます。今後の読み取りは、そのデータでローカルキャッシュを使用します。

図7.1 Ceph イミュータブルなキャッシュデーモン



7.1. CEPH-IMMUTABLE-OBJECT-CACHE デーモンの説明

クローン作成したブロックデバイスイメージは通常、親イメージの一部だけを変更します。たとえば、仮想デスクトップインターフェイス (VDI) では、仮想マシンは同じベースイメージからクローンされ、最初はホスト名と IP アドレスだけが異なります。起動中に、親イメージのローカルキャッシュを使用すると、キャッシュホストでの読み取りが加速されます。この変更で、クライアントからクラスターのネットワークトラフィックが減少します。

ceph-immutable-object-cache デーモンを使用する理由

ceph-immutable-object-cache デーモンは Red Hat Ceph Storage に含まれます。Red Hat Ceph Storage は、スケーラブルなオープンソースの分散ストレージシステムです。これは、**ceph.conf** ファイルを検索するデフォルトの検索パスに依存する、RADOS プロトコルでローカルクラスターに接続し、**/etc/ceph/CLUSTER.conf**、**/etc/ceph/CLUSTER.keyring** および **/etc/ceph/CLUSTER.NAME.keyring** などの認証情報やアドレスを監視します。**CLUSTER** はクラスターの人間が判読できる名前に、**NAME** は例として接続する RADOS ユーザー (**client.ceph-immutable-object-cache**) に置き換えます。

デーモンの主要なコンポーネント

ceph-immutable-object-cache デーモンには以下の部分があります。

- ドメインソケットベースのプロセス通信 (IPC): デーモンは、起動時にローカルドメインソケットをリッスンし、**librbd** クライアントからの接続を待ちます。

- 最近使用された (LRU) ベースのプロモーションまたはデモポリシー: デーモンは、各キャッシュファイルの cache-hits のインメモリー統計を保持します。容量が、設定されたしきい値に達すると、コールドキャッシュをデモートします。

- ファイルベースのキャッシュストア: このデーモンは、簡単なファイルベースのキャッシュストアを保持します。プロモート時には、RADOS オブジェクトは RADOS クラスターからフェッチされ、ローカルのキャッシュディレクトリーに保存されます。

クローン作成された各 RBD イメージを開くと、**librbd** は Unix ドメインソケットを介してキャッシュデーモンへの接続を試みます。正常に接続されたら、**librbd** は、後続の読み取りでデーモンと対話します。キャッシュされない読み取りがある場合は、デーモンは RADOS オブジェクトをローカルキャッシュディレクトリーにプロモートし、そのオブジェクトの次の読み取りがキャッシュから提供されます。また、このデーモンは、容量に制限がある場合など、必要に応じてコールドキャッシュファイルをエビクトし、単純な LRU の統計を維持します。

注記

パフォーマンスを改善するには、SSD を基礎となるストレージとして使用します。

7.2. CEPH-IMMUTABLE-OBJECT-CACHE デーモンの設定

ceph-immutable-object-cache は、Ceph クラスター間の RADOS オブジェクトのオブジェクトキャッシュ用のデーモンです。

重要

ceph-immutable-object-cache デーモンを使用するには、RADOS クラスターを接続できる必要があります。

デーモンは、オブジェクトをローカルディレクトリーにプロモートします。これらのキャッシュオブジェクトは、今後の読み取りに対応します。**ceph-immutable-object-cache** パッケージをインストールしてデーモンを設定できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- キャッシュには最低でも SSD が 1 つある。

手順

1. RBD 共有の読み取り専用の親イメージキャッシュを有効にします。**/etc/ceph/ceph.conf** ファイルの **[client]** の下に以下のパラメーターを追加します。

   例

   ```
   [root@ceph-host01 ~]# vi /etc/ceph/ceph.conf

   [client]
   rbd parent cache enabled = true
   rbd plugins = parent_cache
   ```

   クラスターを再起動します。

2. **ceph-immutable-object-cache** パッケージをインストールします。

   例

   ```
   [root@ceph-host1 ~]# dnf install ceph-immutable-object-cache
   ```

3. 一意の Ceph ユーザー ID (キーリング) を作成します。

   構文

   ```
   ceph auth get-or-create client.ceph-immutable-object-cache.USER_NAME mon 'profile rbd'
   osd 'profile rbd-read-only'
   ```

   例

   ```
   [root@ceph-host1 ~]# ceph auth get-or-create client.ceph-immutable-object-cache.user mon
   'profile rbd' osd 'profile rbd-read-only'

   [client.ceph-immutable-object-cache.user]
    key = AQCVPH1gFgHRAhAAp8ExRIsoxQK4QSYSRoVJLw==
   ```

   このキーリングをコピーします。

4. **/etc/ceph** ディレクトリーで、ファイルを作成し、キーリングを貼り付けます。

   例

   ```
   [root@ceph-host1 ]# vi /etc/ceph/ceph.client.ceph-immutable-object-cache.user.keyring

   [client.ceph-immutable-object-cache.user]
    key = AQCVPH1gFgHRAhAAp8ExRIsoxQK4QSYSRoVJLw
   ```

5. デーモンを有効にします。

   構文

   ```
   systemctl enable ceph-immutable-object-cache@ceph-immutable-object-
   cache.USER_NAME
   ```

   USER_NAME をデーモンインスタンスとして指定します。

   例

   ```
   [root@ceph-host1 ~]# systemctl enable ceph-immutable-object-cache@ceph-immutable-
   object-cache.user

   Created symlink /etc/systemd/system/ceph-immutable-object-cache.target.wants/ceph-
   immutable-object-cache@ceph-immutable-object-cache.user.service →
   /usr/lib/systemd/system/ceph-immutable-object-cache@.service.
   ```

6. **ceph-immutable-object-cache** デーモンを起動します。

   構文

```
systemctl start ceph-immutable-object-cache@ceph-immutable-object-cache.USER_NAME
```

例

```
[root@ceph-host1 ~]# systemctl start ceph-immutable-object-cache@ceph-immutable-object-cache.user
```

検証

- 設定のステータスを確認します。

  構文

```
systemctl status ceph-immutable-object-cache@ceph-immutable-object-cache.USER_NAME
```

  例

```
[root@ceph-host1 ~]# systemctl status ceph-immutable-object-cache@ceph-immutable-object-cache.user

● ceph-immutable-object-cache@ceph-immutable-object-cache.user>
  Loaded: loaded (/usr/lib/systemd/system/ceph-immutable-objec>
  Active: active (running) since Mon 2021-04-19 13:49:06 IST; >
Main PID: 85020 (ceph-immutable-)
   Tasks: 15 (limit: 49451)
  Memory: 8.3M
  CGroup: /system.slice/system-ceph\x2dimmutable\x2dobject\x2d>
          └─85020 /usr/bin/ceph-immutable-object-cache -f --cl>
```

7.3. CEPH-IMMUTABLE-OBJECT-CACHE デーモンの一般的な設定

ceph-immutable-object-cache デーモンの一般的な重要設定を一部表示します。

immutable_object_cache_sock

説明

librbd クライアントと ceph-immutable-object-cache デーモンの間の通信に使用されるドメインソケットへのパス。

型

String

デフォルト

/var/run/ceph/immutable_object_cache_sock

immutable_object_cache_path

説明

イミュータブルなオブジェクトキャッシュデータディレクトリー。

型

String

デフォルト

/tmp/ceph_immutable_object_cache

immutable_object_cache_max_size

説明

イミュータブルなキャッシュの最大サイズ。

型

サイズ

デフォルト

1G

immutable_object_cache_watermark

説明

キャッシュの最高水位標。値は 0 から 1 までです。キャッシュサイズがこのしきい値に達すると、デーモンは LRU 統計に基づいてコールドマイグレーションキャッシュを削除します。

型

浮動小数点 (Float)

デフォルト

0.9

7.4. CEPH-IMMUTABLE-OBJECT-CACHE デーモンの QOS 設定

ceph-immutable-object-cache デーモンは、上記の設定をサポートするスロットリングをサポートします。

immutable_object_cache_qos_schedule_tick_min

説明

イミュータブルオブジェクトキャッシュの最小 schedule_tick。

型

ミリ秒

デフォルト

50

immutable_object_cache_qos_iops_limit

説明

ユーザー定義のイミュータブルなオブジェクトキャッシュ IO 操作の上限 (秒単位)。

型

Integer

デフォルト

0

immutable_object_cache_qos_iops_burst

説明

イミュータブルオブジェクトキャッシュ I/O 操作のユーザー定義のバースト制限。

型

Integer

デフォルト

0

immutable_object_cache_qos_iops_burst_seconds

説明

イミュータブルなオブジェクトキャッシュ I/O 操作のユーザー定義のバースト期間 (秒単位)。

型

秒

デフォルト

1

immutable_object_cache_qos_bps_limit

説明

ユーザー定義のイミュータブルなオブジェクトキャッシュ IO バイトの上限 (秒単位)。

型

Integer

デフォルト

0

immutable_object_cache_qos_bps_burst

説明

イミュータブルオブジェクトキャッシュ I バイトのユーザー定義のバースト制限。

型

Integer

デフォルト

0

immutable_object_cache_qos_bps_burst_seconds

説明

読み取り操作の必要なバースト制限。

型

秒

デフォルト

1

第8章 RBD カーネルモジュール

ストレージ管理者は、**rbd** カーネルモジュールを使用して Ceph ブロックデバイスにアクセスできます。ブロックデバイスをマップして、マッピングを解除し、これらのマッピングを表示できます。また、**rbd** カーネルモジュールを使用してイメージのリストを取得することもできます。

重要

Red Hat Enterprise Linux (RHEL) 以外の Linux ディストリビューションのカーネルクライアントは使用できますが、サポートされていません。これらのカーネルクライアントの使用時に問題がストレージクラスターにある場合には、Red Hat は対応しますが、根本的な原因がカーネルクライアント側にある場合は、ソフトウェアベンダーが問題に対処する必要があります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

8.1. CEPH ブロックデバイスの作成および LINUX カーネルモジュールクライアントからのデバイスの使用

ストレージ管理者は、Red Hat Ceph Storage ダッシュボードで Linux カーネルモジュールクライアントの Ceph ブロックデバイスを作成できます。システム管理者は、コマンドラインを使用して Linux クライアントでそのブロックデバイスをマップし、パーティション作成、フォーマットおよびマウントが可能です。その後、そのファイルの読み取りと書き込みが可能になります。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- Red Hat Enterprise Linux クライアント。

8.1.1. ダッシュボードを使用した Linux カーネルモジュールクライアントの Ceph ブロックデバイスの作成

ダッシュボードの Web インターフェイスを使用して、サポート対象機能だけを有効にし、Linux カーネルモジュールクライアント専用の Ceph ブロックデバイスを作成できます。

カーネルモジュールクライアントは、ディープフラット化、レイヤー、排他的ロック、オブジェクトマップ、Fast diff などの機能をサポートします。

オブジェクトマップ、Fast diff、およびディープフラット化機能には、Red Hat Enterprise Linux 8.2 以降が必要です。

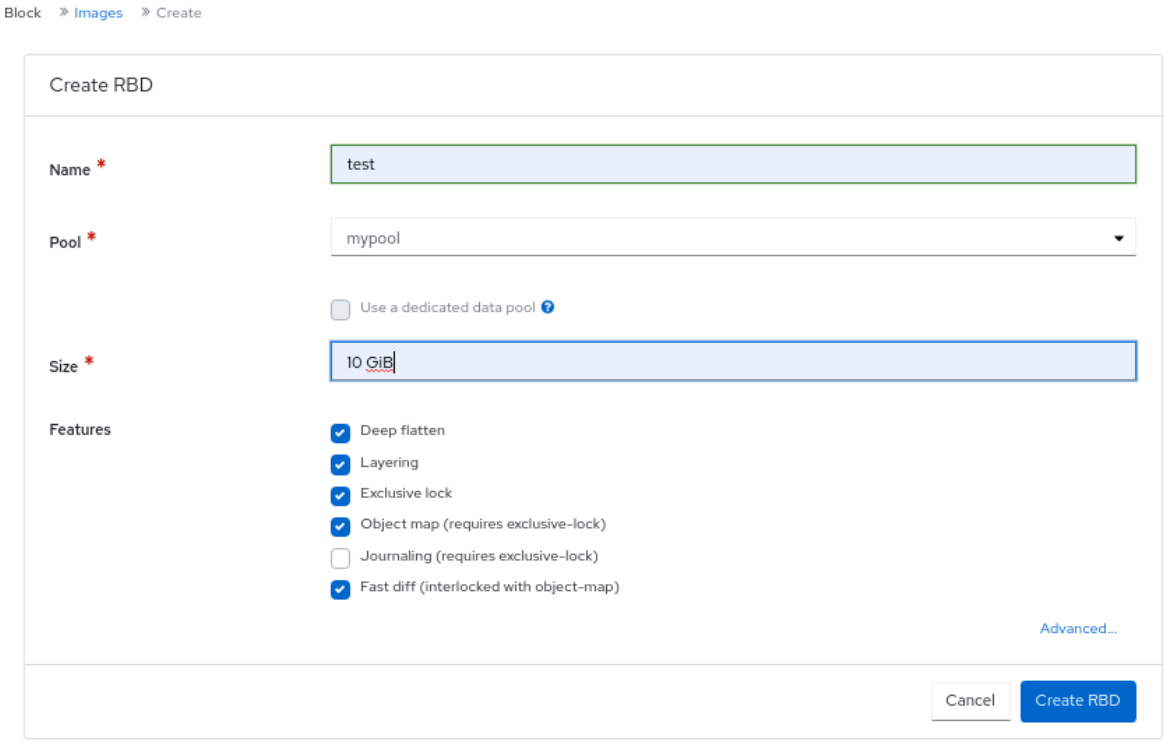
前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- レプリケートされた RBD プールが作成され、有効になっている。

手順

1. **Block** ドロップダウンメニューから、**イメージ** を選択します。

2. **Create** をクリックします。
3. **Create RBD** ウィンドウでイメージ名を入力し、RBD 対応プールを選択し、サポート対象の機能を選択します。



4. **Create RBD** をクリックします。

検証

- イメージが正常に作成されたことを示す通知が表示されます。

関連情報

- 詳細は、**Red Hat Ceph Storage ブロックデバイスガイド**の **コマンドラインを使用した Linux への Ceph ブロックデバイスのマッピングとマウント** を参照してください。
- 詳細は、**Red Hat Ceph Storage ダッシュボードガイド**を参照してください。

8.1.2. コマンドラインを使用した Linux への Ceph ブロックデバイスのマッピングとマウント

Linux **rbd** カーネルモジュールを使用して、Red Hat Enterprise Linux クライアントから Ceph ブロックデバイスをマッピングできます。マッピング後には、パーティション、フォーマット、およびマウントができるため、ファイルに書き込みができます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ダッシュボードを使用して Linux カーネルモジュールクライアントの Ceph ブロックデバイスが作成している。
- Red Hat Enterprise Linux クライアント。

手順

1. Red Hat Enterprise Linux クライアントノードで、Red Hat Ceph Storage 6 Tools リポジトリーを有効にします。

```
[root@rbd-client ~]# subscription-manager repos --enable=rhceph-6-tools-for-rhel-9-x86_64-rpms
```

2. **ceph-common** RPM パッケージをインストールします。

```
[root@rbd-client ~]# dnf install ceph-common
```

3. Ceph 設定ファイルを Monitor ノードからクライアントノードにコピーします。

構文

```
scp root@MONITOR_NODE:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
```

例

```
[root@rbd-client ~]# scp root@cluster1-node2:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
root@192.168.0.32's password:
ceph.conf                                     100%  497   724.9KB/s   00:00
[root@client1 ~]#
```

4. キーファイルを Monitor ノードからクライアントノードにコピーします。

構文

```
scp root@MONITOR_NODE:/etc/ceph/ceph.client.admin.keyring
/etc/ceph/ceph.client.admin.keyring
```

例

```
[root@rbd-client ~]# scp root@cluster1-node2:/etc/ceph/ceph.client.admin.keyring
/etc/ceph/ceph.client.admin.keyring
root@192.168.0.32's password:
ceph.client.admin.keyring                     100%  151   265.0KB/s   00:00
[root@client1 ~]#
```

5. イメージをマップします。

構文

```
rbd map --pool POOL_NAME IMAGE_NAME --id admin
```

例

```
[root@rbd-client ~]# rbd map --pool block-device-pool image1 --id admin
/dev/rbd0
[root@client1 ~]#
```

6. ブロックデバイスにパーティションテーブルを作成します。

構文

```
parted /dev/MAPPED_BLOCK_DEVICE mklabel msdos
```

例

```
[root@rbd-client ~]# parted /dev/rbd0 mklabel msdos
Information: You may need to update /etc/fstab.
```

7. XFS ファイルシステムのパーティションを作成します。

構文

```
parted /dev/MAPPED_BLOCK_DEVICE mkpart primary xfs 0% 100%
```

例

```
[root@rbd-client ~]# parted /dev/rbd0 mkpart primary xfs 0% 100%
Information: You may need to update /etc/fstab.
```

8. パーティションをフォーマットします。

構文

```
mkfs.xfs /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER
```

例

```
[root@rbd-client ~]# mkfs.xfs /dev/rbd0p1
meta-data=/dev/rbd0p1            isize=512    agcount=16, agsize=163824 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=2621184, imaxpct=25
         =                       sunit=16     swidth=16 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=16 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

9. 新しいファイルシステムをマウントするディレクトリーを作成します。

構文

```
mkdir PATH_TO_DIRECTORY
```

例

```
[root@rbd-client ~]# mkdir /mnt/ceph
```

10. ファイルシステムをマウントします。

    構文

    ```
    mount /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER
    PATH_TO_DIRECTORY
    ```

    例

    ```
    [root@rbd-client ~]# mount /dev/rbd0p1 /mnt/ceph/
    ```

11. ファイルシステムがマウントされ、正しいサイズを表示していることを確認します。

    構文

    ```
    df -h PATH_TO_DIRECTORY
    ```

    例

    ```
    [root@rbd-client ~]# df -h /mnt/ceph/
    Filesystem      Size  Used Avail Use% Mounted on
    /dev/rbd0p1      10G  105M  9.9G   2% /mnt/ceph
    ```

関連情報

- 詳細は、**ダッシュボードを使用した Linux カーネルモジュールクライアントの Ceph ブロックデバイスの作成** を参照してください。
- 詳細は、Red Hat Enterprise Linux 8 のファイルシステムの管理 を参照してください。
- 詳細は、Red Hat Enterprise Linux 7 の ストレージ管理ガイド を参照してください。

8.2. ブロックデバイスのマッピング

rbd を使用して、イメージ名をカーネルモジュールにマッピングします。イメージ名、プール名、およびユーザー名を指定する必要があります。**rbd** がまだロードされていない場合は、RBD カーネルモジュールを読み込みます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. イメージの一覧を返します。

   例

   ```
   [root@rbd-client ~]# rbd list
   ```

2. イメージをマップするための 2 つのオプションは次のとおりです。

   - イメージ名をカーネルモジュールにマッピングします。

     構文

     ```
     rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME
     ```

     例

     ```
     [root@rbd-client ~]# rbd device map rbd/myimage --id admin
     ```

   - **cephx** 認証を使用する場合に、キーリングか、シークレットを含むファイルでシークレットを指定します。

     構文

     ```
     [root@rbd-client ~]# rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME --keyring PATH_TO_KEYRING
     ```

     または

     ```
     [root@rbd-client ~]# rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME --keyfile PATH_TO_FILE
     ```

8.3. マップされたブロックデバイスの表示

rbd コマンドを使用して、カーネルモジュールにマップされるブロックデバイスイメージを表示できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. マップされたブロックデバイスを表示します。

   ```
   [root@rbd-client ~]# rbd device list
   ```

8.4. ブロックデバイスのマッピング解除

unmap オプションを使用してデバイス名を指定し、**rbd** コマンドでブロックデバイスイメージのマッピングを解除できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

- マッピングされたイメージ。

手順

1. デバイスの仕様を取得します。

   例

   ```
   [root@rbd-client ~]# rbd device list
   ```

2. ブロックデバイスイメージのマッピングを解除します。

   構文

   ```
   rbd device unmap /dev/rbd/POOL_NAME/IMAGE_NAME
   ```

   例

   ```
   [root@rbd-client ~]# rbd device unmap /dev/rbd/pool1/image1
   ```

8.5. 同じプール内の分離された名前空間内でのイメージの分離

OpenStack または OpenShift Container Storage などの上位システムなしで Ceph ブロックデバイスを直接使用する場合、特定のブロックデバイスイメージへのユーザーアクセスを制限することができませんでした。CephX 機能と組み合わせると、ユーザーを特定のプール名前空間に制限して、イメージへのアクセスを制限することができます。

新規レベルのアイデンティティーである RADOS 名前空間を使用してオブジェクトを特定し、プール内の rados クライアント間を分離できます。たとえば、クライアントは、クライアント向けの名前空間でのみ完全なパーミッションが割り当てられます。これにより、テナントごとに異なる RADOS クライアントが使用されるので、多くの異なるテナントが独自のブロックデバイスイメージにアクセスしているブロックデバイスに特に便利です。

同じプール内の分離された名前空間内でブロックデバイスイメージを分離することができます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- すべてのカーネルを 4x および librbd、全クライアントの Librados にアップグレードする。
- モニターおよびクライアントノードへの root レベルのアクセスがある。

手順

1. **rbd** プールを作成します。

   構文

   ```
   ceph osd pool create POOL_NAME PG_NUM
   ```

   例

   ■

```
[ceph: root@host01 /]# ceph osd pool create mypool 100
pool 'mypool' created
```

2. **rbd** プールを RBD アプリケーションに関連付けます。

   構文

   ```
   ceph osd pool application enable POOL_NAME rbd
   ```

   例

   ```
   [ceph: root@host01 /]# ceph osd pool application enable mypool rbd
   enabled application 'rbd' on pool 'mypool'
   ```

3. RBD アプリケーションでプールを初期化します。

   構文

   ```
   rbd pool init -p POOL_NAME
   ```

   例

   ```
   [ceph: root@host01 /]# rbd pool init -p mypool
   ```

4. 2 つの名前空間を作成します。

   構文

   ```
   rbd namespace create --namespace NAMESPACE
   ```

   例

   ```
   [ceph: root@host01 /]# rbd namespace create --namespace namespace1

   [ceph: root@host01 /]# rbd namespace create --namespace namespace2

   [ceph: root@host01 /]# rbd namespace ls --format=json
   [{"name":"namespace2"},{"name":"namespace1"}]
   ```

5. ユーザー 2 つにこの名前空間へのアクセスを割り当てます。

   構文

   ```
   ceph auth get-or-create client.USER_NAME mon 'profile rbd' osd 'profile rbd pool=rbd
   namespace=NAMESPACE' -o /etc/ceph/client.USER_NAME.keyring
   ```

   例

   ```
   [ceph: root@host01 /]# ceph auth get-or-create client.testuser mon 'profile rbd' osd 'profile
   rbd pool=rbd namespace=namespace1' -o /etc/ceph/client.testuser.keyring
   ```

```
[ceph: root@host01 /]# ceph auth get-or-create client.newuser mon 'profile rbd' osd 'profile
rbd pool=rbd namespace=namespace2' -o /etc/ceph/client.newuser.keyring
```

6. クライアントのキーを取得します。

構文

```
ceph auth get client.USER_NAME
```

例

```
[ceph: root@host01 /]# ceph auth get client.testuser

[client.testuser]
 key = AQDMp61hBf5UKRAAgjQ2In0Z3uwAase7mrlKnQ==
 caps mon = "profile rbd"
 caps osd = "profile rbd pool=rbd namespace=namespace1"
exported keyring for client.testuser

[ceph: root@host01 /]# ceph auth get client.newuser

[client.newuser]
 key = AQDfp61hVfLFHRAA7D80ogmZl80ROY+AUG4A+Q==
 caps mon = "profile rbd"
 caps osd = "profile rbd pool=rbd namespace=namespace2"
exported keyring for client.newuser
```

7. ブロックデバイスイメージを作成し、プール内の事前定義済みの名前空間を使用します。

構文

```
rbd create --namespace NAMESPACE IMAGE_NAME --size SIZE_IN_GB
```

例

```
[ceph: root@host01 /]# rbd create --namespace namespace1 image01 --size 1G

[ceph: root@host01 /]# rbd create --namespace namespace2 image02 --size 1G
```

8. オプション: 名前空間および関連付けられたイメージの詳細を取得します。

構文

```
rbd --namespace NAMESPACE ls --long
```

例

```
[ceph: root@host01 /]#  rbd --namespace namespace1 ls --long
NAME    SIZE  PARENT FMT PROT LOCK
image01 1 GiB          2
```

```
[ceph: root@host01 /]# rbd --namespace namespace2 ls --long
NAME    SIZE  PARENT FMT PROT LOCK
image02 1 GiB          2
```

9. Ceph 設定ファイルを Monitor ノードからクライアントノードにコピーします。

```
scp /etc/ceph/ceph.conf root@CLIENT_NODE:/etc/ceph/
```

例

```
[ceph: root@host01 /]# scp /etc/ceph/ceph.conf root@host02:/etc/ceph/

root@host02's password:
ceph.conf                                                    100%  497   724.9KB/s   00:00
```

10. 管理キーリングを Ceph Monitor ノードからクライアントノードにコピーします。

構文

```
scp /etc/ceph/ceph.client.admin.keyring root@CLIENT_NODE:/etc/ceph
```

例

```
[ceph: root@host01 /]# scp /etc/ceph/ceph.client.admin.keyring root@host02:/etc/ceph/

root@host02's password:
ceph.client.admin.keyring                                    100%  151   265.0KB/s   00:00
```

11. ユーザーのキーリングを Ceph Monitor ノードからクライアントノードにコピーします。

構文

```
scp /etc/ceph/ceph.client.USER_NAME.keyring root@CLIENT_NODE:/etc/ceph/
```

例

```
[ceph: root@host01 /]# scp /etc/ceph/client.newuser.keyring root@host02:/etc/ceph/

[ceph: root@host01 /]# scp /etc/ceph/client.testuser.keyring root@host02:/etc/ceph/
```

12. ブロックデバイスイメージをマッピングします。

構文

```
rbd map --name NAMESPACE IMAGE_NAME -n client.USER_NAME --keyring
/etc/ceph/client.USER_NAME.keyring
```

例

```
[ceph: root@host01 /]# rbd map --namespace namespace1 image01  -n  client.testuser --
keyring=/etc/ceph/client.testuser.keyring
```

```
/dev/rbd0

[ceph: root@host01 /]# rbd map --namespace namespace2 image02 -n client.newuser --
keyring=/etc/ceph/client.newuser.keyring

/dev/rbd1
```

これにより、同じプール内の他の名前空間のユーザーにはアクセスできません。

例

```
[ceph: root@host01 /]# rbd map --namespace namespace2 image02  -n  client.testuser --
keyring=/etc/ceph/client.testuser.keyring

rbd: warning: image already mapped as /dev/rbd1
rbd: sysfs write failed
rbd: error asserting namespace: (1) Operation not permitted
In some cases useful info is found in syslog - try "dmesg | tail".
2021-12-06 02:49:08.106 7f8d4fde2500 -1 librbd::api::Namespace: exists: error asserting
namespace: (1) Operation not permitted
rbd: map failed: (1) Operation not permitted

[ceph: root@host01 /]# rbd map --namespace namespace1 image01 -n client.newuser --
keyring=/etc/ceph/client.newuser.keyring

rbd: warning: image already mapped as /dev/rbd0
rbd: sysfs write failed
rbd: error asserting namespace: (1) Operation not permitted
In some cases useful info is found in syslog - try "dmesg | tail".
2021-12-03 12:16:24.011 7fcad776a040 -1 librbd::api::Namespace: exists: error asserting
namespace: (1) Operation not permitted
rbd: map failed: (1) Operation not permitted
```

13. デバイスを確認します。

例

```
[ceph: root@host01 /]# rbd showmapped

id pool namespace      image   snap device
0  rbd  namespace1  image01 -  /dev/rbd0
1  rbd  namespace2   image02 -  /dev/rbd1
```

第9章 CEPH ブロックデバイス PYTHON モジュールの使用

rbd python モジュールでは、Ceph ブロックデバイスイメージにファイルのようにアクセスできます。この組み込みツールを使用するには、**rbd** モジュールおよび **rados** Python モジュールをインポートします。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- ノードへのルートレベルのアクセス。

手順

1. RADOS に接続し、IO コンテキストを開きます。

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

2. イメージの作成に使用する **:class:rbd.RBD** オブジェクトをインスタンス化します。

```
rbd_inst = rbd.RBD()
size = 4 * 1024**3  # 4 GiB
rbd_inst.create(ioctx, 'myimage', size)
```

3. イメージで I/O を実行するには、**:class:rbd.Image** オブジェクトをインスタンス化します。

```
image = rbd.Image(ioctx, 'myimage')
data = 'foo' * 200
image.write(data, 0)
```

これにより、イメージの最初の 600 バイトに 'foo' が書き込まれます。データは **:type:unicode** に指定できない点に注意してください。**librbd** は **:c:type:char** よりも幅の広い文字の処理方法を認識していません。

4. イメージ、IO コンテキスト、および RADOS への接続を終了します。

```
image.close()
ioctx.close()
cluster.shutdown()
```

念のために、これらの呼び出しごとに、個別の **:finally** ブロックを割り当てる必要があります。

```
import rados
import rbd

cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
```

```
        rbd_inst.create(ioctx, 'myimage', size)
        image = rbd.Image(ioctx, 'myimage')
        try:
            data = 'foo' * 200
            image.write(data, 0)
        finally:
            image.close()
    finally:
        ioctx.close()
finally:
    cluster.shutdown()
```

これは面倒な場合があるので、自動的に終了またはシャットダウンするコンテキストマネージャーとして**Rados**、**Ioctx** および **Image** クラスを使用できます。これらのクラスをコンテキストマネージャーとして使用すると、上記の例は以下のようになります。

```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

付録A CEPH ブロックデバイス設定の参照

ストレージ管理者は、利用可能なさまざまなオプションで、Ceph ブロックデバイスの動作を微調整できます。この参照を使用して、デフォルトの Ceph ブロックデバイスオプションや Ceph ブロックデバイスキャッシュオプションなどを表示できます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

A.1. ブロックデバイスのデフォルトオプション

イメージを作成するデフォルト設定を上書きできます。Ceph は、**2** のフォーマットでイメージを作成し、ストライピングを行わずにイメージを作成します。

rbd_default_format

説明

その他の形式が指定されていない場合のデフォルト形式 (**2**)。フォーマット **1** は、**librbd** およびカーネルモジュールの全バージョンと互換性がある新しいイメージの元の形式ですが、クローンなどの新しい機能をサポートしません。**2** 形式は、**librbd** およびカーネルモジュールバージョン 3.11 以降でサポートされます (ストライピングを除く)。フォーマット **2** により、クローン作成のサポートが追加され、今後より簡単に機能性を持たせることができます。

型

Integer

デフォルト

2

rbd_default_order

説明

他の順序が指定されていない場合のデフォルトの順番です。

型

Integer

デフォルト

22

rbd_default_stripe_count

説明

他のストライプ数が指定されていない場合、デフォルトのストライプ数。デフォルト値を変更するには、v2 機能の削除が必要です。

型

64 ビット未署名の整数

デフォルト

0

rbd_default_stripe_unit

説明

他のストライプユニットが指定されていない場合は、デフォルトのストライプユニットです。単位を **0** (オブジェクトサイズ) から変更するには、v2 ストライピング機能が必要です。

型

64 ビット未署名の整数

デフォルト

0

rbd_default_features

説明

ブロックデバイスイメージの作成時にデフォルトの機能が有効になります。この設定は、2 つのイメージのみに適用されます。設定は以下のとおりです。
1: **レイヤーサポート。**レイヤー化により、クローンを使用できます。

2: **v2 サポートのストライピング。**ストライピングは、データを複数のオブジェクト全体に分散します。ストライピングは、連続の読み取り/書き込みワークロードの並行処理に役立ちます。

4: **排他的ロックのサポート。**有効にすると、書き込みを行う前にクライアントがオブジェクトのロックを取得する必要があります。

8: **オブジェクトマップのサポート。**ブロックデバイスはシンプロビジョニングされており、実際に存在するデータのみを保存します。オブジェクトマップのサポートは、実際に存在するオブジェクト (ドライブに格納されているデータ) を追跡するのに役立ちます。オブジェクトマップを有効にすると、クローン作成用の I/O 操作が高速化され、スパースに設定されたイメージのインポートおよびエクスポートが実行されます。

16: **fast-diff サポート。**fast-diff サポートは、オブジェクトマップのサポートと排他的ロックのサポートに依存します。別の属性をオブジェクトマップに追加して、イメージのスナップショット間の差異の生成と、スナップショットの実際のデータ使用量がはるかに速くなります。

32: **deep-flatten サポート。**deep-flatten を使用すると、イメージ自体に加えて、**rbd flatten** がイメージのすべてのスナップショットで機能します。これを使用しないと、イメージのスナップショットは親に依存するため、スナップショットが削除されるまで親は削除できません。deep-flatten は、スナップショットがある場合でも、クローンから親を切り離します。

64: **ジャーナリングサポート。**ジャーナリングは、イメージの実行順にイメージへの変更をすべて記録します。これにより、リモートイメージのクラッシュ調整ミラーがローカルで使用できるようになります。

有効な機能は、数値設定の合計です。

型

Integer

デフォルト

61: レイヤー化、exclusive-lock、object-map、fast-diff、および deep-flatten が有効にされます。

重要

現在のデフォルト設定は RBD カーネルドライバーや古い RBD クライアントと互換性がありません。

rbd_default_map_options

説明

ほとんどのオプションは、主にデバッグおよびベンチマークに役立ちます。詳細は、**Map Options** の **man rbd** を参照してください。

型

String

デフォルト

""

A.2. ブロックデバイスの一般オプション

rbd_op_threads

説明

ブロックデバイス操作スレッドの数。

型

Integer

デフォルト

1

警告

rbd_op_threads のデフォルト値を変更しないでください。これは、**1** を超える値に設定するとデータが破損する可能性があるためです。

rbd_op_thread_timeout

説明

ブロックデバイス操作スレッドのタイムアウト (秒単位)。

型

Integer

デフォルト

60

rbd_non_blocking_aio

説明

true の場合、Ceph はブロックを防ぐためにワーカースレッドからブロックデバイスの非同期 I/O 操作を処理します。

型

Boolean

デフォルト

true

rbd_concurrent_management_ops

説明

フライトでの同時管理操作の最大数 (イメージの削除またはサイズ変更など)。

型

Integer

デフォルト

10

rbd_request_timed_out_seconds

説明

メンテナンス要求がタイムアウトするまでの秒数。

型

Integer

デフォルト

30

rbd_clone_copy_on_read

説明

true に設定すると、コピーオン読み取りのクローン作成が有効になります。

型

Boolean

デフォルト

false

rbd_enable_alloc_hint

説明

true の場合、割り当てヒントは有効にされ、ブロックデバイスは OSD バックエンドにヒントを発行し、予想されるサイズオブジェクトを示します。

型

Boolean

デフォルト

true

rbd_skip_partial_discard

説明

true の場合、オブジェクト内で範囲を破棄しようとすると、ブロックデバイスは範囲のゼロを省略します。

型

Boolean

デフォルト

true

rbd_tracing

説明

Linux Trace Toolkit Next Generation User Space Tracer (LTTng-UST) トレースポイントを有効にするには、このオプションを **true** に設定します。詳細は、RBD Replay 機能を使用した RADOS Block Device (RBD) ワークロードのトレース を参照してください。

型

Boolean

デフォルト

false

rbd_validate_pool

説明

RBD の互換性について空のプールを検証するには、このオプションを **true** に設定します。

型

Boolean

デフォルト

true

rbd_validate_names

説明

イメージの仕様を検証するには、このオプションを **true** に設定します。

型

Boolean

デフォルト

true

A.3. ブロックデバイスキャッシュオプション

Ceph ブロックデバイスのユーザー空間実装 (**librbd**) は Linux ページキャッシュを利用できないため、**RBD キャッシュ** と呼ばれる独自のインメモリーキャッシュが含まれます。Ceph ブロックデバイスのキャッシュは、適切なハードディスクキャッシングと同様に動作します。オペレーティングシステムがバリアまたはフラッシュ要求を送信すると、ダーティーデータはすべて Ceph OSD に書き込まれます。つまり、フラッシュを適切に送信する仮想マシン (Linux カーネルバージョン 2.6.32 以上) とともに、ライトバックキャッシュを使用すると安全ではありません。キャッシュは Least Recently Used (LRU) アルゴリズムを使用し、ライトバックモードでは、スループット向上のために連続したリクエストを結合できます。

Ceph ブロックデバイスは、ライトバックキャッシュに対応します。ライトバックキャッシュを有効にするには、**rbd_cache = true** を Ceph 設定ファイルの **[client]** セクションに追加します。デフォルトでは、**librbd** はキャッシュを実行しません。書き込みおよび読み取りはストレージクラスターに直接移動し、データがすべてのレプリカのディスクにある場合にのみ書き込みに戻ります。キャッシュを有効にすると、**rbd_cache_max_dirty** の非フラッシュバイト数を超えない限り、書き込みは即座に返します。この場合、書き込みによって、十分なバイト数がフラッシュされるまでライトバックおよびブロックがトリガーされます。

Ceph ブロックデバイスはライトスルーキャッシュに対応します。キャッシュのサイズを設定し、ターゲットと制限を設定して、ライトバックキャッシュから write-through キャッシュに切り替えることができます。write-through モードを有効にするには、**rbd_cache_max_dirty** を 0 に設定します。つまり、書き込みは、データがすべてのレプリカのディスクにある場合にのみ返されますが、読み取りはキャッシュから送られる可能性があります。このキャッシュはクライアントのメモリーにあり、各

Ceph ブロックデバイスイメージ自体があります。キャッシュはクライアントのローカルなので、イメージにアクセスする他の条件がある場合は、一貫性がありません。Ceph ブロックデバイスの上に他のファイルシステムまたは OCFS を実行すると、キャッシュが有効ではありません。

Ceph ブロックデバイスの Ceph 設定は、Ceph 設定ファイルの **[client]** セクションで、デフォルトでは **/etc/ceph/ceph.conf** で設定する必要があります。

設定には以下が含まれます。

rbd_cache

説明

RADOS Block Device (RBD) のキャッシュを有効にします。

型

Boolean

必須

いいえ

デフォルト

true

rbd_cache_size

説明

RBD キャッシュサイズ (バイト単位)。

型

64 ビット整数

必須

いいえ

デフォルト

32 MiB

rbd_cache_max_dirty

説明

キャッシュがライトバックをトリガーする **ダーティー** 制限 (バイト単位)。**0** の場合、ライトスルー (ライトスルー) キャッシュを使用します。

型

64 ビット整数

必須

いいえ

制約

rbd cache size より小さくなければなりません。

デフォルト

24 MiB

rbd_cache_target_dirty

説明

キャッシュがデータストレージにデータを書き込む前に **dirty target**。キャッシュへの書き込みをブロックしません。

型

64 ビット整数

必須

いいえ

制約

rbd cache max dirty 未満である必要があります。

デフォルト

16 MiB

rbd_cache_max_dirty_age

説明

ライトバックの開始前にダーティーデータがキャッシュ内にある秒数。

型

浮動小数点 (Float)

必須

いいえ

デフォルト

1.0

rbd_cache_max_dirty_object

説明

オブジェクトのダーティー制限: **rbd_cache_size** からの自動計算の場合は **0** に設定します。

型

Integer

デフォルト

0

rbd_cache_block_writes_upfront

説明

true の場合、**aio_write** 呼び出しが完了するまでキャッシュへの書き込みをブロックします。**false** の場合、**aio_completion** が呼び出される前にブロックされます。

型

Boolean

デフォルト

false

rbd_cache_writethrough_until_flush

説明

write-through モードで起動し、最初のフラッシュ要求が受信後に write-back に切り替えます。この有効化は Conservative ですが、rbd で実行している仮想マシンが、2.6.32 以前の Linux における virtio ドライバーと同様にフラッシュを送信することが古い場合は安全な設定です。

型

Boolean

必須

いいえ

デフォルト

true

A.4. ブロックデバイスの親および子読み取りのオプション

rbd_balance_snap_reads

説明

Ceph は通常、プライマリー OSD からオブジェクトを読み取ります。読み取りは不変であるため、この機能を使用すると、プライマリー OSD とレプリカとの間で snap の読み取りのバランスを取ることができます。

型

Boolean

デフォルト

false

rbd_localize_snap_reads

説明

rbd_balance_snap_reads は、スナップショットを読み取るためにレプリカをランダム化します。**rbd_localize_snap_reads** を有効にすると、ブロックデバイスは CRUSH マップを検索し、スナップショットを読み取るため最も近い OSD またはローカル OSD を検索します。

型

Boolean

デフォルト

false

rbd_balance_parent_reads

説明

Ceph は通常、プライマリー OSD からオブジェクトを読み取ります。読み取りは不変であるため、この機能を使用すると、プライマリー OSD とレプリカとの間で親読み取りのバランスを取ることができます。

型

Boolean

デフォルト

false

rbd_localize_parent_reads

説明

rbd_balance_parent_reads は親を読み取るためにレプリカをランダム化します。**rbd_localize_parent_reads** を有効にすると、ブロックデバイスは CRUSH マップを検索し、親を読み取るために最も近い OSD またはローカル OSD を検索します。

型

Boolean

デフォルト

true

A.5. ブロックデバイスの読み取りオプション

RBD は、小規模な連続読み取りを最適化するために read-ahead/prefetching をサポートします。これは通常、仮想マシンではゲスト OS で処理する必要がありますが、ブートローダーは効率的な読み取りでは機能しない場合があります。キャッシュが無効になっている場合、先読み (read-ahead) は自動的に無効になります。

rbd_readahead_trigger_requests

説明

read-ahead をトリガーするために必要な順次読み取り要求の数。

型

整数

必須

いいえ

デフォルト

10

rbd_readahead_max_bytes

説明

read-ahead リクエストの最大サイズ。ゼロの場合は、read-ahead が無効になります。

型

64 ビット整数

必須

いいえ

デフォルト

512 KiB

rbd_readahead_disable_after_bytes

説明

この多数のバイトが RBD イメージから読み取られると、閉じられるまでそのイメージの読み取りは無効にされます。これにより、ゲスト OS が起動したら、事前に読み取れることができます。ゼロの場合は、読み取り先は有効のままになります。

型

64 ビット整数

必須

いいえ

デフォルト

50 MiB

A.6. ブロックデバイスの拒否リストオプション

rbd_blocklist_on_break_lock

説明

ロックが破損したクライアントを拒否リストに追加するかどうか

型

Boolean

デフォルト

true

rbd_blocklist_expire_seconds

説明

拒否リストに追加するまでの秒数 (OSD のデフォルトの場合は 0 に設定)。

型

Integer

デフォルト

0

A.7. ブロックデバイスジャーナルオプション

rbd_journal_order

説明

ジャーナルオブジェクトの最大サイズを計算するための移動ビット数。この値は、**12** から **64** までになります。

型

32 ビット未署名の整数

デフォルト

24

rbd_journal_splay_width

説明

アクティブなジャーナルオブジェクトの数。

型

32 ビット未署名の整数

デフォルト

4

rbd_journal_commit_age

説明

コミットの間隔 (秒単位)。

型

倍精度浮動小数点数型

デフォルト

5

rbd_journal_object_flush_interval

説明

ジャーナルオブジェクトごとの保留中のコミットの最大数。

型

Integer

デフォルト

0

rbd_journal_object_flush_bytes

説明

ジャーナルオブジェクトあたりの保留中の最大バイト数。

型

Integer

デフォルト

0

rbd_journal_object_flush_age

説明

保留中のコミットの最大間隔 (秒単位)。

型

倍精度浮動小数点数型

デフォルト

0

rbd_journal_pool

説明

ジャーナルオブジェクトのプールを指定します。

型

String

デフォルト

""

A.8. ブロックデバイス設定の上書きオプション

ブロックデバイス設定オプションは、グローバルおよびプールレベルのオプションを上書きします。

グローバルレベル

利用可能なキー

rbd_qos_bps_burst

説明

希望する IO バイトのバースト制限。

型

Integer

デフォルト

0

rbd_qos_bps_limit

説明

1秒あたりの IO バイトの必要な制限。

型

Integer

デフォルト

0

rbd_qos_iops_burst

説明

IO 操作の必要なバースト制限。

型

Integer

デフォルト

0

rbd_qos_iops_limit

説明

1秒あたりの IO 操作の必要な上限。

型

Integer

デフォルト

0

rbd_qos_read_bps_burst

説明

読み取りバイトの必要なバースト制限。

型

Integer

デフォルト

0

rbd_qos_read_bps_limit

説明

1秒あたりの読み取りバイトの必要な制限。

型

Integer

デフォルト

0

rbd_qos_read_iops_burst

説明

読み取り操作の必要なバースト制限。

型

Integer

デフォルト

0

rbd_qos_read_iops_limit

説明

1秒あたりの読み取り操作の必要な制限。

型

Integer

デフォルト

0

rbd_qos_write_bps_burst

説明

書き込みバイトの必要なバースト制限。

型

Integer

デフォルト

0

rbd_qos_write_bps_limit

説明

1秒あたりの書き込みバイト数の必要な制限。

型

Integer

デフォルト

0

rbd_qos_write_iops_burst

説明

書き込み操作の必要なバースト制限。

型

Integer

デフォルト

0

rbd_qos_write_iops_limit

説明

1秒あたりの書き込み操作のバースト制限を指定します。

型

Integer

デフォルト

0

上記のキーは以下に使用できます。

rbd config global set CONFIG_ENTITY KEY VALUE

説明

グローバルレベルの設定の上書きを設定します。

rbd config global get CONFIG_ENTITY KEY

説明

グローバルレベルの設定の上書きを取得します。

rbd config global list CONFIG_ENTITY

説明

グローバルレベルの設定の上書きをリスト表示します。

rbd config global remove CONFIG_ENTITY KEY

説明

グローバルレベルの設定の上書きを削除します。

プールレベル

rbd config pool set POOL_NAME KEY VALUE

説明

プールレベルの設定の上書きを設定します。

rbd config pool get POOL_NAME KEY

説明

プールレベルの設定の上書きを取得します。

rbd 設定プールリスト POOL_NAME

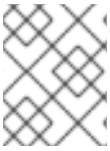
説明

プールレベルの設定のオーバーライドをリスト表示します。

rbd config pool remove POOL_NAME KEY

説明

プールレベルの設定の上書きを削除します。



注記

CONFIG_ENTITY はグローバル、クライアント ID、またはクライアント ID です。 **KEY** は設定キーです。**VALUE** は設定値です。**POOL_NAME** はプールの名前です。

A.9. ブロックデバイスの入出力オプション

Red Hat Ceph Storage の一般的な入出力オプション。

rbd_compression_hint

説明

書き込み操作時に OSD に送信するヒント。**compressible** に設定し、OSD **bluestore_compression_mode** 設定が **passive** の場合に、OSD はデータの圧縮を試行します。**incompressible** に設定されており、OSD の **bluestore_compression_mode** 設定が **aggressive** の場合には、OSD はデータの圧縮を試行しません。

型

Enum

必須

いいえ

デフォルト

none

値

none、**compressible**、**incompressible**

rbd_read_from_replica_policy

説明

読み取り操作を受け取る OSD を決定するポリシー。**default** に設定されている場合には、各 PG のプライマリー OSD は常に読み取り操作に使用されます。**balance** に設定されている場合には、読み取り操作はレプリカセット内で無作為に選択された OSD に送信されます。**localize** に設定されている場合には、読み取り操作は CRUSH マップによって決定され、**crush_location** 設定オプションで最も近い OSD に送信されます。ここで、**crush_location** は **key=value** を表記されます。**key** は CRUSH マップキーと連携します。

注記

この機能により、ストレージクラスターは、Red Hat Ceph Storage の最新バージョンと最小互換のある OSD リリースで設定する必要があります。

型

Enum

必須

いいえ

デフォルト

default

値

default、**balance**、**localize**