Red Hat Ceph Storage 4

管理ガイド

Red Hat Ceph Storage の管理

Last Updated: 2023-08-23

Red Hat Ceph Storage 4 管理ガイド

Red Hat Ceph Storage の管理

法律上の通知

概要

本書では、Red Hat Ceph Storage のプロセスの管理、クラスターの状態の監視、ユーザーの管理、およびデーモンの追加および削除方法を説明します。 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、弊社の CTO、Chris Wright のメッセージを参照してください。

目次

第1章 CEPH 管理

Red Hat Ceph Storage クラスターは、全 Ceph デプロイメントの基盤となります。Red Hat Ceph Storage クラスターをデプロイしたら、Red Hat Ceph Storage クラスターの正常な状態を維持し、最適に実行するための管理操作を実行できます。

Red Hat Ceph Storage 管理ガイドは、ストレージ管理者が以下のようなタスクを実行するのに役立ちます。

- Red Hat Ceph Storage クラスターの正常性を確認する方法
- Red Hat Ceph Storage クラスターサービスを起動および停止する方法
- 実行中の Red Hat Ceph Storage クラスターから OSD を追加または削除する方法
- Red Hat Ceph Storage クラスターに保管されたオブジェクトへのユーザー認証およびアクセス制御を管理する方法
- Red Hat Ceph Storage クラスターでオーバーライドを使用する方法
- Red Hat Ceph Storage クラスターのパフォーマンスを監視する方法

基本的な Ceph ストレージクラスターは、2 種類のデーモンで設定されます。

- Ceph Object Storage Device (OSD) は、OSD に割り当てられた配置グループ内にオブジェクトとしてデータを格納します。
- Ceph Monitor はクラスターマップのマスターコピーを維持します。

実稼働システムでは、高可用性を実現する Ceph Monitor が 3 つ以上含まれます。通常、許容可能な負荷分散、データのリバランス、およびデータ復旧に備えて最低 50 OSD が含まれます。

関連情報

- Red Hat Ceph Storage インストールガイド

第2章 CEPH のプロセス管理について

ストレージ管理者は、ベアメタルまたはコンテナー上の種別またはインスタンスで各種の Ceph デーモンを操作することができます。これらのデーモンを操作すると、必要に応じてすべての Ceph サービスを開始、停止、および再起動することができます。

2.1. 前提条件

- Red Hat Ceph Storage ソフトウェアのインストール

2.2. CEPH プロセスの管理

Red Hat Ceph Storage では、すべてのプロセス管理は Systemd サービスを介して行われます。Ceph デーモンの **start**、**restart**、および **stop** を行う場合には毎回、デーモンの種別またはデーモンインスタンスを指定する必要があります。

関連情報

- Systemd の使用に関する詳細は、Red Hat Enterprise Linux の **システム管理者のガイド** の**systemd によるサービス管理**を参照してください。

2.3. すべての CEPH デーモンの開始、停止、および再起動

ノードからの **admin** として、すべての Ceph デーモンを起動、停止、および再起動します。

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 **root** アクセスを持つ。

手順

1. すべての Ceph デーモンを起動します。

```
[root@admin ~]# systemctl start ceph.target
```

2. すべての Ceph デーモンを停止します。

```
[root@admin ~]# systemctl stop ceph.target
```

3. すべての Ceph デーモンを再起動します。

```
[root@admin ~]# systemctl restart ceph.target
```

2.4. タイプによる CEPH デーモンの開始、停止、および再起動

特定の種別のすべての Ceph デーモンを起動、停止、または再起動するには、Ceph デーモンを実行するノードで以下の手順に従います。

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 **root** アクセスを持つ。

手順

- **Ceph Monitor** ノード上で以下を行います。

  起動:

  ```
  [root@mon ~]# systemctl start ceph-mon.target
  ```

  停止:

  ```
  [root@mon ~]# systemctl stop ceph-mon.target
  ```

  再起動:

  ```
  [root@mon ~]# systemctl restart ceph-mon.target
  ```

- **Ceph Manager** ノードで以下を行います。

  起動:

  ```
  [root@mgr ~]# systemctl start ceph-mgr.target
  ```

  停止:

  ```
  [root@mgr ~]# systemctl stop ceph-mgr.target
  ```

  再起動:

  ```
  [root@mgr ~]# systemctl restart ceph-mgr.target
  ```

- **Ceph OSD** ノード上で以下を行います。

  起動:

  ```
  [root@osd ~]# systemctl start ceph-osd.target
  ```

  停止:

  ```
  [root@osd ~]# systemctl stop ceph-osd.target
  ```

  再起動:

  ```
  [root@osd ~]# systemctl restart ceph-osd.target
  ```

- **Ceph Object Gateway** ノードの場合:

起動:

```
[root@rgw ~]# systemctl start ceph-radosgw.target
```

停止:

```
[root@rgw ~]# systemctl stop ceph-radosgw.target
```

再起動:

```
[root@rgw ~]# systemctl restart ceph-radosgw.target
```

2.5. インスタンスごとの CEPH デーモンの開始、停止、および再起動

インスタンスごとに Ceph デーモンを起動、停止、または再起動するには、Ceph デーモンを実行するノードで以下の手順に従います。

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 **root** アクセスを持つ。

手順

- **Ceph Monitor** ノード上で以下を行います。

  起動:

  ```
  [root@mon ~]# systemctl start ceph-mon@MONITOR_HOST_NAME
  ```

  停止:

  ```
  [root@mon ~]# systemctl stop ceph-mon@MONITOR_HOST_NAME
  ```

  再起動:

  ```
  [root@mon ~]# systemctl restart ceph-mon@MONITOR_HOST_NAME
  ```

  置き換え

  - **MONITOR_HOST_NAME** を、Ceph Monitor ノードの名前に置き換えます。

- **Ceph Manager** ノードで以下を行います。

  起動:

  ```
  [root@mgr ~]# systemctl start ceph-mgr@MANAGER_HOST_NAME
  ```

  停止:

```
[root@mgr ~]# systemctl stop ceph-mgr@MANAGER_HOST_NAME
```

再起動:

```
[root@mgr ~]# systemctl restart ceph-mgr@MANAGER_HOST_NAME
```

置き換え

- **MANAGER_HOST_NAME** は、Ceph Manager ノードの名前に置き換えます。

- Ceph OSD ノード:

  起動:

  ```
  [root@osd ~]# systemctl start ceph-osd@OSD_NUMBER
  ```

  停止:

  ```
  [root@osd ~]# systemctl stop ceph-osd@OSD_NUMBER
  ```

  再起動:

  ```
  [root@osd ~]# systemctl restart ceph-osd@OSD_NUMBER
  ```

  置き換え

  - **OSD_NUMBER** を、Ceph OSD の **ID** 番号に置き換えます。
    たとえば、**ceph osd ツリー** コマンドの出力を確認すると、**osd.0** の **ID** は **0** になります。

- Ceph Object Gateway ノードで以下を行います。

  起動:

  ```
  [root@rgw ~]# systemctl start ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  停止:

  ```
  [root@rgw ~]# systemctl stop ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  再起動:

  ```
  [root@rgw ~]# systemctl restart ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  置き換え

  - **OBJ_GATEWAY_HOST_NAME** は、Ceph Object Gateway ノードの名前に置き換えます。

2.6. コンテナー内で実行される CEPH デーモンの開始、停止、および再起動

systemctl コマンドを使用して、コンテナーで実行する Ceph デーモンの起動、停止、再起動を行います。

前提条件

- Red Hat Ceph Storage ソフトウェアのインストール
- ノードへのルートレベルのアクセス。

手順

1. コンテナーで実行している Ceph デーモンを起動、停止、または再起動するには、以下の形式で設定されるように、**root** で **systemctl** コマンドを実行します。

```
systemctl ACTION ceph-DAEMON@ID
```

置き換え

- ACTION は、実行するアクション (**start**、**stop**、または **restart**) です。
- DAEMON はデーモン (**osd**、**mon**、**mds**、または **rgw**) です。
- ID は以下のいずれかになります。
  - **ceph-mon** デーモン、**ceph-mds** デーモン、または **ceph-rgw** デーモンが実行している短いホスト名。
  - **ceph-osd** デーモンの ID (デプロイされている場合)。

たとえば、**osd01** で **ceph-osd** デーモンを再起動するには、以下のコマンドを実行します。

```
[root@osd ~]# systemctl restart ceph-osd@osd01
```

ceph-monitor01 ホストで実行する **ceph-mon** デーモンを起動するには、以下のコマンドを実行します。

```
[root@mon ~]# systemctl start ceph-mon@ceph-monitor01
```

ceph-rgw01 ホストで実行する **ceph-rgw** デーモンを停止するには、以下のコマンドを実行します。

```
[root@rgw ~]# systemctl stop ceph-radosgw@ceph-rgw01
```

2. アクションが正常に完了したことを確認します。

```
systemctl status ceph-DAEMON@ID
```

以下に例を示します。

```
[root@mon ~]# systemctl status ceph-mon@ceph-monitor01
```

関連情報

- 詳細は、**Red Hat Ceph Storage 管理ガイド**の Ceph のプロセス管理について の章を参照してください。

2.7. コンテナーで実行される CEPH デーモンのログを表示する

コンテナーホストから **journald** デーモンを使用して、コンテナーから Ceph デーモンのログを表示します。

前提条件

- Red Hat Ceph Storage ソフトウェアのインストール
- ノードへのルートレベルのアクセス。

手順

1. Ceph ログ全体を表示するには、**root** として次の形式で設定される **journalctl** コマンドを実行します。

```
journalctl -u ceph-DAEMON@ID
```

置き換え

- DAEMON は Ceph デーモン (**osd**、**mon**、または **rgw**) です。
- ID は以下のいずれかになります。
  - **ceph-mon** デーモン、**ceph-mds** デーモン、または **ceph-rgw** デーモンが実行している短いホスト名。
  - **ceph-osd** デーモンの ID (デプロイされている場合)。

たとえば、ID **osd01** の **ceph-osd** デーモンのログ全体を表示するには、以下のコマンドを実行します。

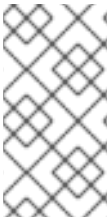
```
[root@osd ~]# journalctl -u ceph-osd@osd01
```

2. 最近のジャーナルエントリーのみを表示するには、**-f** オプションを使用します。

```
journalctl -fu ceph-DAEMON@ID
```

たとえば、**ceph-monitor01** ホストで実行する **ceph-mon** デーモンの最近のジャーナルエントリーのみを表示するには、以下のコマンドを実行します。

```
[root@mon ~]# journalctl -fu ceph-mon@ceph-monitor01
```



注記

sosreport ユーティリティーを使用して **journald** ログを表示することもできます。SOS レポートの詳細については、RedHat カスタマーポータルのソリューション sosreport とは何ですか ?Red Hat Enterprise Linux で sosreport を作成する方法は ? を参照してください。

関連情報

- man ページの **journalctl(1)**

2.8. コンテナー化された CEPH デーモンのファイルへのロギングを有効にする

デフォルトでは、コンテナー化された Ceph デーモンはファイルにログを記録しません。集中設定管理を使用して、コンテナー化された Ceph デーモンがファイルにログを記録できるようにすることができます。

前提条件

- Red Hat Ceph Storage ソフトウェアのインストール
- コンテナー化されたデーモンが実行されるノードへのルートレベルのアクセス。

手順

1. **var/log/ceph** ディレクトリーに移動します。

   例

   ```
   [root@host01 ~]# cd /var/log/ceph
   ```

2. 既存のログファイルをメモします。

   構文

   ```
   ls -l /var/log/ceph/
   ```

   例

   ```
   [root@host01 ceph]# ls -l /var/log/ceph/
   total 396
   -rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.0.log
   -rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.3.log
   -rw-r--r--. 1 root root 181641 Feb  5 14:42 ceph-volume.log
   ```

   この例では、OSD.0 および OSD.3 のファイルへのロギングがすでに有効になっています。

3. ロギングを有効にするデーモンのコンテナー名を取得します。

   Red Hat Enterprise Linux 7

   ```
   [root@host01 ceph]# docker ps -a
   ```

   Red Hat Enterprise Linux 8

   ```
   [root@host01 ceph]# podman ps -a
   ```

4. 集中設定管理を使用して、Ceph デーモンのファイルへのロギングを有効にします。

Red Hat Enterprise Linux 7

```
docker exec CONTAINER_NAME ceph config set DAEMON_NAME log_to_file true
```

Red Hat Enterprise Linux 8

```
podman exec CONTAINER_NAME ceph config set DAEMON_NAME log_to_file true
```

DAEMON_NAME は CONTAINER_NAME から派生しています。**ceph-** を削除し、デーモンとデーモン ID の間のハイフンをピリオドに置き換えます。

Red Hat Enterprise Linux 7

```
[root@host01 ceph]# docker exec ceph-mon-host01 ceph config set mon.host01 log_to_file true
```

Red Hat Enterprise Linux 8

```
[root@host01 ceph]# podman exec ceph-mon-host01 ceph config set mon.host01 log_to_file true
```

5. オプション: クラスターログのファイルへのロギングを有効にするには、**mon_cluster_log_to_file** オプションを使用します。

Red Hat Enterprise Linux 7

```
docker exec CONTAINER_NAME ceph config set DAEMON_NAME mon_cluster_log_to_file true
```

Red Hat Enterprise Linux 8

```
podman exec CONTAINER_NAME ceph config set DAEMON_NAME mon_cluster_log_to_file true
```

Red Hat Enterprise Linux 7

```
[root@host01 ceph]# docker exec ceph-mon-host01 ceph config set mon.host01 mon_cluster_log_to_file true
```

Red Hat Enterprise Linux 8

```
[root@host01 ceph]# podman exec ceph-mon-host01 ceph config set mon.host01 mon_cluster_log_to_file true
```

6. 更新された設定を検証します。

Red Hat Enterprise Linux 7

```
docker exec CONTAINER_NAME ceph config show-with-defaults DAEMON_NAME | grep log_to_file
```

Red Hat Enterprise Linux 8

```
podman exec CONTAINER_NAME ceph config show-with-defaults DAEMON_NAME | grep log_to_file
```

例

```
[root@host01 ceph]# podman exec ceph-mon-host01 ceph config show-with-defaults mon.host01 | grep log_to_file
log_to_file                                    true
mon      default[false]
mon_cluster_log_to_file                        true
mon      default[false]
```

7. オプション: Ceph デーモンを再始動します。

構文

```
systemctl restart ceph-DAEMON@DAEMON_ID
```

例

```
[root@host01 ceph]# systemctl restart ceph-mon@host01
```

8. 新しいログファイルが存在することを確認します。

構文

```
ls -l /var/log/ceph/
```

例

```
[root@host01 ceph]# ls -l /var/log/ceph/
total 408
-rw-------. 1 ceph ceph    202 Feb  5 16:06 ceph.audit.log
-rw-------. 1 ceph ceph   3182 Feb  5 16:06 ceph.log
-rw-r--r--. 1 ceph ceph   2049 Feb  5 16:06 ceph-mon.host01.log
-rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.0.log
-rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.3.log
-rw-r--r--. 1 root root 181641 Feb  5 14:42 ceph-volume.log
```

Monitor デーモン用の **ceph -mon.host01.log** とクラスターログ用の ceph.log の 2 つの新しいファイルが作成されました。

関連情報

- 詳細については、**Red Hat Ceph Storage トラブルシューティングガイド** の **ログの設定** セクションを参照してください。

2.9. CEPH デーモンのログファイルの収集

Ceph デーモンのログファイルを収集するには、**gather-ceph-logs.yml** Ansible Playbook を実行します。現在、Red Hat Ceph Storage は、コンテナー化されていないデプロイメントのみのログ収集をサポートしています。

前提条件

- Red Hat Ceph Storage クラスターが実行されている。
- Ansible ノードへの管理者レベルのアクセス。

手順

1. **/usr/share/ceph-ansible** ディレクトリーに移動します。

```
[ansible@admin ~]# cd /usr/share/ceph-ansible
```

2. Playbook を実行します。

```
[ansible@admin ~]# ansible-playbook infrastructure-playbooks/gather-ceph-logs.yml -i hosts
```

3. Ansible 管理ノードでログが収集されるまで待ちます。

関連情報

- 詳細については、**Red Hat Ceph Storage 管理ガイド** の **コンテナーで実行される Ceph デーモンのログファイルの表示** セクションを参照してください。

2.10. RED HAT CEPH STORAGE クラスターの電源をオフにして再起動

Ceph クラスターの電源をオフにしてリブートするには、以下の手順を実施します。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- **root** アクセスを持つ。

手順

Red Hat Ceph Storage クラスターの電源オフ

1. クライアントがこのクラスターおよび他のクライアントで RBD イメージおよび RADOS Gateway を使用しないようにします。

2. 次のステップに進む前に、クラスターの状態が正常な状態 (**Health_OK** およびすべての PG が **active+clean**) である必要があります。Ceph Monitor や OpenStack コントローラー **ceph status** ノードなどのクライアントキーリングを持つノードで実行し、クラスターが正常であることを確認します。

3. Ceph File System (**CephFS**) を使用する場合は、**CephFS** クラスターを停止する必要があります。**CephFS** クラスターをダウンさせるには、ランク数を **1** に減らし、**cluster_down** フラグを設定して最後のランクを失敗させることで行います。

   以下に例を示します。

```
[root@osd ~]# ceph fs set FS_NAME max_mds 1
[root@osd ~]# ceph mds deactivate FS_NAME:1 # rank 2 of 2
[root@osd ~]# ceph status # wait for rank 1 to finish stopping
[root@osd ~]# ceph fs set FS_NAME cluster_down true
[root@osd ~]# ceph mds fail FS_NAME:0
```

cluster_down フラグを設定することで、スタンバイが失敗したランクを引き継ぐことを防ぎます。

4. **noout** フラグ、**norecover** フラグ、**norebalance** フラグ、**nobackfill** フラグ、**nodown** フラグ、および **pause** フラグを設定します。クライアントキーリングが設定されたノードで以下のコマンドを実行します。Ceph Monitor または OpenStack コントローラーノードの例を以下に示します。

```
[root@mon ~]# ceph osd set noout
[root@mon ~]# ceph osd set norecover
[root@mon ~]# ceph osd set norebalance
[root@mon ~]# ceph osd set nobackfill
[root@mon ~]# ceph osd set nodown
[root@mon ~]# ceph osd set pause
```

5. OSD ノードを 1 つずつシャットダウンします。

```
[root@osd ~]# systemctl stop ceph-osd.target
```

6. 監視ノードを 1 つずつシャットダウンします。

```
[root@mon ~]# systemctl stop ceph-mon.target
```

Red Hat Ceph Storage クラスターのリブート

1. 管理ノードの電源を入れます。
2. モニターノードの電源をオンにします。

```
[root@mon ~]# systemctl start ceph-mon.target
```

3. OSD ノードの電源をオンにします。

```
[root@osd ~]# systemctl start ceph-osd.target
```

4. すべてのノードが起動するのを待ちます。すべてのサービスが稼働中であり、ノード間の接続に問題がないことを確認します。
5. **noout** フラグ、**norecover** フラグ、**norebalance** フラグ、**nobackfill** フラグ、**nodown** フラグ、および **pause** フラグの設定を解除します。クライアントキーリングが設定されたノードで以下のコマンドを実行します。Ceph Monitor または OpenStack コントローラーノードの例を以下に示します。

```
[root@mon ~]# ceph osd unset noout
[root@mon ~]# ceph osd unset norecover
[root@mon ~]# ceph osd unset norebalance
```

```
[root@mon ~]# ceph osd unset nobackfill
[root@mon ~]# ceph osd unset nodown
[root@mon ~]# ceph osd unset pause
```

6. Ceph File System (**CephFS**) を使用する場合は、**cluster_down** フラグを **false** に設定して **CephFS** クラスターをバックアップする必要があります。

```
[root@admin~]# ceph fs set FS_NAME cluster_down false
```

7. クラスターの状態が正常であることを確認します (**Health_OK**、およびすべての PG が **active+clean**)。クライアントキーリングが設定されたノードで **ceph status** を実行します。たとえば、クラスターが正常であることを確認する Ceph Monitor または OpenStack コントローラーノードなど。

2.11. 関連情報

- Ceph のインストールに関する詳しい情報は、Red Hat Ceph Storage インストールガイドを参照してください。

第3章 CEPH STORAGE クラスターのモニタリング

ストレージ管理者は、Ceph の個々のコンポーネントの正常性を監視すると共に、Red Hat Ceph Storage クラスターの全体的な健全性を監視することができます。

Red Hat Ceph Storage クラスターを稼働したら、ストレージクラスターの監視を開始して、Ceph Monitor デーモンおよび Ceph OSD デーモンが高レベルで実行されていることを確認することができます。Ceph Storage クラスタークライアントは Ceph Monitor に接続して、最新バージョンのストレージクラスターマップを受け取ってから、ストレージクラスター内の Ceph プールへのデータの読み取りおよび書き込みを実施することができます。そのため、モニタークラスターには、Ceph クライアントがデータの読み取りおよび書き込みが可能になる前に、クラスターの状態に関する合意が必要です。

Ceph OSD は、セカンダリー OSD の配置グループのコピーと、プライマリー OSD 上の配置グループをピアにする必要があります。障害が発生した場合、ピアリングは **active + clean** 状態以外のものを反映します。

3.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

3.2. CEPH STORAGE クラスターのハイレベル監視

ストレージ管理者は、Ceph デーモンの正常性を監視し、それらが稼働していることを確認します。また、高レベルのモニタリングには、ストレージクラスター容量を確認して、ストレージクラスターが **完全な比率** を超えないようにします。Red Hat Ceph Storage ダッシュボード は、高レベルのモニタリングを実行する最も一般的な方法です。ただし、コマンドラインインターフェイス、Ceph 管理ソケットまたは Ceph API を使用してストレージクラスターを監視することもできます。

3.2.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

3.2.2. Ceph コマンドラインインターフェイスの対話形式の使用

ceph コマンドラインユーティリティーを使用して、Ceph ストレージクラスターと対話的にインターフェイスで接続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。
- ノードへのルートレベルのアクセス。

手順

1. インタラクティブモードで **ceph** ユーティリティーを実行するには、以下を行います。
   a. **ベアメタル** デプロイメント:

   例

```
[root@mon ~]# ceph
ceph> health
ceph> status
```

```
ceph> quorum_status
ceph> mon_status
```

b. **コンテナー** デプロイメント:

Red Hat Enterprise Linux 7

```
docker exec -it ceph-mon-MONITOR_NAME /bin/bash
```

Red Hat Enterprise Linux 8

```
podman exec -it ceph-mon-MONITOR_NAME /bin/bash
```

置き換え

- MONITOR_NAME は、Ceph Monitor コンテナーの名前に置き換えます。 **docker ps** コマンドまたは **podman ps** コマンドを実行します。

  例

  ```
  [root@container-host ~]# podman exec -it ceph-mon-mon01 /bin/bash
  ```

  この例では、**mon01** で対話的なターミナルセッションを開き、Ceph インタラクティブシェルを起動することができます。

3.2.3. ストレージクラスターの正常性の確認

Ceph Storage クラスターを起動してからデータの読み取りまたは書き込みを開始する前に、ストレージクラスターの正常性を確認します。

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. Ceph Storage クラスターの正常性を確認するには、以下を使用します。

   ```
   [root@mon ~]# ceph health
   ```

2. 設定またはキーリングにデフォルト以外の場所を指定した場合は、その場所を指定できます。

   ```
   [root@mon ~]# ceph -c /path/to/conf -k /path/to/keyring health
   ```

Ceph クラスターの起動時に、**HEALTH_WARN XXX num placement groups stale** などの正常性警告が生じる可能性があります。しばらく待ってから再度確認します。ストレージクラスターの準備が整ったら、**ceph health** は **HEALTH_OK** などのメッセージを返すはずです。この時点で、クラスターの使用を開始するのは問題ありません。

3.2.4. ストレージクラスターイベントの監視

コマンドラインインターフェイスを使用して、Ceph Storage クラスターで発生しているイベントを監視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。
- ノードへのルートレベルのアクセス。

手順

1. コマンドラインでクラスターの進行中のイベントを確認するには、新しい端末を開き、以下を入力します。

```
[root@mon ~]# ceph -w
```

Ceph は各イベントを出力します。たとえば、1 台のモニターで設定され、2 つの OSD で設定される小さな Ceph クラスターが以下のように出力されます。

```
cluster b370a29d-9287-4ca3-ab57-3d824f65e339
 health HEALTH_OK
 monmap e1: 1 mons at {ceph1=10.0.0.8:6789/0}, election epoch 2, quorum 0 ceph1
 osdmap e63: 2 osds: 2 up, 2 in
  pgmap v41338: 952 pgs, 20 pools, 17130 MB data, 2199 objects
        115 GB used, 167 GB / 297 GB avail
             952 active+clean

2014-06-02 15:45:21.655871 osd.0 [INF] 17.71 deep-scrub ok
2014-06-02 15:45:47.880608 osd.1 [INF] 1.0 scrub ok
2014-06-02 15:45:48.865375 osd.1 [INF] 1.3 scrub ok
2014-06-02 15:45:50.866479 osd.1 [INF] 1.4 scrub ok
2014-06-02 15:45:01.345821 mon.0 [INF] pgmap v41339: 952 pgs: 952 active+clean; 17130
MB data, 115 GB used, 167 GB / 297 GB avail
2014-06-02 15:45:05.718640 mon.0 [INF] pgmap v41340: 952 pgs: 1
active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
297 GB avail
2014-06-02 15:45:53.997726 osd.1 [INF] 1.5 scrub ok
2014-06-02 15:45:06.734270 mon.0 [INF] pgmap v41341: 952 pgs: 1
active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
297 GB avail
2014-06-02 15:45:15.722456 mon.0 [INF] pgmap v41342: 952 pgs: 952 active+clean; 17130
MB data, 115 GB used, 167 GB / 297 GB avail
2014-06-02 15:46:06.836430 osd.0 [INF] 17.75 deep-scrub ok
2014-06-02 15:45:55.720929 mon.0 [INF] pgmap v41343: 952 pgs: 1
active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
297 GB avail
```

出力には以下が含まれます。

- クラスター ID
- クラスターの正常性ステータス

- モニターマップエポックおよびモニタークォーラムのステータス
- OSD マップエポックおよび OSD のステータス
- 配置グループマップバージョン
- 配置グループとプールの数
- 保存されるデータの **想定** 量および保存されるオブジェクト数
- 保存されるデータの合計量

3.2.5. Ceph のデータ使用量の計算方法

使用される 値は、使用される生のストレージの **実際** の量を反映します。**xxx GB / xxx GB** の値は、クラスターの全体的なストレージ容量のうち、2 つの数字の小さい方の利用可能な量を意味します。概念番号は、複製、クローン、またはスナップショットを作成する前に、保存したデータのサイズを反映します。したがって、Ceph はデータのレプリカを作成し、クローン作成やスナップショットのためにストレージ容量を使用することもあるため、実際に保存されるデータの量は、通常、保存された想定される量を上回ります。

3.2.6. ストレージクラスターの使用統計について

クラスターのデータ使用状況とプール間でのデータ分散を確認するには、**df** オプションを使用します。これは Linux **df** コマンドに似ています。**ceph df** コマンドまたは **ceph df detail** コマンドのいずれかを実行できます。

例

```
[root@mon ~]# ceph df
RAW STORAGE:
    CLASS     SIZE       AVAIL      USED        RAW USED     %RAW USED
    hdd       90 GiB     84 GiB     100 MiB      6.1 GiB         6.78
    TOTAL     90 GiB     84 GiB     100 MiB      6.1 GiB         6.78

POOLS:
    POOL                    ID     STORED      OBJECTS     USED       %USED     MAX AVAIL
    .rgw.root               1     1.3 KiB           4     768 KiB         0        26 GiB
    default.rgw.control     2         0 B           8         0 B         0        26 GiB
    default.rgw.meta        3     2.5 KiB          12     2.1 MiB         0        26 GiB
    default.rgw.log         4     3.5 KiB         208     6.2 MiB         0        26 GiB
    default.rgw.buckets.index    5     2.4 KiB          33     2.4 KiB         0        26 GiB
    default.rgw.buckets.data     6     9.6 KiB          15     1.7 MiB         0        26 GiB
    testpool                10       231 B           5     384 KiB         0        40 GiB
```

ceph df detail コマンドは、quota objects、quota bytes、used compression、および under compression など、その他のプール統計に関する詳細情報を提供します。

例

```
[root@mon ~]# ceph df detail
RAW STORAGE:
    CLASS     SIZE       AVAIL      USED        RAW USED     %RAW USED
    hdd       90 GiB     84 GiB     100 MiB      6.1 GiB         6.78
    TOTAL     90 GiB     84 GiB     100 MiB      6.1 GiB         6.78
```

```
POOLS:
   POOL                    ID     STORED      OBJECTS     USED        %USED     MAX AVAIL
QUOTA OBJECTS     QUOTA BYTES     DIRTY     USED COMPR     UNDER COMPR
   .rgw.root                1     1.3 KiB         4     768 KiB         0       26 GiB     N/A               N/A
4            0 B            0 B
   default.rgw.control      2         0 B         8         0 B         0       26 GiB     N/A               N/A
8            0 B            0 B
   default.rgw.meta         3     2.5 KiB        12     2.1 MiB         0       26 GiB     N/A               N/A
12           0 B            0 B
   default.rgw.log          4     3.5 KiB       208     6.2 MiB         0       26 GiB     N/A               N/A
208          0 B            0 B
   default.rgw.buckets.index      5     2.4 KiB        33     2.4 KiB         0       26 GiB     N/A               N/A
33           0 B            0 B
   default.rgw.buckets.data       6     9.6 KiB        15     1.7 MiB         0       26 GiB     N/A               N/A
15           0 B            0 B
   testpool                10       231 B         5     384 KiB         0       40 GiB     N/A               N/A
5            0 B            0 B
```

出力の **RAW STORAGE** セクションは、ストレージクラスターがデータに使用するストレージ容量の概要を説明します。

- **CLASS:** 使用されるデバイスのタイプ。
- **SIZE:** ストレージクラスターが管理する全ストレージ容量。
  上記の例では、**SIZE** が 90 GiB の場合、レプリケーション係数 (デフォルトでは 3) を考慮しない合計サイズです。レプリケーション係数を考慮した使用可能な合計容量は 90 GiB/3 = 30 GiB です。フル比率 (デフォルトでは 85%) に基づくと、使用可能な最大容量は 30 GiB * 0.85 = 25.5 GiB です。
- **AVAIL:** ストレージクラスターで利用可能な空き容量。
  上記の例では、**SIZE** が 90 GiB、 **USED** 容量が 6 GiB の場合、 **AVAIL** 容量は 84 GiB になります。レプリケーション係数 (デフォルトでは 3) を考慮した使用可能な合計容量は、84 GiB/3 = 28 GiB です。
- **USD:** ユーザーデータ、内部オーバーヘッド、または予約済み容量に消費される、ストレージクラスター内の使用済み領域の量。
  上記の例では、100 MiB がレプリケーション係数を考慮した後で利用可能な合計領域です。実際に使用可能なサイズは 33 MiB です。
- **RAW USED:** **USED** 領域の合計と、BlueStore パーティション **db** および **wal** に割り当てられた領域の合計。
- **% RAW USED:** **RAW USED** の割合。この数字は、**full ratio** と **near full ratio** で使用して、ストレージクラスターの容量に達しないようにします。

出力の **POOLS** セクションは、プールの一覧と、各プールの概念的な使用目的を提供します。このセクションの出力には、レプリカ、クローン、またはスナップショットを **反映しません**。たとえば、1 MB のデータでオブジェクトを保存する場合、概念的な使用量は 1 MB になりますが、実際の使用量は、**size = 3** のクローンやスナップショットなどのレプリカ数によっては 3 MB 以上になる場合があります。

- **POOL:** プールの名前。
- **ID:** プール ID。

- **STOERD:** ユーザーがプールに格納する実際のデータ量。
- **OBJECTS:** プールごとに保存されるオブジェクトの想定数。
- **USED:** メガバイトの場合は **M**、ギガバイトの場合は **G** を付加しない限り、キロバイト単位で保存されたデータの想定量。これは、**STORED** サイズ * レプリケーション係数です。
- **%USED:** プールごとに使用されるストレージの概念パーセンテージ。
- **MAX AVAIL:** このプールに書き込むことができる想定データ量の推定。これは、最初の OSD がフルになる前に使用できるデータの量です。CRUSH マップからディスクにまたがるデータの予測分布を考慮し、フルにする最初の OSD をターゲットとして使用します。
  上記の例では、レプリケーション係数 (デフォルトでは 3) を考慮しない **MAX AVAIL** は 153.85 です。

  MAX AVAIL の値を計算するには、ナレッジベースのアーティクル ceph df MAX AVAIL is incorrect for simple replicated pool を参照してください。
- **QUOTA OBJECTS:** クォータオブジェクトの数。
- **QUOTA BYTES:** クォータオブジェクトのバイト数。
- **USED COMPR:** 圧縮データに割り当てられる領域の量これには、圧縮データ、割り当て、レプリケーション、およびイレイジャーコーディングのオーバーヘッドが含まれます。
- **UNDER COMPR:** 圧縮に渡されるデータの量で、圧縮された形式で保存することが十分に有益です。

注記

POOLS セクションの数字は概念的です。レプリカ、スナップショット、またはクローンの数は含まれていません。その結果、**USED** と **%USED** の量の合計は、出力の **GLOBAL** セクションの **RAW USED** と **%RAW USED** の量に加算されません。

注記

MAX AVAIL の値は、使用されるレプリケーションまたはイレイジャージャーコード、ストレージをデバイスにマッピングする CRUSH ルール、それらのデバイスの使用率、および設定された **mon_osd_full_ratio** の複雑な関数です。

関連情報

- 詳細は、Ceph のデータ使用量の計算方法 を参照してください。
- 詳細は、 OSD の使用状況の統計について を参照してください。

3.2.7. OSD の使用状況の統計について

ceph osd df コマンドを使用して、OSD 使用率の統計を表示します。

```
[root@mon]# ceph osd df
ID CLASS WEIGHT  REWEIGHT SIZE    USE     DATA    OMAP    META    AVAIL   %USE VAR
PGS
 3   hdd 0.90959  1.00000  931GiB 70.1GiB 69.1GiB      0B    1GiB  861GiB 7.53 2.93  66
```

```
 4   hdd 0.90959  1.00000  931GiB 1.30GiB  308MiB      0B    1GiB  930GiB 0.14 0.05  59
 0   hdd 0.90959  1.00000  931GiB 18.1GiB 17.1GiB      0B    1GiB  913GiB 1.94 0.76  57
MIN/MAX VAR: 0.02/2.98  STDDEV: 2.91
```

- **ID:** OSD の名前。
- **CLASS:** OSD が使用するデバイスのタイプ。
- **WEIGHT:** CRUSH マップの OSD の重み。
- **REWEIGHT:** デフォルトの再重み値です。
- **SIZE:** OSD の全体的なストレージ容量
- **USE:** OSD の容量
- **DATA:** ユーザーデータが使用する OSD 容量
- **OMAP:** オブジェクトマップ (**omap**) データを保存するために使用されている **bluefs** ストレージの推定値 (**rocksdb** に保存されたキー/値のペア)。
- **META:** 内部メタデータに割り当てられた **bluefs** の領域、または **bluestore_bluefs_min** パラメーターで設定された値のうちいずれか大きい方の値で、**bluefs** に割り当てられた領域の合計から推定 **omap** データサイズを差し引いた値として計算されます。
- **AVAIL:** OSD で利用可能な空き容量
- **%USE:** OSD で使用されるストレージの表記率
- **VAR:** 平均の使用率を上回るまたは下回る変動。
- **PGS:** OSD 内の配置グループ数
- **MIN/MAX VAR:** すべての OSD における変更の最小値および最大値。

関連情報

- 詳細 h、**Ceph のデータ使用量の計算方法** を参照してください。
- 詳細は、**OSD の使用状況の統計について** を参照してください。
- 詳細は、**Red Hat Ceph Storage ストレージ戦略ガイド**の **CRUSH の重み** を参照してください。

3.2.8. Red Hat Ceph Storage クラスターのステータスの確認

コマンドラインインターフェイスから Red Hat Ceph Storage クラスターのステータスを確認することができます。**status** サブコマンドまたは **-s** 引数は、ストレージクラスターの現在のステータスを表示します。

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

手順

1. ストレージクラスターのステータスを確認するには、以下を実行します。

```
[root@mon ~]# ceph status
```

または、以下を実行します。

```
[root@mon ~]# ceph -s
```

2. インタラクティブモードで **status** と入力して、**Enter** を押します。

```
[root@mon ~]# ceph> status
```

たとえば、1 台のモニターで設定される小さな Ceph クラスターと、2 つの OSD が以下のように出力されます。

```
cluster b370a29d-9287-4ca3-ab57-3d824f65e339
 health HEALTH_OK
 monmap e1: 1 mons at {ceph1=10.0.0.8:6789/0}, election epoch 2, quorum 0 ceph1
 osdmap e63: 2 osds: 2 up, 2 in
  pgmap v41332: 952 pgs, 20 pools, 17130 MB data, 2199 objects
        115 GB used, 167 GB / 297 GB avail
               1 active+clean+scrubbing+deep
             951 active+clean
```

3.2.9. Ceph Monitor ステータスの確認

ストレージクラスターに複数の Ceph Monitor がある場合 (実稼働環境用の Red Hat Ceph Storage クラスターに必要)、ストレージクラスターの起動後に Ceph Monitor クォーラム (定足数) のステータスを確認し、データの読み取りまたは書き込みを実施する前に、Ceph Monitor クォーラムのステータスを確認します。

複数のモニターを実行している場合はクォーラムが存在する必要があります。

Ceph Monitor ステータスを定期的にチェックし、実行していることを確認します。Ceph Monitor に問題があり、ストレージクラスターの状態に関する合意ができない場合は、その障害により Ceph クライアントがデータを読み書きできなくなる可能性があります。

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. 監視マップを表示するには、以下を実行します。

```
[root@mon ~]# ceph mon stat
```

または

```
[root@mon ~]# ceph mon dump
```

2. ストレージクラスターのクォーラムステータスを確認するには、以下を実行します。

```
[root@mon ~]# ceph quorum_status -f json-pretty
```

Ceph はクォーラムのステータスを返します。3 つのモニターで設定される Red Hat Ceph Storage クラスターは、以下を返す場合があります。

例

```
{ "election_epoch": 10,
  "quorum": [
        0,
        1,
        2],
  "monmap": { "epoch": 1,
      "fsid": "444b489c-4f16-4b75-83f0-cb8097468898",
      "modified": "2011-12-12 13:28:27.505520",
      "created": "2011-12-12 13:28:27.505520",
      "mons": [
            { "rank": 0,
              "name": "a",
              "addr": "127.0.0.1:6789\/0"},
            { "rank": 1,
              "name": "b",
              "addr": "127.0.0.1:6790\/0"},
            { "rank": 2,
              "name": "c",
              "addr": "127.0.0.1:6791\/0"}
           ]
    }
}
```

3.2.10. Ceph 管理ソケットの使用

管理ソケットを使用して、UNIX ソケットファイルを使用して、指定したデーモンと直接対話します。たとえば、ソケットを使用すると以下を行うことができます。

- ランタイム時に Ceph 設定を一覧表示します。
- Monitor に依存せずに直接ランタイムに設定値を設定します。これは、モニターが **ダウン** している場合に便利です。
- ダンプの履歴操作
- 操作優先度キューの状態をダンプします。
- 再起動しないダンプ操作
- パフォーマンスカウンターのダンプ

さらに、Monitor または OSD に関連する問題のトラブルシューティングを行う場合は、ソケットの使用に役立ちます。

重要

管理ソケットは、デーモンの実行中にのみ利用できます。デーモンを正常にシャットダウンすると、管理ソケットが削除されます。ただし、デーモンが突然終了すると、管理ソケットが永続化される可能性があります。

デーモンが実行されていない場合でも、管理ソケットの使用を試みる際に以下のエラーが返されます。

```
Error 111: Connection Refused
```

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. ソケットを使用するには、以下を実行します。

   構文

   ```
   [root@mon ~]# ceph daemon TYPE.ID COMMAND
   ```

   以下を置き換えます。

   - **TYPE** を、Ceph デーモンのタイプ (**mon**、**osd**、**mds**) に置き換えます。
   - **ID** を、デーモン ID に置き換えます。
   - **COMMAND** を、実行するコマンドに置き換えます。指定のデーモンで利用可能なコマンドを一覧表示するには、**help** を使用します。

     例

     mon.0 という名前の Ceph Monitor の Monitor ステータスを表示するには、以下を実行します。

     ```
     [root@mon ~]# ceph daemon mon.0 mon_status
     ```

2. または、ソケットファイルを使用して Ceph デーモンを指定します。

   ```
   ceph daemon /var/run/ceph/SOCKET_FILE COMMAND
   ```

3. **osd.2** という名前の Ceph OSD のステータスを表示するには、以下のコマンドを実行します。

   ```
   [root@mon ~]# ceph daemon /var/run/ceph/ceph-osd.2.asok status
   ```

4. Ceph プロセスのソケットファイルの一覧を表示するには、以下のコマンドを実行します。

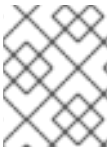
   ```
   [root@mon ~]# ls /var/run/ceph
   ```

関連情報

- 詳細は、Red Hat Ceph Storage トラブルシューティングガイドを参照してください。

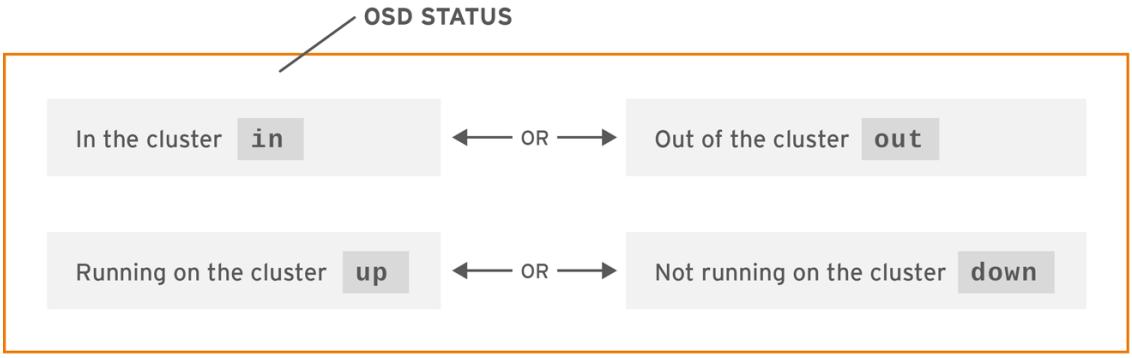
3.2.11. Ceph OSD のステータスについて

OSD のステータスは、クラスター内 (**in**) またはクラスター外 (**out**) のいずれかになります。これは、稼働中 (**up**) が、ダウンしているか (**down**) のいずれかになります。OSD が **up** の場合は、データの読み取りおよび書き込みが可能なストレージクラスターにある (**in**) か、ストレージクラスターの外 (**out**) にあるかのいずれかになります。クラスター内 (**in**) にあり、最近クラスターの外 (**out**) に移動すると、Ceph は配置グループを他の OSD に移行します。OSD がクラスター **外** の場合、CRUSH は配置グループを OSD に割り当てません。OSD が **down** している場合は、それも **out** となるはずです。



注記

OSD が **down** して **in** にある場合は問題があり、クラスターは正常な状態になりません。



CEPH_459704_1017

ceph health、**ceph -s**、**ceph -w** などのコマンドを実行すると、クラスターが常に **HEALTH OK** をエコーバックしないことが分かります。慌てないでください。OSD に関連して、予想される状況でクラスターが **HEALTH OK** をエコー **しない** ことが予想されます。

- クラスターを起動していないと、応答しません。
- クラスターを起動または再起動したばかりで、配置グループが作成されつつあり、OSD がピアリング中であるため、準備はできていません。
- OSD を追加または削除したのみです。
- クラスターマップを変更しただけです。

OSD の監視の重要な要素は、クラスターの起動時および稼働時にクラスター **内** のすべての OSD が **稼働** していることを確認することです。

すべての OSD が実行中かどうかを確認するには、以下を実行します。

```
[root@mon ~]# ceph osd stat
```

または

```
[root@mon ~]# ceph osd dump
```

結果により、マップのエポック (**eNNNN**)、OSD の総数 (**x**)、いくつの **y** が **up** で、いくつの **z** が **in** であるかが分かります。

```
eNNNN: x osds: y up, z in
```

クラスターにある (**in**) OSD の数が、稼働中 (**up**) の OSD 数を超える場合。以下のコマンドを実行して、実行していない **ceph-osd** デーモンを特定します。

```
[root@mon ~]# ceph osd tree
```

例

```
# id    weight  type name   up/down reweight
-1  3   pool default
-3  3       rack mainrack
-2  3           host osd-host
0   1               osd.0   up  1
1   1               osd.1   up  1
2   1               osd.2   up  1
```

ヒント

適切に設計された CRUSH 階層で検索する機能は、物理ロケーションをより迅速に特定してストレージクラスターをトラブルシューティングするのに役立ちます。

OSD がダウンしている (**down**) 場合は、ノードに接続して開始します。Red Hat Storage コンソールを使用して OSD ノードを再起動するか、コマンドラインを使用できます。

例

```
[root@mon ~]# systemctl start ceph-osd@OSD_ID
```

3.2.12. 関連情報

- Red Hat Ceph Storage ダッシュボードガイド。

3.3. CEPH STORAGE クラスターの低レベルの監視

ストレージ管理者は、低レベルの視点から Red Hat Ceph Storage クラスターの正常性をモニターできます。通常、低レベルのモニタリングでは、Ceph OSD が適切にピアリングされるようにする必要があります。ピアの障害が発生すると、配置グループは動作が低下した状態で動作します。このパフォーマンスの低下状態は、ハードウェア障害、Ceph デーモンのハングまたはクラッシュした Ceph デーモン、ネットワークレイテンシー、完全なサイト停止など多くの異なる状態によって生じる可能性があります。

3.3.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

3.3.2. 配置グループセットの監視

CRUSH が配置グループを OSD に割り当てると、プールのレプリカ数を確認し、配置グループの各レプリカが別の OSD に割り当てられるように配置グループを OSD に割り当てます。たとえば、プールに配置グループの 3 つのレプリカが必要な場合、CRUSH はそれらをそれぞれ **osd.1**、**osd.2**、および

osd.3 に割り当てることができます。CRUSH は実際には、CRUSH マップで設定した障害ドメインを考慮した擬似ランダムな配置を求めているため、大規模なクラスター内で最も近い OSD に割り当てられた配置グループを目にすることはほとんどありません。特定の配置グループのレプリカを **Acting Set** として組み込む必要がある OSD のセットを参照します。場合によっては、Acting Set の OSD が **down** になった場合や、配置グループ内のオブジェクトのリクエストに対応できない場合があります。このような状況になっても、慌てないでください。以下に一般的な例を示します。

- OSD を追加または削除しています。次に、CRUSH は配置グループを他の OSD に再度割り当てます。これにより、動作セットの設定を変更し、バックフィルプロセスでデータの移行を生成します。
- OSD が **down** になり、再起動されてリカバリー中 (**recovering**) となっています。
- 動作セットの OSD は **down** となっているが、要求に対応できず、別の OSD がそのロールを一時的に想定しています。

Ceph は **Up Set** を使用してクライアント要求を処理します。これは、実際に要求を処理する OSD のセットです。ほとんどの場合、Up Set と Acting Set はほぼ同じです。そうでない場合には、Ceph がデータを移行しているか、OSD が復旧するか、問題がある場合に、通常 Ceph がこのようなシナリオで stuck stale メッセージと共に **HEALTH WARN** 状態を出すことを示しています。

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. 配置グループの一覧を取得するには、次のコマンドを実行します。

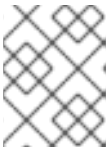
```
[root@mon ~]# ceph pg dump
```

2. Acting Set にどの OSD があるか、特定の配置グループの Up Set を表示するには、以下を実行します。

```
[root@mon ~]# ceph pg map PG_NUM
```

結果により、osdmap エポック (**eNNN**)、配置グループ番号 (**PG_NUM**)、Up Set の OSD (**up[]**(、動作セット (**acting[]**) であることが分かります。

```
[root@mon ~]# ceph osdmap eNNN pg PG_NUM-> up [0,1,2] acting [0,1,2]
```



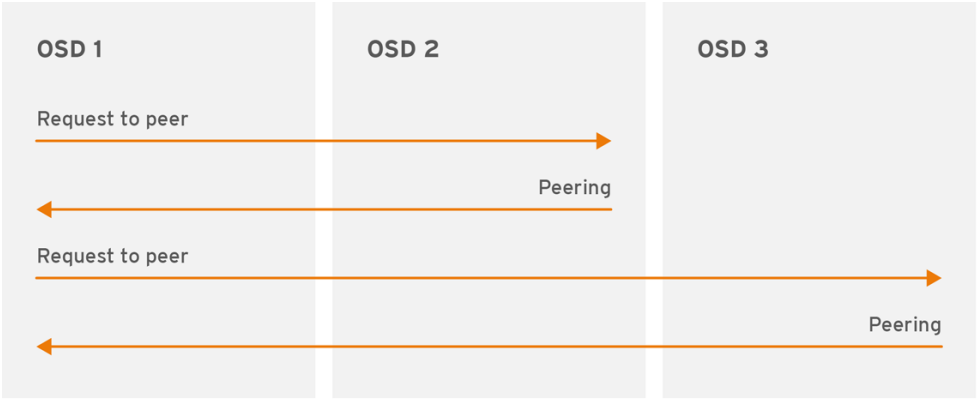
注記

Up Set と Acting Set が一致しない場合は、クラスター自体をリバランスするか、クラスターで潜在的な問題があることを示している可能性があります。

3.3.3. Ceph OSD のピアリング

配置グループにデータを書き込む前に、そのデータを **active** 状態にし、**clean** な状態で **なければなりません**。Ceph が配置グループの現在の状態を決定するためには、配置グループのプライマリー OSD、すなわちアクティングセットの最初の OSD が、セカンダリーおよびターシエリィー OSD とピアリング

を行い、配置グループの現在の状態についての合意を確立します。PG のレプリカが 3 つあるプールを想定します。



3.3.4. 配置グループの状態

ceph health、**ceph -s**、**ceph -w** などのコマンドを実行すると、クラスターが常に **HEALTH OK** をエコーバックしないことが分かります。OSD が実行中であるかを確認したら、配置グループのステータスも確認する必要があります。数多くの配置グループのピア関連状況で、クラスターが **HEALTH OK** を **しない** ことが予想されます。

- プールを作成したばかりで、配置グループはまだピアリングしていません。
- 配置グループは復旧しています。
- クラスターに OSD を追加したり、クラスターから OSD を削除したりしたところです。
- CRUSH マップを変更し、配置グループが移行中である必要があります。
- 配置グループの異なるレプリカに一貫性のないデータがあります。
- Ceph は配置グループのレプリカをスクラビングします。
- Ceph には、バックフィルの操作を完了するのに十分なストレージ容量がありません。

前述の状況のいずれかにより Ceph が **HEALTH WARN** をエコーしても慌てる必要はありません。多くの場合、クラスターは独自にリカバリーします。場合によっては、アクションを実行する必要がある場合があります。配置グループを監視する上で重要なことは、クラスターの起動時にすべての配置グループが **active** で、できれば **clean** な状態であることを確認することです。

すべての配置グループのステータスを表示するには、以下を実行します。

```
[root@mon ~]# ceph pg stat
```

その結果、配置グループマップバージョン (**vNNNNNN**)、配置グループの合計 (**x**)、および配置グループの数 (**y**) が、**active+clean** などの特定の状態にあることを示します。

```
vNNNNNN: x pgs: y active+clean; z bytes data, aa MB used, bb GB / cc GB avail
```

注記

Ceph では、配置グループについて複数の状態を報告するのが一般的です。

スナップショットトリミングの PG の状態

スナップショットが存在する場合には、追加の PG ステータスが 2 つ報告されます。

- **snaptrim**: PG は現在トリミング中です。
- **snaptrim_wait**: PG はトリム処理を待機中です。

出力例:

```
244 active+clean+snaptrim_wait
 32 active+clean+snaptrim
```

Ceph は、配置グループの状態に加えて、使用データ量 (**aa**)、ストレージ容量残量 (**bb**)、配置グループの総ストレージ容量をエコーバックします。いくつかのケースでは、これらの数字が重要になります。

- **near full ratio** または **full ratio** に達しています。
- CRUSH 設定のエラーにより、データがクラスター全体に分散されません。

配置グループ ID

配置グループ ID は、プール名ではなくプール番号で設定され、ピリオド (.) と配置グループ ID が続きます (16 進数)。**ceph osd lspools** の出力で、プール番号およびその名前を表示することができます。デフォルトのプール名 **data**、**metadata**、**rbd** はそれぞれプール番号 **0**、**1**、**2** に対応しています。完全修飾配置グループ ID の形式は以下のとおりです。

```
POOL_NUM.PG_ID
```

出力例:

```
0.1f
```

- 配置グループの一覧を取得するには、次のコマンドを実行します。

  ```
  [root@mon ~]# ceph pg dump
  ```

- JSON 形式で出力をフォーマットし、ファイルに保存するには、以下を実行します。

  ```
  [root@mon ~]# ceph pg dump -o FILE_NAME --format=json
  ```

- 特定の配置グループをクエリーするには、次のコマンドを実行します。

  ```
  [root@mon ~]# ceph pg POOL_NUM.PG_ID query
  ```

  JSON 形式の出力例:

  ```
  {
    "state": "active+clean",
    "up": [
      1,
      0
    ],
    "acting": [
      1,
  ```

```
    0
  ],
  "info": {
    "pgid": "1.e",
    "last_update": "4'1",
    "last_complete": "4'1",
    "log_tail": "0'0",
    "last_backfill": "MAX",
    "purged_snaps": "[]",
    "history": {
      "epoch_created": 1,
      "last_epoch_started": 537,
      "last_epoch_clean": 537,
      "last_epoch_split": 534,
      "same_up_since": 536,
      "same_interval_since": 536,
      "same_primary_since": 536,
      "last_scrub": "4'1",
      "last_scrub_stamp": "2013-01-25 10:12:23.828174"
    },
    "stats": {
      "version": "4'1",
      "reported": "536'782",
      "state": "active+clean",
      "last_fresh": "2013-01-25 10:12:23.828271",
      "last_change": "2013-01-25 10:12:23.828271",
      "last_active": "2013-01-25 10:12:23.828271",
      "last_clean": "2013-01-25 10:12:23.828271",
      "last_unstale": "2013-01-25 10:12:23.828271",
      "mapping_epoch": 535,
      "log_start": "0'0",
      "ondisk_log_start": "0'0",
      "created": 1,
      "last_epoch_clean": 1,
      "parent": "0.0",
      "parent_split_bits": 0,
      "last_scrub": "4'1",
      "last_scrub_stamp": "2013-01-25 10:12:23.828174",
      "log_size": 128,
      "ondisk_log_size": 128,
      "stat_sum": {
        "num_bytes": 205,
        "num_objects": 1,
        "num_object_clones": 0,
        "num_object_copies": 0,
        "num_objects_missing_on_primary": 0,
        "num_objects_degraded": 0,
        "num_objects_unfound": 0,
        "num_read": 1,
        "num_read_kb": 0,
        "num_write": 3,
        "num_write_kb": 1
      },
      "stat_cat_sum": {

      },
```

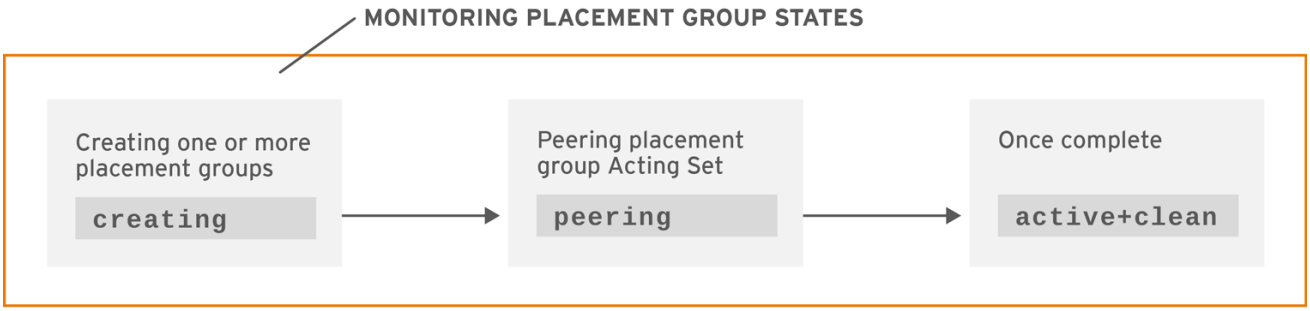
```
      "up": [
        1,
        0
      ],
      "acting": [
        1,
        0
      ]
    },
    "empty": 0,
    "dne": 0,
    "incomplete": 0
  },
  "recovery_state": [
    {
      "name": "Started\/Primary\/Active",
      "enter_time": "2013-01-23 09:35:37.594691",
      "might_have_unfound": [

      ],
      "scrub": {
        "scrub_epoch_start": "536",
        "scrub_active": 0,
        "scrub_block_writes": 0,
        "finalizing_scrub": 0,
        "scrub_waiting_on": 0,
        "scrub_waiting_on_whom": [

        ]
      }
    },
    {
      "name": "Started",
      "enter_time": "2013-01-23 09:35:31.581160"
    }
  ]
}
```


関連情報

- スナップショットトリミングの設定に関する詳細は、Red Hat Ceph Storage 4 の **設定ガイド** の**OSD (Object Storage Daemon) の設定オプション**の章を参照してください。

3.3.5. 配置グループの状態の作成

プールを作成すると、指定した数の配置グループが作成されます。Ceph は、1 つ以上の配置グループの **作成** 時に作成をエコーします。これが作成されると、配置グループのアクティングセットの一部である OSD がピアリングを行います。ピアリングが完了すると、配置グループのステータスは **active+clean** になり、Ceph クライアントが配置グループへの書き込みを開始できるようになります。



3.3.6. 配置グループのピア状態

Ceph が配置グループをピアリングする場合、Ceph は配置グループのレプリカを保存する OSD を配置グループ内のオブジェクトおよびメタデータの **状態について合意** に持ち込みます。Ceph がピアリングを完了すると、配置グループを格納する OSD が配置グループの現在の状態について合意することを意味します。ただし、ピアリングプロセスを完了しても、各レプリカに最新のコンテンツがある **わけではありません**。

権威の履歴

Ceph は、動作セットのすべての OSD が書き込み操作を持続させるまで、クライアントへの書き込み操作を **承認しません**。これにより、有効なセットの少なくとも 1 つメンバーが、最後に成功したピア操作以降の確認済みの書き込み操作がすべて記録されるようになります。

それぞれの確認応答書き込み操作の正確なレコードにより、Ceph は配置グループの新しい権威履歴を構築して公開することができます。完全かつ完全に命令された一連の操作が実行されれば、OSD の配置グループのコピーを最新の状態にすることができます。

3.3.7. 配置グループのアクティブな状態

Ceph がピア処理を完了すると、配置グループが **active** になる可能性があります。**active** 状態とは、配置グループのデータがプライマリー配置グループで一般的に利用可能で、読み取り操作および書き込み操作用のレプリカになります。

3.3.8. 配置グループの clean の状態

配置グループが **クリーン** な状態にある場合、プライマリー OSD とレプリカ OSD は正常にピアリングを行い、配置グループ用の迷子のレプリカが存在しないことを意味します。Ceph は、配置グループ内のすべてのオブジェクトを正しい回数で複製します。

3.3.9. 配置グループの状態が低下した状態

クライアントがプライマリー OSD にオブジェクトを書き込む際に、プライマリー OSD はレプリカ OSD にレプリカを書き込むロールを担います。プライマリー OSD がオブジェクトをストレージに書き込んだ後に、配置グループは、Ceph がレプリカオブジェクトを正しく作成したレプリカ OSD からプライマリー OSD が確認応答を受け取るまで、動作が **低下** した状態になります。

配置グループが **active+degraded** になる理由は、OSD がまだすべてのオブジェクトを保持していない場合でも **active** である可能性があることです。OSD が **down** する場合、Ceph は OSD に割り当てられた各配置グループを **degraded** としてマークします。OSD がオンラインに戻る際に、OSD を再度ピアする必要があります。ただし、クライアントは、**active** であれば、**degraded** である配置グループに新しいオブジェクトを記述できます。

OSD が **down** していてパフォーマンスの低下 (**degraded**) が続く場合には、Ceph は **down** 状態である

OSD をクラスターの外 (**out**) としてマークし、**down** 状態である OSD から別の OSD にデータを再マッピングする場合があります。**down** とマークされた時間と **out** とマークされた時間の間の時間は **mon_osd_down_out_interval** によって制御され、デフォルトでは **600** に設定されています。

また、配置グループは、Ceph が配置グループにあるべきだと考えるオブジェクトを 1 つ以上見つけることができないため、**低下** してしまうこともあります。未検出オブジェクトへの読み取りまたは書き込みはできませんが、動作が低下した (**degraded**) 配置グループ内の他のすべてのオブジェクトにアクセスできます。

3 方のレプリカプールに 9 つの OSD があるとします。OSD の数の 9 がダウンすると、9 の OSD に割り当てられた PG は動作が低下します。OSD 9 がリカバリーされない場合は、クラスターから送信され、クラスターがリバランスします。このシナリオでは、PG のパフォーマンスが低下してから、アクティブな状態に戻ります。

3.3.10. 配置グループの状態のリカバリー

Ceph は、ハードウェアやソフトウェアの問題が継続している規模でのフォールトトレランスを目的として設計されています。OSD がダウンする (**down**) と、そのコンテンツは配置グループ内の他のレプリカの現在の状態のままになる可能性があります。OSD が **up** 状態に戻ったら、配置グループの内容を更新して、現在の状態を反映させる必要があります。その間、OSD は **リカバリー** の状態を反映する場合があります。

ハードウェアの故障は、複数の OSD のカスケード障害を引き起こす可能性があるため、回復は常に些細なことではありません。たとえば、ラックやキャビネット用のネットワークスイッチが故障して、多数のホストマシンの OSD がクラスターの現在の状態から遅れてしまうことがあります。各 OSD は、障害が解決されたら回復しなければなりません。

Ceph は、新しいサービス要求とデータオブジェクトの回復と配置グループを現在の状態に復元するニーズの間でリソース競合のバランスを取るためのいくつかの設定を提供しています。**osd recovery delay start** 設定により、回復プロセスを開始する前に OSD を再起動し、ピアリングを再度行い、さらにはいくつかの再生要求を処理できます。**osd recovery threads** 設定により、デフォルトで 1 つのスレッドでリカバリープロセスのスレッド数が制限されます。**osd recovery thread timeout** は、複数の OSD が驚きの速さで失敗、再起動、再ピアする可能性があるため、スレッドタイムアウトを設定します。**osd recovery max active** 設定では、OSD が送信に失敗するのを防ぐために OSD が同時に実行するリカバリー要求の数を制限します。**osd recovery の max chunk** 設定により、復元されたデータチャンクのサイズが制限され、ネットワークの輻輳を防ぐことができます。

3.3.11. バックフィルの状態

新規 OSD がクラスターに参加する際に、CRUSH はクラスター内の OSD から新たに追加された OSD に配置グループを再割り当てします。新規 OSD が再割り当てされた配置グループをすぐに許可するように強制すると、新規 OSD に過剰な負荷が生じる可能性があります。OSD を配置グループでバックフィルすると、このプロセスはバックグラウンドで開始できます。バックフィルが完了すると、新しい OSD の準備が整い次第、リクエストへの対応を開始します。

バックフィル操作中は、いくつかの状態のうちの 1 つが表示されます。* **backfill_wait** は、バックフィル操作が保留されているが、まだ進行していないことを示します * **backfill** は、バックフィル操作が進行中であることを示します * **backfill_too_full** は、バックフィル操作が要求されたが、ストレージ容量が不足しているために完了できなかったことを示します。

配置グループをバックフィルできない場合は、**incomplete** とみなされることがあります。

Ceph は、OSD、特に新しい OSD への配置グループの再割り当てに伴い負荷の急増を管理するいくつかの設定を提供しています。デフォルトでは、**osd_max_backfills** は、OSD から 10 への同時バックフィルの最大数を設定します。**osd backfill full ratio** により、OSD は、OSD が完全な比率 (デフォルトでは 85%) に近づけている場合にバックフィル要求を拒否することができます。OSD がバックフィル

要求を拒否する場合は、**osd backfill retry interval** により、OSD はデフォルトで 10 秒後に要求を再試行できます。また、OSD は、スキャン間隔 (デフォルトで 64 および 512) を管理するために、**osd backfill scan min** および **osd backfill scan max** を設定することもできます。

ワークロードによっては、通常のリカバリーを完全に回避し、代わりにバックフィルを使用することが推奨されます。バックフィルはバックグラウンドで実行されるため、I/O は OSD のオブジェクトで続行できます。復元せずにバックフィルを強制するには、**osd_min_pg_log_entries** を **1** に設定し、**osd_max_pg_log_entries** を **2** に設定します。この状況がご使用のワークロードに適切な場合についての詳細は、Red Hat サポートアカウントチームにお問い合わせください。

3.3.12. リカバリーまたはバックフィル操作の優先度の変更

一部の配置グループ (PG) にリカバリーやバックフィルが必要で、一部の配置グループに他のグループよりも重要なデータが含まれているという状況が発生する場合があります。**pg force-recovery** または **pg force-backfill** コマンドを使用して、優先度の高いデータを持つ PG が最初にリカバリーまたはバックフィルされるようにします。

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. **pg force-recovery** または **pg force-backfill** コマンドを実行し、優先度の高いデータを持つ PG の優先順位を指定します。

   構文

   ```
   ceph pg force-recovery PG1 [PG2] [PG3 ...]
   ceph pg force-backfill PG1 [PG2] [PG3 ...]
   ```

   例

   ```
   [root@node]# ceph pg force-recovery group1 group2
   [root@node]# ceph pg force-backfill group1 group2
   ```

   このコマンドにより、Red Hat Ceph Storage は指定された配置グループ (PG) でリカバリーまたはバックフィルを実行してから、他の配置グループを処理します。コマンドを実行しても、現在実行中のバックフィルまたはリカバリー操作は中断されません。現在実行中の操作が終了した後、指定の PG に対してできるだけ早期にリカバリーまたはバックフィルが行われます。

3.3.13. 指定の配置グループでリカバリーまたはバックフィルの操作を変更またはキャンセル

ストレージクラスター内の特定の配置グループ (PG) で優先度の高い **force-recovery** または **force-backfill** 操作をキャンセルすると、これらの PG の操作はデフォルトのリカバリー設定またはバックフィル設定に戻ります。

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. 指定の配置グループでリカバリーまたはバックフィルの操作を変更またはキャンセルするには、以下を実行します。

   構文

   ```
   ceph pg cancel-force-recovery PG1 [PG2] [PG3 ...]
   ceph pg cancel-force-backfill PG1 [PG2] [PG3 ...]
   ```

   例

   ```
   [root@node]# ceph pg cancel-force-recovery group1 group2
   [root@node]# ceph pg cancel-force-backfill group1 group2
   ```

   これにより、**force** フラグが取り消され、デフォルトの順序で PG を処理します。

   指定された PG のリカバリーまたはバックフィル操作が完了したら、処理の順序がデフォルトに戻ります。

関連情報

- RADOS でのリカバリーおよびバックフィル操作の優先順位の詳細については、RADOS での配置グループのリカバリーおよびバックフィル の優先順位を参照してください。

3.3.14. プールのリカバリーまたはバックフィル操作の優先度の強制

プールのすべての配置グループに優先度の高いリカバリーまたはバックフィルが必要な場合は、**force-recovery** または **force-backfill** オプションを使用して操作を開始します。

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. 指定されたプールのすべての配置グループで優先度の高いリカバリーまたはバックフィルを強制するには、以下を実行します。

   構文

   ```
   ceph osd pool force-recovery POOL_NAME
   ceph osd pool force-backfill POOL_NAME
   ```

   例

   ```
   [root@node]# ceph osd pool force-recovery pool1
   [root@node]# ceph osd pool force-backfill pool1
   ```

注記

force-recovery および **force-backfill** コマンドの使用には注意が必要です。これらの操作の優先度を変更すると、Ceph の内部優先度計算の順序付けが乱れる可能性があります。

3.3.15. プールのリカバリーまたはバックフィル操作の優先度のキャンセル

プールのすべての配置グループで優先度の高い **force-recovery** または **force-backfill** 操作をキャンセルすると、そのプールの PG の操作はデフォルトのリカバリーまたはバックフィルの設定に戻ります。

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. 指定されたプールのすべての配置グループで優先度の高いリカバリーまたはバックフィル操作をキャンセルするには、以下を実行します。

   構文

   ```
   ceph osd pool cancel-force-recovery POOL_NAME
   ceph osd pool cancel-force-backfill POOL_NAME
   ```

   例

   ```
   [root@node]# ceph osd pool cancel-force-recovery pool1
   [root@node]# ceph osd pool cancel-force-backfill pool1
   ```

3.3.16. プールのリカバリー操作またはバックフィルの操作の優先度の再調整

現在、同じ基礎となる OSD を使用している複数のプールがあり、一部のプールに優先度の高いデータが含まれている場合、操作の実行順序を再調整できます。**recovery_priority** オプションを使用して、優先度の高いデータのあるプールに優先度の高い値を割り当てます。これらのプールは、優先度の低い値を持つプールやデフォルトの優先度に設定されたプールよりも先に実行されます。

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. プールのリカバリーやバックフィルの優先度を再調整するには、以下を実行します。

   構文

   ```
   ceph osd pool set POOL_NAME recovery_priority VALUE
   ```

例

```
ceph osd pool set pool1 recovery_priority 10
```

VALUE は優先順位を設定します。たとえば、プールが 10 個ある場合、優先度の値が 10 のプールは最初に処理され、次に優先順位が 9 のプールが処理されます。一部のプールのみ優先度が高い場合は、そのプールのみにに優先度の値を設定できます。優先度の値が設定されていないプールは、デフォルトの順序で処理されます。

3.3.17. RADOS における配置グループのリカバリーの優先順位

ここでは、RADOS における配置グループ (PG) のリカバリーおよびバックフィルの相対的な優先度の値について説明します。高い値が先に処理されます。非アクティブな PG の値は、アクティブまたはデグレードの PG よりも優先度が高い値になります。

操作	値	詳細
OSD_RECOVERY_PRIORITY_MIN	0	最小リカバリー値
OSD_BACKFILL_PRIORITY_BASE	100	MBackfillReserve のベースのバックフィル優先度
OSD_BACKFILL_DEGRADED_PRIORITY_BASE	140	MBackfillReserve のベースのバックフィル優先度 (デグレード PG)
OSD_RECOVERY_PRIORITY_BASE	180	MBackfillReserve のベースのリカバリー優先度
OSD_BACKFILL_INACTIVE_PRIORITY_BASE	220	MBackfillReserve のベースのバックフィル優先度 (非アクティブ PG)
OSD_RECOVERY_INACTIVE_PRIORITY_BASE	220	MRecoveryReserve のベースのリカバリー優先度 (非アクティブ)
OSD_RECOVERY_PRIORITY_MAX	253	MBackfillReserve の手動または自動で設定される最大リカバリー優先度
OSD_BACKFILL_PRIORITY_FORCED	254	MBackfillReserve のバックフィル優先度 (手動で強制)
OSD_RECOVERY_PRIORITY_FORCED	255	MRecoveryReserve のリカバリー優先度 (手動で強制)
OSD_DELETE_PRIORITY_NORMAL	179	OSD が満杯状態でない場合の PG の削除の優先度
OSD_DELETE_PRIORITY_FULLISH	219	OSD がほぼ満杯状態である場合の PG の削除の優先度

操作	値	詳細
OSD_DELETE_PRIORITY_FULL	255	OSD が満杯である場合の削除の優先度

3.3.18. 配置グループの再マッピングの状態

配置グループにサービスを提供する動作セットが変更すると、古い動作セットから新しい動作セットにデータを移行します。新規プライマリー OSD がリクエストを処理するには、多少時間がかかる場合があります。したがって、配置グループの移行が完了するまで、古いプライマリーに要求への対応を継続するように依頼する場合があります。データの移行が完了すると、マッピングは新しい動作セットのプライマリー OSD を使用します。

3.3.19. 配置グループの stale 状態

Ceph はハートビートを使用してホストとデーモンが実行されていることを確認しますが、**ceph-osd** デーモンも **スタック** 状態になり、統計をタイムリーに報告しない場合があります。たとえば、一時的なネットワーク障害などが挙げられます。デフォルトでは、OSD デーモンは、配置グループ、アップスルー、ブート、失敗の統計情報を半秒 (**0.5**) ごとに報告しますが、これはハートビートのしきい値よりも頻度が高くなります。配置グループの動作セットの **プライマリー OSD** がモニターへの報告に失敗した場合や、他の OSD がプライマリー OSD の **down** を報告した場合、モニターは配置グループに **stale** マークを付けます。

ストレージクラスターを起動すると、ピアリング処理が完了するまで **stale** 状態になるのが一般的です。ストレージクラスターがしばらく稼働している間に、配置グループが **stale** 状態になっているのが確認された場合は、その配置グループのプライマリー OSD が **down** になっているか、モニターに配置グループの統計情報を報告していないことを示しています。

3.3.20. 配置グループの不配置の状態

PG が OSD に一時的にマップされる一時的なバックフィルシナリオがあります。**一時的** な状況がなくなった場合には、PG は一時的な場所に留まり、適切な場所にない可能性があります。いずれの場合も、それらは **誤って配置** されます。それは、実際には正しい数の追加コピーが存在しているのに、1 つ以上のコピーが間違った場所にあるためです。

たとえば、3 つの OSD が 0、1、2 であり、すべての PG はこれらの 3 つのうちのいくつかの配列にマップされます。別の OSD (OSD 3) を追加する場合、一部の PG は、他のものではなく OSD 3 にマッピングされるようになりました。しかし、OSD 3 がバックフィルされるまで、PG には一時的なマッピングがあり、古いマッピングからの I/O を提供し続けることができます。その間、PG には一時的な待っピンがありますが、コピーが 3 つあるため **degraded** はしていないため、間違った場所に置かれます (**misplaced**)。

例

```
pg 1.5: up=acting: [0,1,2]
ADD_OSD_3
pg 1.5: up: [0,3,1] acting: [0,1,2]
```

[0,1,2] は一時的なマッピングです。そのため、**up** セットは **acting** なセットとは等しくならず、[0,1,2] がまだ 3 つのコピーであるため、PG は誤って配置されます (**misplaced**) が、パフォーマンスは低下 (**degraded**) しません。

例

```
pg 1.5: up=acting: [0,3,1]
```

OSD 3 はバックフィルされ、一時的なマッピングは削除され、パフォーマンスは低下せず、誤って配置されなくなりました。

3.3.21. 配置グループの不完全な状態

PG は、不完全なコンテンツがあり、ピアリングが失敗したとき、すなわち、リカバリーを実行するためのな現在の完全な OSD が十分にないときに、**incomplete** 状態になる。

例えば、OSD 1、2、3 が動作中の OSD セットで、それが OSD 1、4、3 に切り替わったとすると、**osd.1** が 4 をバックフィルする間、OSD 1、2、3 の一時的な動作中の OSD セットを要求することになリマス。この間、OSD 1、2、および 3 すべてがダウンすると、**osd.4** は、すべてのデータが完全にバックフィルされていない可能性がある唯一のものとして残されています。このとき、PG は **incomplete** となり、リカバリーを実行するのに十分な現在の完全な OSD がないことを示す不完全な状態になります。

別の方法として、**osd.4** が関与しておらず、OSD の 1、2、3 がダウンしたときに動作セットが単に OSD 1、2、3 になっている場合、PG はおそらく動作セットが変更されてからその PG で何も聞いていないことを示す **stale** になります。新規 OSD に通知する OSD がない理由。

3.3.22. スタックした配置グループの特定

前述のように、配置グループは、その状態が **active+clean** ではないため、必ずしも問題になるとは限りません。一般的に、Ceph の自己修復機能は、配置グループが停止しても機能しない場合があります。スタック状態には、以下が含まれます。

- **Unclean**: 配置グループには、必要な回数複製しないオブジェクトが含まれます。これらは回復中である必要があります。
- **Inactive**: 配置グループは、最新のデータを持つ OSD が **up** に戻るのを待っているため、読み取りや書き込みを処理できません。
- **Stale**: 配置グループは不明な状態です。配置グループは、これらをホストする OSD がしばらくモニタークラスターに報告されず、**mon osd report timeout** 設定で設定できるためです。

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. スタックした配置グループを特定するには、以下のコマンドを実行します。

```
ceph pg dump_stuck {inactive|unclean|stale|undersized|degraded
[inactive|unclean|stale|undersized|degraded...]} {<int>}
```

3.3.23. オブジェクトの場所の検索

Ceph クライアントは最新のクラスターマップを取得し、CRUSH アルゴリズムはオブジェクトを配置グループにマッピングする方法を計算してから、配置グループを OSD に動的に割り当てる方法を計算します。

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. オブジェクトの場所を見つけるには、オブジェクト名とプール名のみが必要です。

```
ceph osd map POOL_NAME OBJECT_NAME
```

第4章 CEPH の動作のオーバーライド

ストレージ管理者は、ランタイム時に Red Hat Ceph Storage クラスターのオーバーライドを使用して Ceph オプションを変更する方法を理解する必要があります。

4.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

4.2. CEPH のオーバーライドオプションの設定および設定解除

Ceph のデフォルト動作を上書きするために、Ceph オプションを設定および設定解除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。
- ノードへのルートレベルのアクセス。

手順

1. Ceph のデフォルトの動作を上書きするには、**ceph osd set** コマンドおよび上書きする動作を使用します。

   ```
   ceph osd set FLAG
   ```

   動作を設定したら、**ceph health** には、クラスターに設定したオーバーライドが反映されます。

2. Ceph のデフォルトの動作を上書きするには、**ceph osd unset** コマンドおよび停止するオーバーライドを使用します。

   ```
   ceph osd unset FLAG
   ```

フラグ	説明
noin	OSD が、クラスター **内** での扱いになるのを防ぎます。
noout	OSD が、クラスター **外** での扱いになるのを防ぎます。
noup	OSD が **up** で稼働している扱いになるのを防ぎます。
nodown	OSD が **down** 扱いされるのを防ぎます。
full	クラスターが **full_ratio** に達したように表示され、書き込み操作が阻止されます。
pause	Ceph は読み取り操作および書き込み操作の処理を停止しますが、ステータスの OSD のステータス **in**、**out**、**up**、または **down** は影響を受けません。

フラグ	説明
nobackfill	Ceph により、新しいバックフィルの操作が回避されます。
norebalance	Ceph により、新たなリバランス操作が回避されます。
norecover	Ceph により、新たなリカバリー操作が回避されます。
noscrub	Ceph は新規スクラビングの操作を回避します。
nodeep-scrub	Ceph は、新たな深層スクラブ作業を行いません。
notieragent	Ceph は、コールド/ダーティーオブジェクトをフラッシュしてエビクトすることを目的とするプロセスを無効にします。

4.3. CEPH のオーバーライドのユースケース

- **noin**: フラッピング OSD に対応するために、多くの場合 **noout** と一緒に使用されます。
- **noout**: **mon osd report timeout** を超え、OSD がモニターに報告されていない場合には、OSD は **out** とマークされます。誤って発生する場合は、問題のトラブルシューティング中に OSD が **out** とマークされないように **noout** を設定できます。
- **noup**: 一般的に、**nodown** で使用され、フラグッピング OSD に対応します。
- **nodown**: ネットワークの問題が Ceph の heartbeat プロセスが中断する可能性があり、OSD が **up** にある可能性がありますが、down をマークされる場合もあります。**nodown** を設定すると、問題のトラブルシューティング中に OSD が down をマークされないようにできます。
- **full**: クラスターが **full_ratio** に到達する場合は、事前にクラスターを **full** に設定し、容量を拡張することができます。

注記

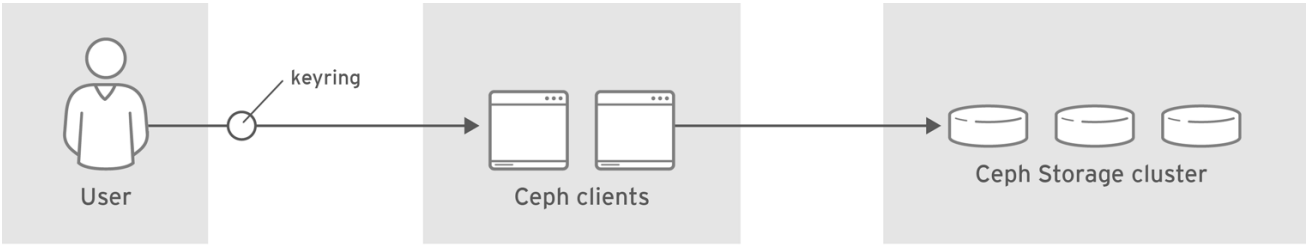
クラスターを **full** に設定すると書き込み操作ができなくなります。

- **pause**: クライアントがデータの読み取りおよび書き込みを行わずに実行中の Ceph クラスターをトラブルシューティングする必要がある場合は、クライアントの操作を防ぐためにクラスターを **一時停止** するように設定できます。
- **nobackfill**: OSD またはノードを一時的に **down** する必要がある場合 (デーモンのアップグレードなど) は、OSD が **down** になっている間に Ceph がバックフィルしないように、**nobackfill** を設定できます。
- **norecover**: OSD を置き換える必要がえあり、ディスクをホットスワップする間に PG を別の OSD に復元しないようする場合は、他の OSD のセットが他の OSD に新しいセットをコピーしないように、**norecover** も設定できます。
- **noscrub** および **nodeep-scrubb**: たとえば、高負荷、復旧、バックフィル、およびリバランス中のオーバーヘッドを減らすためにスクラビングを防ぐために、**noscrub** や **nodeep-scrub** を設定して、クラスターが OSD をスクラビングしないようにすることができます。

- **notieragent**: 階層エージェントプロセスで、バッキングストレージ層にコールドオブジェクトを検索しないようにするには、**notieragent** を設定する可能性があります。

第5章 CEPH ユーザー管理

ストレージ管理者は、Red Hat Ceph Storage クラスターのオブジェクトへの認証、キーリング管理、およびアクセス制御を提供することで、Ceph ユーザーのベースを管理できます。



5.1. 前提条件

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

5.2. CEPH ユーザー管理の背景

Ceph が認証および認可が有効な状態で実行された場合には、ユーザー名およびキーリングに、指定したユーザーの秘密鍵が含まれるキーリングを指定する必要があります。ユーザー名を指定しない場合、Ceph は **client.admin** 管理ユーザーをデフォルトのユーザー名として使用します。キーリングを指定しない場合には、Ceph 設定の **keyring** 設定を使用してキーリングを探します。たとえば、ユーザーやキーリングを指定せずに **ceph health** コマンドを実行する場合は、以下を実行します。

```
# ceph health
```

Ceph は以下のようなコマンドを解釈します。

```
# ceph -n client.admin --keyring=/etc/ceph/ceph.client.admin.keyring health
```

ユーザー名およびシークレットの再入力を避けるために、**CEPH_ARGS** 環境変数を使用できます。

Ceph クライアントのタイプ (ブロックデバイス、オブジェクトストア、ファイルシステム、ネイティブ API、Ceph コマンドラインなど) に関係なく、Ceph はすべてのデータをオブジェクトとしてプールに保存します。データの読み取りおよび書き込みを行うには、Ceph ユーザーはプールにアクセスできる必要があります。また、管理用 Ceph ユーザーには、Ceph の管理コマンドを実行するパーミッションが必要です。

Ceph ユーザー管理の概念は以下のとおりです。

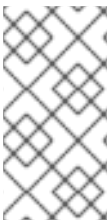
ストレージクラスターユーザー

Red Hat Ceph Storage クラスターのユーザーは、個別またはアプリケーションです。ユーザーを作成することで、誰がストレージクラスター、そのプール、およびそれらのプール内のデータにアクセスできるかを制御することができます。

Ceph の概念にはユーザーの **タイプ** があります。ユーザー管理の目的で、タイプは常に **client** になります。Ceph は、ユーザータイプとユーザー ID で設定されるピリオド (.) で区切られたユーザーを識別します。たとえば、**TYPE.ID**、**client.admin**、**client.user1** などです。ユーザーの入力は、Ceph

Monitor および OSD も Cephx プロトコルを使用しますが、それらはクライアントではないために必要になります。ユーザータイプの分類することにより、クライアントユーザーと他のユーザーを区別でき、アクセス制御、ユーザーの監視および追跡可能性をさらに単純化します。

Ceph コマンドラインを使用すると、コマンドラインでの使用に応じて、タイプを使用せずにユーザーを指定できるため、Ceph のユーザータイプが混乱する場合があります。**--user** または **--id** を指定した場合は、タイプを省略できます。そのため、**client.user1** は **user1** として簡単に入力できます。**--name** または **-n** を指定する場合は、**client.user1** などのタイプおよび名前を指定する必要があります。Red Hat は、可能な限り、タイプと名前をベストプラクティスとして使用することを推奨します。



注記

Red Hat Ceph Storage クラスターユーザーは、Ceph Object Gateway ユーザーと同じではありません。オブジェクトゲートウェイは Red Hat Ceph Storage クラスターユーザーを使用してゲートウェイデーモンとストレージクラスター間の通信を行いますが、ゲートウェイにはエンドユーザー向けの独自のユーザー管理機能があります。

認証ケイパビリティー

Ceph は、認証されたユーザーが Ceph Monitors および OSD の機能を行使する認可することを示すためにケイパビリティー (capabilities/caps) という用語を使用しています。ケイパビリティーは、プール内のデータやプール内の namespace へのアクセスを制限することもできます。ユーザーを作成または更新する際に、Ceph 管理ユーザーはユーザーのケイパビリティーを設定します。ケイパビリティーの構文は以下の形式に従います。

構文

```
DAEMON_TYPE 'allow CAPABILITY' [DAEMON_TYPE 'allow CAPABILITY']
```

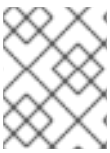
- **Monitor Caps:** モニターのケイパビリティーには、**r**、**w**、**x**、**allow profile CAP**、および **profile rbd** があります。

  例

  ```
  mon 'allow rwx`
  mon 'allow profile osd'
  ```

- **OSD Caps:** OSD ケイパビリティーには、**r**、**w**、**x**、**class-read**、**class-write**、**profile osd**、**profile rbd**、および **profile rbd-read-only** があります。さらに OSD ケイパビリティーは、プールおよび namespace の設定も許可します。

  ```
  osd 'allow CAPABILITY' [pool=POOL_NAME] [namespace=NAMESPACE_NAME]
  ```



注記

Ceph Object Gateway デーモン (**radosgw**) は Ceph ストレージクラスターのクライアントであるため、Ceph Storage クラスターデーモンタイプとしては表示されません。

以下のエントリーは、それぞれのケイパビリティーについて説明します。

allow	デーモンのアクセス設定の前に使用してください。

r	ユーザーに読み取り権限を付与します。CRUSH マップを取得するためにモニターで必要です。
w	ユーザーがオブジェクトへの書き込みアクセス権を付与します。
x	クラスメソッド (読み取りおよび書き込みの両方) をユーザーに呼び出し、モニターで **auth** 操作を実行する機能を提供します。
class-read	クラスの読み取りメソッドを呼び出すケイパビリティーを提供します。**x** のサブセット。
class-write	クラスの書き込みメソッドを呼び出すケイパビリティーを提供します。**x** のサブセット。
*	特定のデーモンまたはプールに対する読み取り、書き込み、実行のパーミッション、および admin コマンドの実行権限をユーザーに付与します。
profile osd	OSD として他の OSD またはモニターに接続するためのパーミッションをユーザーに付与します。OSD がレプリケーションのハートビートトラフィックおよびステータス報告を処理できるようにするために OSD に付与されました。
profile bootstrap-osd	OSD のブートストラップ時にキーを追加するパーミッションを持つように、OSD をブートストラップするユーザーパーミッションを付与します。
profile rbd	ユーザーに、Ceph ブロックデバイスへの読み取り/書き込み権限を付与します。
profile rbd-read-only	ユーザーに、Ceph ブロックデバイスへの読み取り専用アクセス権を付与します。

プール

プールは Ceph クライアントのストレージストラテジーを定義し、そのストラテジーの論理パーティションとして機能します。

Ceph デプロイメントでは、さまざまな種類のユースケースをサポートするプールを作成することが一般的です。たとえば、クラウドボリュームまたはイメージ、オブジェクトストレージ、ホットストレージ、コールドストレージなど。OpenStack のバックエンドとして Ceph をデプロイする場合、標準的なデプロイメントにはボリューム、イメージ、バックアップと、**client.glance**、**client.cinder** などユーザーのプールが含まれます。

Namespace

プール内のオブジェクトは、プール内のオブジェクトの論理グループである namespace に関連付けることができます。ユーザーのプールへのアクセスは、ユーザーによる読み書きが名前空間内でのみ行われるように、その名前空間と関連付けることができます。プール内の名前空間に書き込まれたオブジェクトには、その名前空間にアクセスできるユーザーのみがアクセスできます。

注記

現在、名前空間は、**librados** に記述されたアプリケーションにのみ役立ちます。ブロックデバイスやオブジェクトストレージなどの Ceph クライアントでは、この機能は現在サポートされていません。

名前空間の合理的な理由は、各プールが OSD にマッピングされる配置グループのセットを作成するため、プールはユースケース別にデータを分離するために計算量の多い方法になる可能性があるからです。複数のプールが同じ CRUSH 階層とルールセットを使用する場合、OSD のパフォーマンスは負荷の増加に応じて低下する可能性があります。

たとえば、プールには、OSD ごとに約 100 個の配置グループが必要です。そのため、1000 個 の OSD を持つ模範的なクラスターは、1 つのプールに対して 10 万個の配置グループを持つことになります。同じ CRUSH 階層とルールセットにマップされた各プールは、例示的なクラスターにさらに 10 万の配置グループを作成します。一方、namespace にオブジェクトを書き込むと、別のプールの計算オーバーヘッドを排除して、namespace をオブジェクト名に関連付けられます。ユーザーまたはユーザーセットに個別のプールを作成するのではなく、名前空間を使用できます。

注記

現時点では、**librados** のみを使用できます。

関連情報

- 認証の使用に関する詳細は、Red Hat Ceph Storage 設定ガイドを参照してください。

5.3. CEPH ユーザーの管理

ストレージ管理者は、ユーザーの作成、修正、削除、およびインポートにより Ceph ユーザーを管理できます。Ceph クライアントユーザーは、Ceph クライアントを使用して Red Hat Ceph Storage クラスターデーモンと対話する個人またはアプリケーションのいずれかになります。

5.3.1. 前提条件

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

5.3.2. Ceph ユーザーの一覧表示

コマンドラインインターフェイスを使用して、ストレージクラスター内のユーザーを一覧表示できます。

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. ストレージクラスターのユーザーを一覧表示するには、以下を実行します。

```
[root@mon ~]# ceph auth list
```

Ceph はストレージクラスター内の全ユーザーを一覧表示します。例えば、2 ノードの例示的なストレージクラスターでは、**ceph auth list** は以下のようなものを出力します。

例

```
installed auth entries:

osd.0
    key: AQCvCbtToC6MDhAATtuT70Sl+DymPCfDSsyV4w==
    caps: [mon] allow profile osd
    caps: [osd] allow *
osd.1
    key: AQC4CbtTCFJBChAAVq5spj0ff4eHZICxIOVZeA==
    caps: [mon] allow profile osd
    caps: [osd] allow *
client.admin
    key: AQBHCbtT6APDHhAA5W00cBchwkQjh3dkKsyPjw==
    caps: [mds] allow
    caps: [mon] allow *
    caps: [osd] allow *
client.bootstrap-mds
    key: AQBICbtTOK9uGBAAdbe5zcIGHZL3T/u2g6EBww==
    caps: [mon] allow profile bootstrap-mds
client.bootstrap-osd
    key: AQBHCbtT4GxqORAADE5u7RkpCN/oo4e5W0uBtw==
    caps: [mon] allow profile bootstrap-osd
```

注記

ユーザーの **TYPE.ID** 記法が適用され、**osd.0** は **osd** 型のユーザーでその ID は **0**、**client.admin** は **client** 型のユーザーでその ID は **admin**、つまりデフォルトの **client.admin** ユーザーとなります。また、各エントリーには、**key: VALUE** エントリー、および 1 つ以上の **caps:** エントリーがあることに注意してください。

-o FILE_NAME オプションを **ceph auth list** と共に使用して、出力をファイルに保存することができます。

5.3.3. Ceph ユーザー情報の表示

コマンドラインインターフェイスを使用して、Ceph のユーザー情報を表示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。
- ノードへのルートレベルのアクセス。

手順

1. 特定のユーザー、キーおよびケイパビリティーを取得するには、以下を実行します。

```
ceph auth export TYPE.ID
```

例

```
[root@mon ~]# ceph auth get client.admin
```

2. **ceph auth get** に **-o FILE_NAME** オプションを使用して、出力をファイルに保存することもできます。開発者は以下を実行することもできます。

```
ceph auth export TYPE.ID
```

例

```
[root@mon ~]# ceph auth export client.admin
```

auth export コマンドは、**auth get** と同じですが、エンドユーザーとは無関係な内部 **auid** も出力します。

5.3.4. 新しい Ceph ユーザーの追加

ユーザーを追加すると、ユーザー名 (つまり **TYPE.ID**)、シークレットキー、およびユーザーの作成に使用するコマンドに含まれるケイパビリティー) が作成されます。

ユーザーのキーにより、ユーザーは Ceph Storage クラスターとの認証を行うことができます。ユーザーの機能により、Ceph モニター (**mon**)、Ceph OSD (**osd**)、または Ceph Metadata Server (**mds**) での読み取り、書き込み、実行を承認します。

ユーザーを追加する方法はいくつかあります。

- **ceph auth add**: このコマンドは、ユーザーを追加する正規の方法になります。ユーザーを作成し、キーを生成し、指定の機能を追加します。

- **ceph auth get-or-create**: ユーザー名 (括弧内) とキーを持つキーファイルの形式を返すため、このコマンドはユーザーを作成する最も便利な方法です。ユーザーがすでに存在する場合、このコマンドは単にキーファイル形式でユーザー名およびキーを返します。**-o FILE_NAME** オプションを使用して、出力をファイルに保存します。

- **ceph auth get-or-create-key**: このコマンドはユーザーを作成し、ユーザーのキーのみを返す便利な方法です。これは、鍵のみを必要とするクライアント (例: **libvirt**) に役立ちます。ユーザーがすでに存在する場合は、このコマンドが鍵を返すだけです。**-o FILE_NAME** オプションを使用して、出力をファイルに保存します。

クライアントユーザーの作成時に、ケイパビリティーのないユーザーを作成できます。クライアントはモニターからクラスターマップを取得できないため、ケイパビリティーのないユーザーには認証以上のことができません。ただし、後で **ceph auth caps** コマンドを使用してケイパビリティーを追加する場合には、ケイパビリティーがないユーザーを作成することができます。

通常ユーザーは、Ceph OSD における Ceph モニターおよび読み取り/書き込みケイパビリティーにおいて、少なくとも読み取りケイパビリティーを持ちます。また、ユーザーの OSD パーミッションは、多くの場合、特定のプールへのアクセスに制限されます。

```
[root@mon ~]# ceph auth add client.john mon 'allow r' osd 'allow rw pool=liverpool'
[root@mon ~]# ceph auth get-or-create client.paul mon 'allow r' osd 'allow rw pool=liverpool'
[root@mon ~]# ceph auth get-or-create client.george mon 'allow r' osd 'allow rw pool=liverpool' -o
george.keyring
[root@mon ~]# ceph auth get-or-create-key client.ringo mon 'allow r' osd 'allow rw pool=liverpool' -o
ringo.key
```

重要

ユーザーに OSD に対するケイパビリティーを提供する場合に、特定のプールへのアクセスを制限しない場合は、ユーザーはクラスター内のすべてのプールにアクセスできるようになります。

5.3.5. Ceph ユーザーの変更

ceph auth caps コマンドを使用すると、ユーザーを指定してユーザーのケイパビリティーを変更できます。新しい機能を設定すると、現在の機能が上書きされます。したがって、最初に現在の機能を表示し、新しい機能を追加するときにそれらを含めます。

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. 現在の機能を表示します。

```
ceph auth get USERTYPE.USERID
```

例

```
[root@mon ~]# ceph auth get client.john
exported keyring for client.john
[client.john]
 key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
 caps mon = "allow r"
 caps osd = "allow rw pool=liverpool"
```

2. ケイパビリティーを追加するには、以下の形式を使用します。

```
ceph auth caps USERTYPE.USERID DAEMON 'allow [r|w|x|*|...] [pool=POOL_NAME]
[namespace=NAMESPACE_NAME]'
```

例

```
[root@mon ~]# ceph auth caps client.john mon 'allow r' osd 'allow rwx pool=liverpool'
```

この例では、OSD の実行機能が追加されています。

3. 追加された機能を確認します。

```
ceph auth get _USERTYPE_._USERID_
```

例

```
[root@mon ~]# ceph auth get client.john
exported keyring for client.john
[client.john]
```

```
key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
caps mon = "allow r"
caps osd = "allow rwx pool=liverpool"
```

この例では、OSD の実行機能を確認できます。

4. 機能を削除するには、削除する機能を除く現在のすべての機能を設定します。

```
ceph auth caps USERTYPE.USERID DAEMON 'allow [r|w|x|*|...] [pool=POOL_NAME]
[namespace=NAMESPACE_NAME]'
```

例

```
[root@mon ~]# ceph auth caps client.john mon 'allow r' osd 'allow rw pool=liverpool'
```

この例では、OSD の実行機能が含まれていないため、削除されます。

5. 削除された機能を確認します。

```
ceph auth get _USERTYPE_._USERID_
```

例

```
[root@mon ~]# ceph auth get client.john
exported keyring for client.john
[client.john]
 key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
 caps mon = "allow r"
 caps osd = "allow rw pool=liverpool"
```

この例では、OSD の実行機能は記載されていません。

関連情報

- ケイパビリティーの詳細は、**認可ケイパビリティー** を参照してください。

5.3.6. Ceph ユーザーの削除

コマンドラインインターフェイスを使用して、Ceph Storage クラスターからユーザーを削除できます。

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。

手順

1. ユーザーを削除するには、**ceph auth del** を使用します。

```
[root@mon ~]# ceph auth del TYPE.ID
```

ここで、**TYPE** は **クライアント**、**osd**、**mon**、または **mds** のいずれかで、**ID** はデーモンのユーザー名または ID です。

5.3.7. Ceph ユーザーキーの出力

コマンドラインインターフェイスを使用して、Ceph ユーザーのキー情報を表示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。
- ノードへのルートレベルのアクセス。

手順

1. ユーザーの認証キーを標準出力に出力するには、以下を実行します。

```
ceph auth print-key TYPE.ID
```

ここで、**TYPE** は **クライアント**、**osd**、**mon**、または **mds** のいずれかで、**ID** はデーモンのユーザー名または ID です。

2. ユーザーのキーを出力すると、クライアントソフトウェアにユーザーのキー (例] **libvirt**) を設定する必要がある場合に便利です。

```
mount -t ceph HOSTNAME:/MOUNT_POINT -o name=client.user,secret=ceph auth print-key client.user
```

5.3.8. Ceph ユーザーのインポート

コマンドラインインターフェイスを使用して Ceph ユーザーをインポート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。
- ノードへのルートレベルのアクセス。

手順

1. 1 つ以上のユーザーをインポートするには、**ceph auth import** を使用してキーリングを指定します。

```
ceph auth import -i /PATH/TO/KEYRING
```

例

```
[root@mon ~]# ceph auth import -i /etc/ceph/ceph.keyring
```

注記

Ceph Storage クラスターは、新規ユーザー、そのキー、それらのケイパビリティーを追加し、既存のユーザー、それらのキー、およびそのケイパビリティーを更新します。

5.4. CEPH キーリングの管理

ストレージ管理者として、Ceph ユーザーキーの管理は、Red Hat Ceph Storage クラスターにアクセスする際に重要です。キーリングを作成したり、キーリングにユーザーを追加したり、キーリングでユーザーを修正したりすることができます。

5.4.1. 前提条件

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

5.4.2. キーリングの作成

Ceph クライアントにユーザーキーを指定して、Ceph クライアントで、指定したユーザーのキーを取得して Ceph Storage Cluster で認証できるようにする必要があります。Ceph クライアントはキーリングにアクセスしてユーザー名を検索し、ユーザーのキーを取得します。

ceph-authtool ユーティリティーを使用すると、キーリングを作成できます。

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. 空のキーリングを作成するには、**--create-keyring** または **-C** を使用します。

   例

   ```
   [root@mon ~]# ceph-authtool --create-keyring /path/to/keyring
   ```

   複数のユーザーでキーリングを作成する場合は、クラスター名を使用することが推奨されます。例えば、キーリングファイル名に **CLUSTER_NAME.keyring`** を指定し、**/etc/ceph/** ディレクトリーに保存することで、Ceph 設定ファイルのローカルコピーで指定しなくても、**キーリング** 設定のデフォルト設定によりファイル名が指定されます。

2. 以下のコマンドを実行して **ceph.keyring** を作成します。

   ```
   [root@mon ~]# ceph-authtool -C /etc/ceph/ceph.keyring
   ```

1 人のユーザーでキーリングを作成する場合は、クラスター名、ユーザータイプ、およびユーザー名を使用して、**/etc/ceph/** ディレクトリーに保存することが推奨されます。たとえば、**client.admin** ユーザーの場合は **ceph.client.admin.keyring** です。

/etc/ceph/ でキーリングを作成するには、**root** として設定する必要があります。これは、そのファイルには **root** ユーザーのみに **rw** パーミッションが付与されることを意味し、キーリングに管理者キーが含まれている場合にはこれが適切です。ただし、特定のユーザーまたはユーザーグループにキーリングを使用する場合は、**chown** または **chmod** を実行して、適切なキーリングの所有権とアクセスを確立するようにしてください。

5.4.3. キーリングへのユーザーの追加

ユーザーを Ceph Storage クラスターに追加する際に、**get** 手順を使用してユーザー、キー、およびケイパビリティーを取得し、ユーザーをキーリングファイルに保存します。キーリングごとに 1 人のユーザーのみを使用する場合には、**-o** オプションを使用した **Ceph ユーザー情報の表示** 手順により、出力がキーリングファイル形式で保存されます。

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. **client.admin** ユーザーにキーリングを作成するには、以下を実行します。

```
[root@mon ~]# ceph auth get client.admin -o /etc/ceph/ceph.client.admin.keyring
```

   個々のユーザーに推奨されるファイル形式を使用することに注意してください。

2. ユーザーをキーリングにインポートする場合には、**ceph-authtool** を使用して、宛先キーリングとソースキーリングを指定することができます。

```
[root@mon ~]# ceph-authtool /etc/ceph/ceph.keyring --import-keyring
/etc/ceph/ceph.client.admin.keyring
```

5.4.4. キーリングを使用した Ceph ユーザーの作成

Ceph は、Red Hat Ceph Storage クラスターでユーザーを直接作成する機能を提供します。ただし、Ceph クライアントキーリングでユーザー、キー、およびケイパビリティーを直接作成することもできます。

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. ユーザーをキーリングにインポートします。

   例

```
[root@mon ~]# ceph-authtool -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx'
/etc/ceph/ceph.keyring
```

2. キーリングを作成し、キーリングに新規ユーザーを同時に追加します。

   以下に例を示します。

```
[root@mon ~]# ceph-authtool -C /etc/ceph/ceph.keyring -n client.ringo --cap osd 'allow rwx' --
cap mon 'allow rwx' --gen-key
```

前述のシナリオでは、新しいユーザー **client.ringo** はキーリングにのみ使用されます。

3. 新規ユーザーを Ceph Storage クラスターに追加するには、以下を実行します。

```
[root@mon ~]# ceph auth add client.ringo -i /etc/ceph/ceph.keyring
```

関連情報

- ケイパビリティーの詳細は、Ceph ユーザー管理の背景 を参照してください。

5.4.5. キーリングを使用した Ceph ユーザーの変更

コマンドラインインターフェイスを使用して、Ceph ユーザーとそのキーリングを変更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。
- ノードへのルートレベルのアクセス。

手順

1. キーリングでユーザーレコードのケイパビリティーを変更するには、キーリングを指定し、ユーザーがその後のケイパビリティーを指定します。以下に例を示します。

```
[root@mon ~]# ceph-authtool /etc/ceph/ceph.keyring -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx'
```

1. ユーザーを Red Hat Ceph Storage クラスターに更新するには、キーリングのユーザーを Red Hat Ceph Storage クラスターのユーザーエントリーに更新する必要があります。

```
[root@mon ~]# ceph auth import -i /etc/ceph/ceph.keyring
```

また、ストレージクラスターでユーザー機能を直接変更し、結果をキーリングファイルに保存してから、キーリングをメインの **ceph.keyring** ファイルにインポートすることもできます。

関連情報

- キーリングから Red Hat Ceph Storage クラスターユーザーを更新する方法は、**ユーザーの更新** を参照してください。

5.4.6. Ceph ユーザーのコマンドライン使用方法

Ceph では、ユーザー名およびシークレットについて、以下の用途がサポートされます。

--id | **--user**

詳細

Ceph は、種別と ID を持つユーザーを識別します。たとえば、**TYPE.ID** または **client.admin**、**client.user1** です。**id** オプション、**name** オプション、および **-n** オプションを使用すると、ユーザー名の ID 部分を指定できます。たとえば、**admin**、**user1**、**foo** などです。**--id** で

ユーザーを指定し、タイプを省略できます。たとえば、ユーザー **client.foo** を指定するには、次のコマンドを実行します。

```
[root@mon ~]# ceph --id foo --keyring /path/to/keyring health
[root@mon ~]# ceph --user foo --keyring /path/to/keyring health
```

--name | **-n**

詳細

Ceph は、種別と ID を持つユーザーを識別します。たとえば、**TYPE.ID** または **client.admin**、**client.user1** です。**--name** オプションおよび **-n** オプションを使用すると、完全修飾ユーザー名を指定できます。ユーザータイプ (通常は **client**) をユーザー ID で指定する必要があります。以下に例を示します。

```
[root@mon ~]# ceph --name client.foo --keyring /path/to/keyring health
[root@mon ~]# ceph -n client.foo --keyring /path/to/keyring health
```

--keyring

詳細

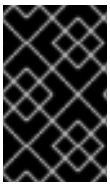
1 つ以上のユーザー名およびシークレットが含まれるキーリングへのパス。**--secret** オプションは、同じ機能を提供しますが、別の目的で **--secret** を使用する Ceph RADOS Gateway では機能しません。**ceph auth get-or-create** でキーリングを取得して、ローカルに保存する場合があります。キーリングパスを切り替えなくてもユーザー名を切り替えることができるため、これは優先されます。以下に例を示します。

```
[root@mon ~]# rbd map foo --pool rbd myimage --id client.foo --keyring /path/to/keyring
```

5.4.7. Ceph ユーザー管理の制限

cephx プロトコルは、Ceph クライアントとサーバーを相互に認証します。これは、人間のユーザーの認証や、ユーザーに代わって実行されるアプリケーションプログラムを扱うことを意図したものではありません。アクセス制御のニーズの処理に効果が必要な場合は、Ceph オブジェクトストアへのアクセスに使用されるフロントエンドに固有の別のメカニズムが必要です。この他のメカニズムは、Ceph がオブジェクトストアへのアクセスを許可するマシン上で、許容されるユーザーとプログラムのみが実行できるようにするロールを持っています。

Ceph クライアントおよびサーバーの認証に使用される鍵は、通常信頼できるホストの適切なパーミッションを持つプレーンテキストファイルに保存されます。



重要

プレーンテキストファイルにキーを保存するにはセキュリティー上の欠陥がありますが、Ceph がバックグラウンドで使用する基本的な認証方法により、その回避は困難です。Ceph システムを構築する方は、これらの欠点を認識しておく必要があります。

特に、任意のユーザーマシン (特にポータブルマシン) は Ceph と直接対話するように設定することはできません。これは、このようなモードにはセキュアでないマシンにプレーンテキスト認証キーのストレージが必要になるためです。そのマシンを盗んだ者や秘密のアクセス権を得た者は、自分のマシンを Ceph に認証させる鍵を手に入れることができます。

潜在的にセキュアでないマシンが Ceph オブジェクトストアに直接アクセスできるようにするのではなく、目的のために十分なセキュリティーを提供する方法を使用して、環境内の信頼済みマシンにログイ

ンする必要があります。信頼されるマシンには、ユーザー用のプレーンテキストの Ceph キーが保存されます。Ceph の今後のバージョンでは、これらの特定の認証に関する問題に対処する場合があります。

現時点では、Ceph 認証プロトコルのいずれの場合でも、転送中のメッセージの機密性は提供されていません。このように、ネットワーク上の盗聴者は、たとえ作成や変更ができなくても、Ceph 内のクライアントとサーバーとの間で送信されるすべてのデータを聞いて理解することができます。Ceph に機密データを保存する場合には、Ceph システムにデータを提供する前に、データを暗号化する必要があります。

例えば、Ceph Object Gateway は S3 API サーバー側暗号化 を提供しており、Ceph Object Gateway クライアントから受け取った未暗号化のデータを暗号化してから Ceph Storage クラスターに保存し、同様に Ceph Storage クラスターから取得したデータを復号してからクライアントに送信します。クライアントと Ceph Object Gateway 間の移行で暗号化を確実に行うために、Ceph Object Gateway は SSL を使用するように設定する必要があります。

第6章 CEPH-VOLUME ユーティリティー

ストレージ管理者は、**ceph-volume** ユーティリティーを使用して Ceph OSD を準備、作成、およびアクティベートすることができます。**ceph-volume** ユーティリティーは、論理ボリュームを OSD としてデプロイするための単一の目的コマンドラインツールです。プラグインタイプのフレームワークを使用して、異なるデバイス技術を持つ OSD をデプロイします。**ceph-volume** ユーティリティーは、OSD のデプロイに使用する **ceph-disk** ユーティリティーと同様のワークフローに従います。これは、OSD の準備、アクティブ化、および起動を可能にする予測可能で堅牢な方法です。現在、**ceph-volume** ユーティリティーは **lvm** プラグインのみをサポートします。また、今後、その他のテクノロジーをサポートする予定があります。

重要

ceph-disk コマンドは非推奨となりました。

6.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

6.2. CEPH ボリュームの LVM プラグイン

LVM タ グを利用することで、**lvm** サブコマンドは OSD に関連付けられたデバイスを照会して保存し、再検出できるため、それらをアクティブにすることができます。これには、**dm-cache** などの lvm ベースのテクノロジーもサポートします。

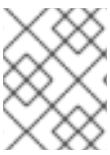
ceph-volume を使用する場合は、**dm-cache** の使用は透過的になり、**dm-cache** は論理ボリュームのように処理されます。**dm-cache** を使用した場合のパフォーマンスの損益は、特定のワークロードに依存します。一般的に、ランダム読み取りおよび順次読み取りにより、ブロックサイズが小さいとパフォーマンスが向上することになります。ランダムな書き込みや順次書き込みでは、ブロックサイズが大きくなるとパフォーマンスが低下します。

LVM プラグインを使用するには、**lvm** をサブコマンドとして **ceph-volume** コマンドに追加します。

```
[root@osd ~]# ceph-volume lvm
```

以下のように **lvm** サブコマンドには 3 つのサブコマンドがあります。

- **prepare**
- **activate**
- **create**
- **batch**



注記

create サブコマンドを使用すると、**prepare** および **activate** サブコマンドが 1 つのサブコマンドに統合されます。

関連情報

- 詳細は、**create** サブコマンドの セクション を参照してください。

6.3. CEPH-VOLUME が CEPH-DISK の代替になる理由

以前のバージョンの Red Hat Ceph Storage では、**ceph-disk** ユーティリティーを使用して OSD を準備、アクティブ化、および作成していました。Red Hat Ceph Storage 4 以降では、**ceph-disk** が **ceph-volume** ユーティリティーに置き換えられました。これは、OSD として論理ボリュームをデプロイする 1 つの目的のコマンドラインツールとなることを目的としています。一方、OSD の準備、アクティブ化、および作成時に同様の API を **ceph-disk** に維持します。

ceph-volume はどのように機能しますか。

ceph-volume は、ハードウェアデバイスのプロビジョニングに関して、現在、レガシーの **ceph-disk** デバイスと LVM (Logical Volume Manager) デバイスの 2 つの方法に対応しているモジュラーツールです。**ceph-volume lvm** コマンドは、LVM タグを使用して、Ceph 固有のデバイスと、OSD との関係に関する情報を保存します。これらのタグを使用して、OSDS に関連付けられたデバイスを後で再検出し、クエリーし、それらをアクティベートできるようにします。LVM および **dm-cache** に基づく技術にも対応しています。

ceph-volume ユーティリティーは **dm-cache** を透過的に使用し、論理ボリュームとして処理します。処理する特定のワークロードに応じて、**dm-cache** を使用した場合のパフォーマンスの損益を考慮するとよいでしょう。一般的に、ブロックサイズが小さくなるとランダムおよび順次読み出し操作のパフォーマンスが向上し、ブロックサイズが大きくなるとランダムおよび連続書き込み操作のパフォーマンスが低下します。**ceph-volume** を使用しても、パフォーマンスが大幅に低下することはありません。

重要

ceph-disk ユーティリティーは非推奨になりました。

注記

ceph-volume simple コマンドは、レガシーの **ceph-disk** デバイスが使用されている場合には、そのデバイスを処理することができます。

ceph-disk はどのように機能しますか。

ceph-disk ユーティリティーは、**upstart** や **sysvinit** のような異なるタイプの init システムを多数サポートしながら、デバイスを検出することができるようにする必要がありました。このため、**ceph-disk** は GUID パーティションテーブル (GPT) パーティションにのみ集中します。具体的には、デバイスを独自の方法でラベル付けする GPT GUID で、次のような質問に答えます。

- このデバイスは **ジャーナル** ですか。
- このデバイスは暗号化されたデータパーティションですか。
- デバイスが部分的に準備されましたか。

これらの質問を解決するために、**ceph-disk** は UDEV ルールを使用して GUID に一致させます。

ceph-disk を使用するデメリットはなんですか。

UDEV ルールを使用して **ceph-disk** を呼び出すと、**ceph-disk systemd** ユニットと **ceph-disk** 実行ファイルの間に行き来が発生することがあります。このプロセスは信頼性が非常に低く、時間がかかるため、ノードのブートプロセス中に OSD が全く起動しなくなることがあります。さらに、UDEV の非同期動作により、これらの問題をデバッグしたり複製することもすることは困難です。

ceph-disk は GPT パーティションと排他的に機能するため、論理ボリュームマネージャー (LVM) ボリュームや同様のデバイスマッパーデバイスなどの他のテクノロジーには対応できません。

GPT パーティションがデバイス検出ワークフローで正しく機能するようにするには、**ceph-disk** で多数の特別なフラグを使用する必要があります。また、これらのパーティションには、デバイスを Ceph が排他的に所有する必要があります。

6.4. CEPH-VOLUME を使用した CEPH OSD の準備

prepare サブコマンドは、OSD バックエンドのオブジェクトストアを準備し、OSD データとジャーナルの両方に論理ボリューム (LV) を消費します。これは、LVM を使用して追加のメタデータタグを追加する以外は、論理ボリュームを変更しません。これらのタグはボリュームの検出を容易にし、Ceph ストレージクラスターの一部としてのボリュームとストレージクラスター内でのボリュームのロールを識別します。

BlueStore OSD バックエンドでは、以下の設定がサポートされます。

- ブロックデバイス、**block.wal** デバイス、および **block.db** デバイス
- ブロックデバイスと **block.wal** デバイス
- ブロックデバイスと **block.db** デバイス
- 1 つのブロックデバイス

prepare サブコマンドは、全デバイスまたはパーティション、または **ブロック** の論理ボリュームを受け入れます。

前提条件

- OSD ノードへのルートレベルのアクセス。
- 必要に応じて、論理ボリュームを作成します。物理デバイスへのパスを指定すると、サブコマンドはデバイスを論理ボリュームに変換します。このアプローチはシンプルですが、論理ボリュームの作成方法を設定したり、変更したりすることはできません。

手順

1. LVM ボリュームを準備します。

   構文

   ```
   ceph-volume lvm prepare --bluestore --data VOLUME_GROUP/LOGICAL_VOLUME
   ```

   例

   ```
   [root@osd ~]# ceph-volume lvm prepare --bluestore --data example_vg/data_lv
   ```

   a. 必要に応じて、RocksDB 用に別のデバイスを使用する場合は、**--block.db** オプションおよび **--block.wal** オプションを指定します。

      構文

```
ceph-volume lvm prepare --bluestore --block.db --block.wal --data
VOLUME_GROUP/LOGICAL_VOLUME
```

例

```
[root@osd ~]# ceph-volume lvm prepare --bluestore --block.db --block.wal --data
example_vg/data_lv
```

b. 必要に応じて、データを暗号化するには、**--dmcrypt** フラグを使用します。

構文

```
ceph-volume lvm prepare --bluestore --dmcrypt --data
VOLUME_GROUP/LOGICAL_VOLUME
```

例

```
[root@osd ~]# ceph-volume lvm prepare --bluestore --dmcrypt --data
example_vg/data_lv
```

関連情報

- 詳細は、**Red Hat Ceph Storage 管理ガイド**の `ceph-volume` を使用した Ceph OSD 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 管理ガイド**の `ceph-volume` を使用した Ceph OSD の作成セクションを参照してください。

6.5. CEPH-VOLUME を使用した CEPH OSD のアクティブ化

アクティベーションプロセスにより、システムの起動時に **systemd** ユニットが有効になり、正しい OSD 識別子とその UUID が有効になり、マウントされます。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- Ceph OSD ノードへのルートレベルのアクセス。
- **ceph-volume** ユーティリティーが準備する Ceph OSD。

手順

1. OSD ノードから OSD ID および UUID を取得します。

```
[root@osd ~]# ceph-volume lvm list
```

2. OSD をアクティベートします。

構文

```
ceph-volume lvm activate --bluestore OSD_ID OSD_UUID
```

–

例

```
[root@osd ~]# ceph-volume lvm activate --bluestore 0 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8
```

アクティブ化用に準備されているすべての OSD を有効にするには、**--all** オプションを使用します。

例

```
[root@osd ~]# ceph-volume lvm activate --all
```

関連情報

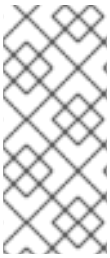
- 詳細は**Red Hat Ceph Storage 管理ガイド**の `ceph-volume` を使用した Ceph OSD の準備セクションを参照してください。
- 詳細は**Red Hat Ceph Storage 管理ガイド**の `ceph-volume` を使用した Ceph OSD の作成セクションを参照してください。

6.6. CEPH-VOLUME を使用した CEPH OSD の作成

create サブコマンドは **prepare** サブコマンドを呼び出し、**activate** サブコマンドを呼び出します。

前提条件

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



注記

作成プロセスに対する制御を強化する場合は、サブコマンドの **prepare** および **activate** を個別に使用して、**create** を使用する代わりに OSD を作成できます。この 2 つのサブコマンドを使用すると、大量のデータをリバランスせずに、新規 OSD をストレージクラスターに段階的に導入することができます。**create** サブコマンドを使用すると、完了直後に OSD が **up** および **in** になりますが、どちらのアプローチも同じように機能します。

手順

1. 新規 OSD を作成するには、以下を実行します。

構文

```
ceph-volume lvm create --bluestore --data VOLUME_GROUP/LOGICAL_VOLUME
```

例

```
[root@osd ~]# ceph-volume lvm create --bluestore --data example_vg/data_lv
```

関連情報

- 詳細はRed Hat Ceph Storage 管理ガイドの `ceph-volume` を使用した Ceph OSD の準備セクションを参照してください。
- 詳細は、Red Hat Ceph Storage 管理ガイドの `ceph-volume` を使用した Ceph OSD の有効化 セクションを参照してください。

6.7. CEPH-VOLUME でのバッチモードの使用

batch サブコマンドは、単一デバイスが提供されると複数の OSD の作成を自動化します。

ceph-volume コマンドは、ドライブタイプに基づいて OSD の作成に使用する最適な方法を決定します。Ceph OSD の最適化は、利用可能なデバイスによって異なります。

- すべてのデバイスが従来のハードドライブの場合、**batch** はデバイスごとに OSD を 1 つ作成します。
- すべてのデバイスがソリッドステートドライブの場合は、**バッチ** によりデバイスごとに OSD が 2 つ作成されます。
- 従来のハードドライブとソリッドステートドライブが混在している場合、**バッチ** はデータに従来のハードドライブを使用し、ソリッドステートドライブに可能な限り大きいジャーナル (**block.db**) を作成します。

注記

batch サブコマンドは、write-ahead-log (**block.wal**) デバイスに別の論理ボリュームを作成すること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。
- Ceph OSD ノードへのルートレベルのアクセス。

手順

1. 複数のドライブに OSD を作成するには、以下の手順を実行します。

   構文

   ```
   ceph-volume lvm batch --bluestore PATH_TO_DEVICE [PATH_TO_DEVICE]
   ```

   例

   ```
   [root@osd ~]# ceph-volume lvm batch --bluestore /dev/sda /dev/sdb /dev/nvme0n1
   ```

関連情報

- 詳細は、Red Hat Ceph Storage 管理ガイドの `ceph-volume` を使用した Ceph OSD の作成セクションを参照してください。

第7章 CEPH パフォーマンスベンチマーク

ストレージ管理者は、Red Hat Ceph Storage クラスターのパフォーマンスをベンチマークできます。本セクションの目的は、Ceph 管理者が Ceph のネイティブベンチマークツールの基本を理解することを目的としています。これらのツールにより、Ceph Storage クラスターの実行方法についての洞察が提供されます。これは、Ceph パフォーマンスベンチマークの最終ガイドではなく、Ceph を適宜調整する方法に関するガイドです。

7.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

7.2. パフォーマンスベースライン

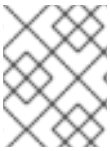
ジャーナル、ディスク、ネットワークのスループットを含む OSD には、比較すべきパフォーマンスベースラインがあるはずです。ベースラインのパフォーマンスデータと Ceph のネイティブツールのデータを比較することで、潜在的なチューニング効果を特定することができます。Red Hat Enterprise Linux には、これらのタスクを実現するために利用可能なオープンソースコミュニティーツールが複数含まれています。

関連情報

- 利用可能なツールの詳細は、ナレッジベースアーティクル Red Hat Enterprise Linux で利用できるベンチマークツールおよびパフォーマンステストツールはありますか？を参照してください。

7.3. CEPH パフォーマンスのベンチマーク

Ceph には、RADOS ストレージクラスターでパフォーマンスベンチマークを行う **rados bench** コマンドが含まれます。このコマンドは、書き込みテストと 2 種類の読み取りテストを実行します。**--no-cleanup** オプションは、読み取りおよび書き込みパフォーマンスの両方をテストする際に使用することが重要です。デフォルトでは、**rados bench** コマンドは、ストレージプールに書き込まれたオブジェクトを削除します。これらのオブジェクトをそのまま残すと、2 つの読み取りテストで、順次読み取りパフォーマンスとランダムな読み取りパフォーマンスを測定できます。



注記

これらのパフォーマンステストを実行する前に、次のコマンドを実行してファイルシステムのキャッシュをすべて破棄します。

```
[root@mon~ ]# echo 3 | sudo tee /proc/sys/vm/drop_caches && sudo sync
```

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. 新しいストレージプールを作成します。

```
[root@osd~ ]# ceph osd pool create testbench 100 100
```

2. 新規作成されたストレージプールへの書き込みテストを 10 秒実行します。

```
[root@osd~ ]# rados bench -p testbench 10 write --no-cleanup
```

出力例

```
Maintaining 16 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_cephn1.home.network_10510
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
     0       0         0         0         0         0         -         0
     1      16        16         0         0         0         -         0
     2      16        16         0         0         0         -         0
     3      16        16         0         0         0         -         0
     4      16        17         1  0.998879         1   3.19824   3.19824
     5      16        18         2   1.59849         4   4.56163   3.87993
     6      16        18         2   1.33222         0         -   3.87993
     7      16        19         3   1.71239         2   6.90712     4.889
     8      16        25         9   4.49551        24   7.75362   6.71216
     9      16        25         9   3.99636         0         -   6.71216
    10      16        27        11   4.39632         4   9.65085   7.18999
    11      16        27        11   3.99685         0         -   7.18999
    12      16        27        11   3.66397         0         -   7.18999
    13      16        28        12   3.68975   1.33333   12.8124   7.65853
    14      16        28        12   3.42617         0         -   7.65853
    15      16        28        12   3.19785         0         -   7.65853
    16      11        28        17   4.24726   6.66667   12.5302   9.27548
    17      11        28        17   3.99751         0         -   9.27548
    18      11        28        17   3.77546         0         -   9.27548
    19      11        28        17   3.57683         0         -   9.27548
 Total time run:         19.505620
Total writes made:      28
Write size:             4194304
Bandwidth (MB/sec):     5.742

Stddev Bandwidth:       5.4617
Max bandwidth (MB/sec): 24
Min bandwidth (MB/sec): 0
Average Latency:        10.4064
Stddev Latency:         3.80038
Max latency:            19.503
Min latency:            3.19824
```

3. ストレージプールへの 10 秒間の順次読み取りテストを実行します。

```
[root@osd~ ]## rados bench -p testbench 10 seq
```

出力例

```
sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
  0       0         0         0         0         0         -         0
Total time run:        0.804869
Total reads made:      28
```

```
Read size:              4194304
Bandwidth (MB/sec):     139.153

Average Latency:        0.420841
Max latency:            0.706133
Min latency:            0.0816332
```

4. ストレージプールに対して、10 秒間ランダムな読み取りテストを実行します。

```
[root@osd ~]# rados bench -p testbench 10 rand
```

出力例

```
sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
  0       0         0         0         0         0         -         0
  1      16        46        30   119.801       120  0.440184  0.388125
  2      16        81        65   129.408       140  0.577359  0.417461
  3      16       120       104   138.175       156  0.597435  0.409318
  4      15       157       142   141.485       152  0.683111  0.419964
  5      16       206       190   151.553       192  0.310578  0.408343
  6      16       253       237   157.608       188 0.0745175  0.387207
  7      16       287       271   154.412       136  0.792774   0.39043
  8      16       325       309   154.044       152  0.314254   0.39876
  9      16       362       346   153.245       148  0.355576  0.406032
 10      16       405       389   155.092       172   0.64734  0.398372
Total time run:         10.302229
Total reads made:       405
Read size:              4194304
Bandwidth (MB/sec):     157.248

Average Latency:        0.405976
Max latency:            1.00869
Min latency:            0.0378431
```

5. 同時の読み書き数を増やすには、**-t** オプションを使用します (デフォルトは 16 スレッド)。また、**-b** パラメーターは、書き込まれているオブジェクトのサイズを調整することもできます。デフォルトのオブジェクトサイズは 4 MB です。安全な最大オブジェクトサイズは 16 MB です。Red Hat は、このベンチマークテストの複数のコピーを異なるプールで実行することを推奨します。これにより、複数のクライアントのパフォーマンスが変更になりました。**--run-name <label>** オプションを追加して、ベンチマークテスト中に作成するオブジェクトの名前を制御します。実行中の各コマンドインスタンスの **--run-name** ラベルを変更すると、複数の **rados bench** コマンドを同時に実行できます。これにより、複数のクライアントが同じオブジェクトにアクセスしようとし、異なるクライアントが異なるオブジェクトにアクセスしようとすると発生する可能性のある I/O エラーを防ぐことができます。**--run-name** オプションは、実世界のワークロードをシミュレートしようとしているときにも便利です。以下に例を示します。

```
[root@osd ~]# rados bench -p testbench 10 write -t 4 --run-name client1
```

出力例

```
Maintaining 4 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_node1_12631
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
```

```
    0      0       0       0        0        0       -        0
    1      4       4       0        0        0       -        0
    2      4       6       2  3.99099        4  1.94755  1.93361
    3      4       8       4  5.32498        8    2.978  2.44034
    4      4       8       4  3.99504        0        -  2.44034
    5      4      10       6  4.79504        4  2.92419   2.4629
    6      3      10       7  4.64471        4  3.02498   2.5432
    7      4      12       8  4.55287        4  3.12204  2.61555
    8      4      14      10   4.9821        8  2.55901  2.68396
    9      4      16      12  5.31621        8  2.68769  2.68081
   10      4      17      13  5.18488        4  2.11937  2.63763
   11      4      17      13  4.71431        0        -  2.63763
   12      4      18      14  4.65486        2   2.4836  2.62662
   13      4      18      14  4.29757        0        -  2.62662
Total time run:         13.123548
Total writes made:      18
Write size:             4194304
Bandwidth (MB/sec):     5.486

Stddev Bandwidth:       3.0991
Max bandwidth (MB/sec): 8
Min bandwidth (MB/sec): 0
Average Latency:        2.91578
Stddev Latency:         0.956993
Max latency:            5.72685
Min latency:            1.91967
```

6. **rados bench** コマンドで作成したデータを削除します。

```
[root@osd ~]# rados -p testbench cleanup
```

7.4. CEPH ブロックパフォーマンスのベンチマーク

Ceph には、ブロックデバイスへの順次書き込みをテストする **rbd bench-write** コマンドが含まれます。これは、スループットとレイテンシーの測定を行います。デフォルトのバイトサイズは 4096 で、デフォルトの I/O スレッド数は 16 で、書き込みするデフォルトのバイト数は 1 GB です。これらのデフォルトは、それぞれ **--io-size** オプション、**--io-threads** オプション、および **--io-total** オプションで変更できます。

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. **rbd** カーネルモジュールを読み込んでいない場合は読み込みます。

```
[root@mon ~]# modprobe rbd
```

2. **testbench** プールに 1 GB の **rbd** イメージファイルを作成します。

```
[root@mon ~]# rbd create image01 --size 1024 --pool testbench
```

3. イメージファイルをデバイスファイルにマッピングします。

```
[root@mon ~]# rbd map image01 --pool testbench --name client.admin
```

4. ブロックデバイスに **ext4** ファイルシステムを作成します。

```
[root@mon ~]# mkfs.ext4 /dev/rbd/testbench/image01
```

5. 新しいディレクトリーを作成します。

```
[root@mon ~]# mkdir /mnt/ceph-block-device
```

6. ブロックデバイスを **/mnt/ceph-block-device/** にマウントします。

```
[root@mon ~]# mount /dev/rbd/testbench/image01 /mnt/ceph-block-device
```

7. ブロックデバイスに対して書き込みパフォーマンスのテストを実行します。

```
[root@mon ~]# rbd bench --io-type write image01 --pool=testbench
```

例

```
bench-write  io_size 4096 io_threads 16 bytes 1073741824 pattern seq
  SEC       OPS   OPS/SEC   BYTES/SEC
    2     11127   5479.59  22444382.79
    3     11692   3901.91  15982220.33
    4     12372   2953.34  12096895.42
    5     12580   2300.05  9421008.60
    6     13141   2101.80  8608975.15
    7     13195    356.07  1458459.94
    8     13820    390.35  1598876.60
    9     14124    325.46  1333066.62
    ..
```

関連情報

- **rbd** コマンドに関する詳しい情報は、**Red Hat Ceph Storage ブロックデバイスガイド**の ブロックデバイスコマンド セクションを参照してください。

第8章 CEPH パフォーマンスカウンター

ストレージ管理者は、Red Hat Ceph Storage クラスターのパフォーマンスメトリックを収集できます。Ceph パフォーマンスカウンターは、内部インフラストラクチャーメトリックのコレクションです。このメトリックデータの収集、集計、およびグラフ化は、さまざまなツールで実行でき、パフォーマンス分析に役立ちます。

8.1. 前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。

8.2. CEPH パフォーマンスカウンターへのアクセス

パフォーマンスカウンターは、Ceph Monitor および OSD のソケットインターフェイスを介して利用できます。各デーモンのソケットファイルは、デフォルトでは **/var/run/ceph** の下にあります。パフォーマンスカウンターは、コレクション名にグループ化されます。これらのコレクション名はサブシステムまたはサブシステムのインスタンスを表します。

以下は、Monitor および OSD コレクション名のカテゴリーの一覧です。それぞれの簡単な説明を以下に示します。

コレクション名カテゴリーの監視

- Cluster Metrics - ストレージクラスターに関する情報を表示します (モニター、OSD、プール、PG)。
- Level Database Metrics - バックエンドの **KeyValueStore** データベースに関する情報を表示します。
- Monitor Metrics - 一般的なモニター情報を表示します。
- Paxos Metrics - クラスタークォーラム管理に関する情報を表示します。
- Throttle Metrics - モニターのスロットリング方法の統計を表示します。

OSD コレクションの名前カテゴリー

- Write Back Throttle Metrics - 書き込みバックスロットルがフラッシュされていない IO を追跡する方法についての統計を表示します。
- Level Database Metrics - バックエンドの **KeyValueStore** データベースに関する情報を表示します。
- Objecter Metrics - さまざまなオブジェクトベースの操作に関する情報を表示します。
- Read and Write Operations Metrics - さまざまな読み取りおよび書き込み操作に関する情報を表示します。
- Recovery State Metrics - さまざまなリカバリーの状態のレイテンシーを表示します。
- OSD Throttle Metrics - OSD のスロットリング方法の統計の表示

RADOS ゲートウェイコレクションの名前カテゴリー

- Object Gateway Client Metrics - GET 要求および PUT 要求の統計を表示します。

- Objecter Metrics - さまざまなオブジェクトベースの操作に関する情報を表示します。
- Object Gateway Throttle Metrics: OSD のスロットリングに関する統計の表示

8.3. CEPH パフォーマンスカウンターを表示します。

ceph daemon .. perf schema コマンドは、利用可能なメトリックを出力します。各メトリックには、関連付けられたビットフィールド値タイプがあります。

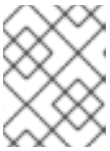
前提条件

- 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. メトリックのスキーマを表示するには、以下を実行します。

```
ceph daemon DAEMON_NAME perf schema
```



注記

デーモンを実行するノードから **ceph daemon** コマンドを実行する必要があります。

2. Monitor ノードから **ceph daemon .. perf schema** コマンドを実行するには、以下を実行します。

```
[root@mon ~]# ceph daemon mon.`hostname -s` perf schema
```

例

```
{
    "cluster": {
        "num_mon": {
            "type": 2
        },
        "num_mon_quorum": {
            "type": 2
        },
        "num_osd": {
            "type": 2
        },
        "num_osd_up": {
            "type": 2
        },
        "num_osd_in": {
            "type": 2
        },
...
```

3. OSD ノードから **ceph daemon .. perf schema** コマンドを実行するには、以下を実行します。

```
[root@mon ~]# ceph daemon osd.0 perf schema
```

例

```
...
"filestore": {
        "journal_queue_max_ops": {
            "type": 2
        },
        "journal_queue_ops": {
            "type": 2
        },
        "journal_ops": {
            "type": 10
        },
        "journal_queue_max_bytes": {
            "type": 2
        },
        "journal_queue_bytes": {
            "type": 2
        },
        "journal_bytes": {
            "type": 10
        },
        "journal_latency": {
            "type": 5
        },
...
```

表8.1 ビットフィールド値の定義

ビット	意味
1	浮動小数点数
2	署名されていない 64 ビットの整数値
4	平均 (合計 + カウント)
8	カウンター

各値には、型 (浮動小数点または整数値) を示すビット 1 または 2 が設定されます。ビット 4 が設定されている場合、読み取る値は合計とカウントの 2 つになります。ビット 8 が設定されている場合、以前の間隔の平均は、以前の読み取り以降、合計差分になります。これは、カウントデルタで除算されます。値を除算すると、有効期間の平均値が提供されることになります。通常、これらはレイテンシー、リクエスト数、およびリクエストレイテンシーの合計を測定するために使用されます。ビットの値は組み合わせられます (例: 5、6、10)。ビット値 5 は、ビット 1 とビット 4 の組み合わせです。つまり、平均は浮動小数点の値になります。ビット値 6 は、ビット 2 とビット 4 の組み合わせです。これは、平均値が整数になることを意味します。ビット値 10 は、ビット 2 とビット 8 の組み合わせです。これは、カウンター値が整数値であることを意味します。

関連情報

- 詳細は、**平均数と合計** を参照してください。

8.4. CEPH パフォーマンスカウンターのダンプ

ceph daemon .. perf dump コマンドは、現在の値を出力し、各サブシステムのコレクション名でメトリックをグループ化します。

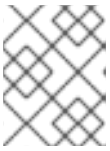
前提条件

- 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. 現在のメトリクスデータを表示するには、以下を実行します。

```
# ceph daemon DAEMON_NAME perf dump
```



注記

デーモンを実行するノードから **ceph daemon** コマンドを実行する必要があります。

2. Monitor ノードから **ceph daemon .. perf dump** コマンドを実行するには、以下のコマンドを実行します。

```
# ceph daemon mon.`hostname -s` perf dump
```

例

```
{
    "cluster": {
        "num_mon": 1,
        "num_mon_quorum": 1,
        "num_osd": 2,
        "num_osd_up": 2,
        "num_osd_in": 2,
...
```

3. OSD ノードから **ceph daemon .. perf dump** コマンドを実行するには、以下のコマンドを実行します。

```
# ceph daemon osd.0 perf dump
```

例

```
...
"filestore": {
        "journal_queue_max_ops": 300,
        "journal_queue_ops": 0,
        "journal_ops": 992,
```

```
        "journal_queue_max_bytes": 33554432,
        "journal_queue_bytes": 0,
        "journal_bytes": 934537,
        "journal_latency": {
            "avgcount": 992,
            "sum": 254.975925772
        },
...
```

関連情報

- 利用可能な各 Monitor メトリックの簡単な説明を表示するには、Ceph Monitor メトリックテーブル を参照してください。

8.5. 平均数と合計

すべてのレイテンシー番号は、ビットフィールドの値は 5 です。このフィールドには、平均数と合計の浮動小数点値が含まれます。**avgcount** は、この範囲内の操作数で、**sum** はレイテンシーの合計 (秒単位) です。**sum** を **avgcount** で除算すると、操作ごとのレイテンシーを把握することができます。

関連情報

- 利用可能な各 OSD メトリックの簡単な説明を表示するには、Ceph OSD テーブル を参照してください。

8.6. CEPH MONITOR メトリック

表8.2 クラスターメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
cluster	**num_mon**	2	モニター数
	num_mon_quorum	2	クォーラムのモニター数
	num_osd	2	OSD の合計数
	num_osd_up	2	稼働中の OSD 数
	num_osd_in	2	クラスターにある OSD 数
	osd_epoch	2	OSD マップの現在のエポック
	osd_bytes	2	クラスターの合計容量 (バイト単位)
	osd_bytes_used	2	クラスターで使用されているバイト数

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	osd_bytes_avail	2	クラスターで利用可能なバイト数
	num_pool	2	プールの数
	num_pg	2	配置グループの合計数
	num_pg_active_clean	2	active+clean 状態の配置グループの数
	num_pg_active	2	アクティブな状態の配置グループの数
	num_pg_peering	2	ピア状態の配置グループの数
	num_object	2	クラスター上のオブジェクトの合計数
	num_object_degraded	2	パフォーマンス低下 (レプリカが欠落している) オブジェクトの数
	num_object_misplaced	2	配置が間違っているオブジェクトの数 (クラスター内の間違った場所)
	num_object_unfound	2	不明なオブジェクトの数
	num_bytes	2	すべてのオブジェクトの合計バイト数
	num_mds_up	2	稼働している MDS の数
	num_mds_in	2	クラスターにある MDS の数
	num_mds_failed	2	失敗した MDS の数
	mds_epoch	2	MDS マップの現在のエポック

表8.3 レベルのデータベースメトリクステーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
leveldb	**leveldb_get**	10	取得

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	leveldb_transaction	10	トランザクション
	leveldb_compact	10	補完
	leveldb_compact_range	10	範囲ごとの比較
	leveldb_compact_queue_merge	10	圧縮キューにおける範囲のマージ
	leveldb_compact_queue_len	2	圧縮キューの長さ

表8.4 一般的なモニターメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
mon	**num_sessions**	2	現在開いているモニターセッションの数
	session_add	10	作成されたモニターセッションの数
	session_rm	10	モニターにおける remove_session 呼び出しの数
	session_trim	10	トリミングされたモニターセッションの数
	num_elections	10	参加する選択モニターの数
	election_call	10	モニターにより開始した選択の数
	election_win	10	モニターが勝利した選択の数
	election_lose	10	モニターにより失われた選択の数

表8.5 Paxos Metrics テーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
paxos	**start_leader**	10	リーダーロールで始まります。

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	start_peon	10	peon ロールで開始します。
	restart	10	再起動
	refresh	10	更新
	refresh_latency	5	更新の待ち時間
	begin	10	開始および処理開始
	begin_keys	6	開始時のトランザクションのキー
	begin_bytes	6	トランザクションの開始時にデータ
	begin_latency	5	開始操作のレイテンシー
	commit	10	コミット
	commit_keys	6	コミット時にトランザクションのキー
	commit_bytes	6	コミット時のトランザクションのデータ
	commit_latency	5	コミットレイテンシー
	collect	10	Peon が収集する
	collect_keys	6	peon collect 上のトランザクションのキー
	collect_bytes	6	peon collect 上のトランザクションのデータ
	collect_latency	5	Peon がレイテンシーを収集
	collect_uncommitted	10	開始および処理された収集のコミットされていない値
	collect_timeout	10	タイムアウトの収集
	accept_timeout	10	タイムアウトの受け入れ
	lease_ack_timeout	10	リースの確認タイムアウト

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	lease_timeout	10	リースタイムアウト
	store_state	10	共有状態をディスクに保存
	store_state_keys	6	保存された状態のトランザクションのキー
	store_state_bytes	6	保存された状態のトランザクションのデータ
	store_state_latency	5	状態レイテンシーの保存
	share_state	10	状態の共有
	share_state_keys	6	共有状態のキー
	share_state_bytes	6	共有状態のデータ
	new_pn	10	新しい提案番号のクエリー
	new_pn_latency	5	レイテンシーを取得する新しい提案番号

表8.6 スロットルメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
throttle-*	**val**	10	現在利用できるスロットル
	max	10	スロットルの最大値
	get	10	取得
	get_sum	10	取得したデータ
	get_or_fail_fail	10	get_or_fail 時にブロックされる
	get_or_fail_success	10	get_or_fail 時の get 成功
	take	10	取得
	take_sum	10	取得したデータ
	put	10	送る

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	put_sum	10	データを送る
	wait	5	待機レイテンシー

関連情報

- クラスターメトリックテーブル
- レベルのデータベースメトリックテーブル
- 一般的なモニターメトリクステーブル
- Paxos Metrics テーブル
- スロットルメトリックテーブル

8.7. CEPH OSD メトリック

表8.7 ライトバックスロットルメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
WBThrottle	**bytes_dirtied**	2	ダーティーデータ
	bytes_wb	2	書き込まれたデータ
	ios_dirtied	2	ダーティー操作
	ios_wb	2	書き込みされた操作
	inodes_dirtied	2	書き込みを待っているエントリー
	inodes_wb	2	書き込まれたエントリー

表8.8 レベルのデータベースメトリクステーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
leveldb	**leveldb_get**	10	取得
	leveldb_transaction	10	トランザクション
	leveldb_compact	10	補完

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	leveldb_compact_range	10	範囲ごとの比較
	leveldb_compact_queue_merge	10	圧縮キューにおける範囲のマージ
	leveldb_compact_queue_len	2	圧縮キューの長さ

表8.9 Objecter Metrics テーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
objecter	**op_active**	2	アクティブな操作
	op_laggy	2	遅延操作
	op_send	10	送信された操作
	op_send_bytes	10	送信されたデータ
	op_resend	10	再送信捜査
	op_ack	10	コールバックのコミット
	op_commit	10	操作のコミット
	op	10	操作
	op_r	10	読み取り操作
	op_w	10	書き込み操作
	op_rmw	10	read-modify-write 操作
	op_pg	10	PG 操作
	osdop_stat	10	統計操作
	osdop_create	10	オブジェクト操作の作成
	osdop_read	10	読み取り操作
	osdop_write	10	書き込み操作

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	osdop_writefull	10	完全なオブジェクト操作の書き込み
	osdop_append	10	追加操作
	osdop_zero	10	オブジェクトをゼロ操作に設定
	osdop_truncate	10	オブジェクト操作の切り捨て
	osdop_delete	10	オブジェクト操作の削除
	osdop_mapext	10	エクステント操作のマップ
	osdop_sparse_read	10	スパース読み取り操作
	osdop_clonerange	10	範囲のクローン操作
	osdop_getxattr	10	xattr 操作の取得
	osdop_setxattr	10	xattr 操作の設定
	osdop_cmpxattr	10	xattr の比較操作
	osdop_rmxattr	10	xattr 操作の削除
	osdop_resetxattrs	10	xattr 操作のリセット
	osdop_tmap_up	10	TMAP 更新操作
	osdop_tmap_put	10	TMAP の put 操作
	osdop_tmap_get	10	TMAP の get 操作
	osdop_call	10	操作の呼び出し (実行)
	osdop_watch	10	オブジェクト操作による監視
	osdop_notify	10	オブジェクト操作に関する通知
	osdop_src_cmpxattr	10	複数演算における拡張属性比較
	osdop_other	10	その他の操作
	linger_active	2	アクティブな linger 操作

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	linger_send	10	送信された linger 操作
	linger_resend	10	再送信された linger 操作
	linger_ping	10	linger 操作に ping が送信
	poolop_active	2	アクティブなプール操作
	poolop_send	10	送信したプール操作
	poolop_resend	10	再送されたプール操作
	poolstat_active	2	アクティブな get pool stat 操作
	poolstat_send	10	送信されたプール統計操作
	poolstat_resend	10	再送信されたプール統計
	statfs_active	2	statfs 操作
	statfs_send	10	送信された FS 統計
	statfs_resend	10	再送信された FS 統計
	command_active	2	アクティブなコマンド
	command_send	10	送信されたコマンド
	command_resend	10	再送信された コマンド
	map_epoch	2	OSD マップエポック
	map_full	10	受け取った完全な OSD マップ
	map_inc	10	受け取った増分 OSD マップ
	osd_sessions	2	セッションを開く
	osd_session_open	10	開いたセッション
	osd_session_close	10	閉じたセッション
	osd_laggy	2	Laggy OSD セッション

表8.10 読み出し操作および書き込み操作のメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
osd	**op_wip**	2	現在処理中のレプリケーション操作 (プライマリー)
	op_in_bytes	10	クライアント操作の合計書き込みサイズ
	op_out_bytes	10	クライアント操作合計読み取りサイズ
	op_latency	5	クライアント操作のレイテンシー (キュー時間を含む)
	op_process_latency	5	クライアント操作のレイテンシー (キュー時間を除く)
	op_r	10	クライアントの読み取り操作
	op_r_out_bytes	10	読み込まれたクライアントデータ
	op_r_latency	5	読み取り操作のレイテンシー (キュー時間を含む)
	op_r_process_latency	5	読み取り操作のレイテンシー (キュー時間を除く)
	op_w	10	クライアントの書き込み操作
	op_w_in_bytes	10	書き込まれたクライアントデータ
	op_w_rlat	5	クライアントの書き込み操作の読み取り可能/適用されるレイテンシー
	op_w_latency	5	書き込み操作のレイテンシー (キュー時間を含む)
	op_w_process_latency	5	書き込み操作のレイテンシー (キュー時間を除く)
	op_rw	10	クライアントの read-modify-write 操作
	op_rw_in_bytes	10	クライアントの read-modify-write 操作の書き込み

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	op_rw_out_bytes	10	クライアントの read-modify-write 操作の読み出し
	op_rw_rlat	5	クライアントの読み取り/書き込み操作の読み取り/適用のレイテンシー
	op_rw_latency	5	read-modify-write 操作のレイテンシー (キュー時間を含む)
	op_rw_process_latency	5	read-modify-write 操作のレイテンシー (キュー時間を除く)
	subop	10	サブ操作
	subop_in_bytes	10	サブ操作の合計サイズ
	subop_latency	5	サブ操作レイテンシー
	subop_w	10	レプリケートされた書き込み
	subop_w_in_bytes	10	複製された書き込みデータのサイズ
	subop_w_latency	5	レプリケートされた書き込みレイテンシー
	subop_pull	10	サブオペレーションのプルリクエスト
	subop_pull_latency	5	サブオペレーションのプルレイテンシー
	subop_push	10	サブオペレーションのプッシュメッセージ
	subop_push_in_bytes	10	サブオペレーションのプッシュサイズ
	subop_push_latency	5	サブオペレーションのプッシュレイテンシー
	pull	10	送信されたプル要求
	push	10	送信されたメッセージをプッシュ

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	push_out_bytes	10	プッシュされたサイズ
	push_in	10	インバウンドプッシュメッセージ
	push_in_bytes	10	インバウンドプッシュされるサイズ
	recovery_ops	10	開始したリカバリー操作
	loadavg	2	CPU 負荷
	buffer_bytes	2	割り当てられたバッファー合計サイズ
	numpg	2	配置グループ
	numpg_primary	2	この osd がプライマリーとなる配置グループ
	numpg_replica	2	この osd がレプリカである配置グループ
	numpg_stray	2	この osd から削除する準備ができている配置グループ
	heartbeat_to_peers	2	送信先のハートビート (ping) ピア
	heartbeat_from_peers	2	受信元ハートビート (ping) のピア
	map_messages	10	OSD マップメッセージ
	map_message_epochs	10	OSD マップエポック
	map_message_epoch_dups	10	OSD マップの複製
	stat_bytes	2	OSD のサイズ
	stat_bytes_used	2	使用されている領域
	stat_bytes_avail	2	利用可能な領域

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	copyfrom	10	RADOS の copy-from 操作
	tier_promote	10	階層の昇格
	tier_flush	10	階層フラッシュ
	tier_flush_fail	10	レイヤーフラッシュの失敗
	tier_try_flush	10	階層のフラッシュ試行
	tier_try_flush_fail	10	階層のフラッシュ試行の失敗
	tier_evict	10	階層エビクション
	tier_whiteout	10	階層のホワイトアウト
	tier_dirty	10	ダーティー階層フラグセット
	tier_clean	10	消去されたダーティー階層フラグ
	tier_delay	10	階層遅延 (エージェント待機)
	tier_proxy_read	10	階層プロキシーの読み込み
	agent_wake	10	階層化エージェントのウェイクアップ
	agent_skip	10	エージェントによりスキップされるオブジェクト
	agent_flush	10	階層化エージェントのフラッシュ
	agent_evict	10	階層化エージェントエビクション
	object_ctx_cache_hit	10	オブジェクトコンテキストキャッシュのヒット数
	object_ctx_cache_total	10	オブジェクトコンテキストキャッシュの検索

表8.11 リカバリー状態のメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
recoverystate_perf	**initial_latency**	5	リカバリーの状態の最初のレイテンシー
	started_latency	5	リカバリー状態のレイテンシーを開始
	reset_latency	5	リカバリー状態レイテンシーのリセット
	start_latency	5	リカバリー状態レイテンシーの開始
	primary_latency	5	プライマリーリカバリーの状態のレイテンシー
	peering_latency	5	リカバリー状態のレイテンシーのピア設定
	backfilling_latency	5	リカバリー状態レイテンシーのバックフィル
	waitremotebackfillreserved_latency	5	リモートバックフィルの予約されたリカバリー状態レイテンシーを待機する
	waitlocalbackfillreserved_latency	5	ローカルバックフィルの予約されたリカバリー状態のレイテンシーを待機する
	notbackfilling_latency	5	Notbackfilling のリカバリー状態のレイテンシー
	repnotrecovering_latency	5	Repnotrecovering リカバリー状態のレイテンシー
	repwaitrecoveryreserved_latency	5	repwaitrecovery が予約したリカバリー状態のレイテンシー
	repwaitbackfillreserved_latency	5	repwaitbackfill が予約したリカバリー状態のレイテンシー
	RepRecovering_latency	5	repRecovering リカバリー状態のレイテンシー
	activating_latency	5	リカバリー状態のレイテンシーの有効化

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	waitlocalrecoveryreserved_latency	5	waitlocalrecovery が予約したリカバリー状態のレイテンシー
	waitremoterecoveryreserved_latency	5	waitremoterecovery が予約したリカバリー状態のレイテンシー
	recovering_latency	5	リカバリー状態のレイテンシーのリカバリー
	recovered_latency	5	リカバリーしたリカバリー状態のレイテンシー
	clean_latency	5	リカバリー状態のレイテンシーの削除
	active_latency	5	アクティブリカバリーの状態のレイテンシー
	replicaactive_latency	5	replicaactive のリカバリー状態レイテンシー
	stray_latency	5	迷子のリカバリー状態レイテンシー
	getinfo_latency	5	getinfo リカバリー状態レイテンシー
	getlog_latency	5	getlog リカバリー状態レイテンシー
	waitactingchange_latency	5	Waitactingchange リカバリー状態レイテンシー
	incomplete_latency	5	不完全なリカバリー状態レイテンシー
	getmissing_latency	5	復旧状態のレイテンシーの取得
	waitupthru_latency	5	waitupthru リカバリー状態のレイテンシー

表8.12 OSD スロットルのメトリクステーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
throttle-*	**val**	10	現在利用できるスロットル
	max	10	スロットルの最大値
	get	10	取得
	get_sum	10	取得したデータ
	get_or_fail_fail	10	get_or_fail 時にブロックされる
	get_or_fail_success	10	get_or_fail 時の get 成功
	take	10	取得
	take_sum	10	取得したデータ
	put	10	送る
	put_sum	10	データを送る
	wait	5	待機レイテンシー

関連情報

- スロットルメトリックテーブルの書き戻し
- レベルのデータベースメトリックテーブル
- Objecter Metrics テーブル
- 読み出し操作および書き込み操作のメトリックテーブル
- リカバリー状態のメトリックテーブル
- OSD スロットルのメトリックテーブル

8.8. CEPH OBJECT GATEWAY メトリックス

表8.13 RADOS クライアントメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
client.rgw.<rgw_node_name>	**req**	10	要求
	failed_req	10	中止要求

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	get	10	取得
	get_b	10	取得サイズ
	get_initial_lat	5	レイテンシーの取得
	put	10	送る
	put_b	10	送信サイズ
	put_initial_lat	5	レイテンシーの送信
	qlen	2	キューの長さ
	qactive	2	アクティブなリクエストキュー
	cache_hit	10	キャッシュのヒット数
	cache_miss	10	キャッシュミス
	keystone_token_cache_hit	10	Keystone トークンキャッシュのヒット数
	keystone_token_cache_miss	10	Keystone のトークンキャッシュのミス
	gc_retire_object	10	Ceph Object Gateway の最後の再起動以降にリタイアしたオブジェクトの数

表8.14 Objecter Metrics テーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
objecter	**op_active**	2	アクティブな操作
	op_laggy	2	遅延操作
	op_send	10	送信された操作
	op_send_bytes	10	送信されたデータ
	op_resend	10	再送信捜査
	op_ack	10	コールバックのコミット

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	op_commit	10	操作のコミット
	op	10	操作
	op_r	10	読み取り操作
	op_w	10	書き込み操作
	op_rmw	10	read-modify-write 操作
	op_pg	10	PG 操作
	osdop_stat	10	統計操作
	osdop_create	10	オブジェクト操作の作成
	osdop_read	10	読み取り操作
	osdop_write	10	書き込み操作
	osdop_writefull	10	完全なオブジェクト操作の書き込み
	osdop_append	10	追加操作
	osdop_zero	10	オブジェクトをゼロ操作に設定
	osdop_truncate	10	オブジェクト操作の切り捨て
	osdop_delete	10	オブジェクト操作の削除
	osdop_mapext	10	エクステント操作のマップ
	osdop_sparse_read	10	スパース読み取り操作
	osdop_clonerange	10	範囲のクローン操作
	osdop_getxattr	10	xattr 操作の取得
	osdop_setxattr	10	xattr 操作の設定
	osdop_cmpxattr	10	xattr の比較操作
	osdop_rmxattr	10	xattr 操作の削除

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	osdop_resetxattrs	10	xattr 操作のリセット
	osdop_tmap_up	10	TMAP 更新操作
	osdop_tmap_put	10	TMAP の put 操作
	osdop_tmap_get	10	TMAP の get 操作
	osdop_call	10	操作の呼び出し (実行)
	osdop_watch	10	オブジェクト操作による監視
	osdop_notify	10	オブジェクト操作に関する通知
	osdop_src_cmpxattr	10	複数演算における拡張属性比較
	osdop_other	10	その他の操作
	linger_active	2	アクティブな linger 操作
	linger_send	10	送信された linger 操作
	linger_resend	10	再送信された linger 操作
	linger_ping	10	linger 操作に ping が送信
	poolop_active	2	アクティブなプール操作
	poolop_send	10	送信したプール操作
	poolop_resend	10	再送されたプール操作
	poolstat_active	2	アクティブな get pool stat 操作
	poolstat_send	10	送信されたプール統計操作
	poolstat_resend	10	再送信されたプール統計
	statfs_active	2	statfs 操作
	statfs_send	10	送信された FS 統計
	statfs_resend	10	再送信された FS 統計

コレクション名	メトリック名	ビットフィールド値	簡単な説明
	command_active	2	アクティブなコマンド
	command_send	10	送信されたコマンド
	command_resend	10	再送信された コマンド
	map_epoch	2	OSD マップエポック
	map_full	10	受け取った完全な OSD マップ
	map_inc	10	受け取った増分 OSD マップ
	osd_sessions	2	セッションを開く
	osd_session_open	10	開いたセッション
	osd_session_close	10	閉じたセッション
	osd_laggy	2	Laggy OSD セッション

表8.15 RADOS ゲートウェイスロットルメトリックテーブル

コレクション名	メトリック名	ビットフィールド値	簡単な説明
throttle-*	**val**	10	現在利用できるスロットル
	max	10	スロットルの最大値
	get	10	取得
	get_sum	10	取得したデータ
	get_or_fail_fail	10	get_or_fail 時にブロックされる
	get_or_fail_success	10	get_or_fail 時の get 成功
	take	10	取得
	take_sum	10	取得したデータ
	put	10	送る
	put_sum	10	データを送る

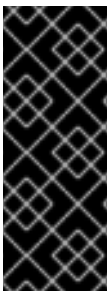
コレクション名	メトリック名	ビットフィールド値	簡単な説明
	wait	5	待機レイテンシー

関連情報

- RADOS Gateway クライアントテーブル
- Objecter Metrics テーブル
- RADOS ゲートウェイスロットルメトリックテーブル

第9章 BLUESTORE

Red Hat Ceph Storage 4 以降、BlueStore は OSD デーモンのデフォルトオブジェクトストアです。以前のオブジェクトストアである FileStore では、生のブロックデバイス上のファイルシステムを必要とします。その後、オブジェクトはファイルシステムに書き込まれます。BlueStore はブロックデバイスに直接オブジェクトを配置するため、BlueStore は最初のファイルシステムを必要としません。



重要

BlueStore は、本番環境で OSD デーモン向けに高パフォーマンスのバックエンドを提供します。デフォルトでは、BlueStore はセルフチューニングするように設定されています。BlueStore を手動でチューニングした方が環境のパフォーマンスが良いと判断された場合は、Red Hat サポート に連絡して設定の詳細を共有し、自動チューニングのケイパビリティーを改善する支援を受けるようにしてください。Red Hat は、フィードバックをお待ちしており、お客様の提案に感謝いたします。

9.1. CEPH BLUESTORE

以下は、BlueStore を使用する主な機能の一部です。

ストレージデバイスの直接管理

BlueStore は raw ブロックデバイスまたはパーティションを使用します。これにより、XFS などのローカルファイルシステムなど、抽象化の層が回避され、パフォーマンスの制限や複雑さの増加が発生する可能性があります。

RocksDB を使用したメタデータ管理

BlueStore は、ディスク上の場所をブロックするオブジェクト名からのマッピングなど、内部メタデータの管理にデーターベースのキーと値データベースを使用します。

完全なデータおよびメタデータのチェックサム

デフォルトでは、BlueStore に書き込まれたすべてのデータおよびメタデータは、1 つ以上のチェックサムによって保護されます。検証せずにディスクから読み取られたり、ユーザーに返されたデータやメタデータはありません。

効率的なコピーオンライト

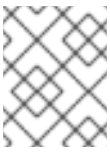
Ceph Block Device および Ceph File System のスナップショットは、BlueStore に効率的に実装されるコピーオンライトのクローンメカニズムに依存します。これにより、通常のスナップショットと、効率的な 2 フェーズコミットを実装するためにクローン作成に依存するイレイジャーコーディングプールの両方で効率的な I/O が実現します。

大きな二重書き込みなし

BlueStore は、まずブロックデバイス上の未割り当ての領域に新しいデータを書き込み、次にディスクの新しい領域を参照するためにオブジェクトのメタデータを更新する RocksDB トランザクションをコミットします。書き込み操作が設定可能なサイズしきい値を下回る場合にのみ、FileStore の操作方法と同様に、書き込み優先ジャーナリング方式にフォールバックします。

マルチデバイスのサポート

BlueStore は、複数のブロックデバイスを使用して異なるデータを保存できます。たとえば、データ用のハードディスクドライブ (HDD)、メタデータ用のソリッドステートドライブ (SSD)、不揮発性メモリー (NVM) や不揮発性ランダムアクセスメモリー (NVRAM)、RocksDB のライトアヘッドログ (WAL) 用の永続メモリーなどです。詳細は、**Ceph BlueStore デバイス** を参照してください。



注記

ceph-disk ユーティリティーがまだ複数のデバイスをプロビジョニングしません。複数のデバイスを使用するには、OSD を手動で設定する必要があります。

ブロックデバイスの効率的な使用方法

BlueStore はファイルシステムを使用しないため、ストレージデバイスキャッシュを削除する必要が最小限に抑えられます。

9.2. CEPH BLUESTORE デバイス

本セクションでは、BlueStore バックエンドが使用するブロックデバイスを説明します。

BlueStore は、1 つ、2 つ、または 3 つのストレージデバイスを管理します。

- プライマリー
- WAL
- DB

最も単純なケースでは、BlueStore は単一の (プライマリー) ストレージデバイスを使用します。ストレージデバイスは、以下を含む 2 つの部分に分割されます。

- **OSD メタデータ**: OSD の基本的なメタデータが含まれる XFS でフォーマットされた小規模なパーティション。このデータディレクトリーには、OSD に関する情報 (所属するクラスター、およびプライベートキーリング) が含まれます。
- **データ**: BlueStore によって直接管理され、すべての OSD データが含まれる残りのデバイスを占有する大容量パーティション。このプライマリーデバイスは、data ディレクトリーのブロックシンボリックリンクで識別されます。

2 つの追加デバイスを使用することもできます。

- **WAL (write-ahead-log) デバイス**: BlueStore 内部ジャーナルまたは write-ahead ログを保存するデバイス。これは、data ディレクトリーの **block.wal** シンボリックリンクによって識別されます。デバイスがプライマリーデバイスよりも高速の場合にのみ WAL デバイスを使用することを検討してください。たとえば、WAL デバイスが SSD ディスクを使用し、プライマリーデバイスが HDD ディスクを使用する場合です。
- **DB デバイス**: BlueStore 内部メタデータを保存するデバイス。組み込み RocksDB データベースは、パフォーマンスを向上させるために、プライマリーデバイスではなく DB デバイスにできるだけ多くのメタデータを配置します。DB デバイスが満杯になると、プライマリーデバイスへのメタデータの追加が開始します。デバイスがプライマリーデバイスよりも高速の場合にのみ DB デバイスを使用することを検討してください。

警告

高速デバイスで利用可能なギガバイトストレージのみが存在する場合。Red Hat は、WAL デバイスとして使用することを推奨します。より高速なデバイスがある場合は、DB デバイスとして使用することを検討してください。BlueStore ジャーナルは常に最速のデバイスに配置されるため、DB デバイスを使用すると、WAL デバイスと同じ利点が得られます。また、追加のメタデータを格納することもできます。

9.3. CEPH BLUESTORE キャッシュ

BlueStore キャッシュババッファーの集合体で、設定によっては OSD デーモンがディスクからの読み込みや書き込みを行う際に、データで埋められることがあります。Red Hat Ceph Storage のデフォルトでは、BlueStore は読み取り時にキャッシュされますが、書き込みは行いません。これは、キャッシュエビクションに関連するオーバーヘッドを回避するために **bluestore_default_buffered_write** オプションが **false** に設定されているためです。

bluestore_default_buffered_write オプションが **true** に設定されていると、データは最初にバッファーに書き込まれ、その後ディスクにコミットされます。その後、書き込みの確認がクライアントに送信されます。これにより、データがエビクトされるまで、キャッシュ内のデータへの読み取り速度が速くなります。

読み取り量の多いワークロードでは、BlueStore キャッシングからすぐに利益を得ることはできません。より多くの読み取りが行われると、キャッシュは時間の経過とともに増大し、後続の読み取りではパフォーマンスが向上するようになります。キャッシュがどのくらいの速さで生成されるかは、BlueStore のブロックおよびデータベースのディスクタイプ、ならびにクライアントのワークロード要件に依存します。

重要

bluestore_default_buffered_write オプションを有効にする前に、Red Hat サポート にお問い合わせください。

9.4. CEPH BLUESTORE のサイジングに関する考慮事項

BlueStore OSD を使用して従来のドライブとソリッドステートドライブを混在させる場合には、JusDB 論理ボリューム (**block.db**) のサイズを適切に設定することが重要です。Red Hat では、オブジェクト、ファイル、混合ワークロードで RocksDB の論理ボリュームをブロックサイズの 4% 以下にすることを推奨しています。Red Hat は、JlowsDB および OpenStack のブロックワークロードにおいて、BlueStore ブロックサイズの 1% をサポートしています。たとえば、オブジェクトワークロードのブロックサイズが 1 TB の場合は、最低でも 40 GB の RocksDB 論理ボリュームを作成します。

ドライブタイプを混合しない場合は、個別の RocksDB 論理ボリュームを持つ必要はありません。BlueStore は、RocksDB のサイジングを自動的に管理します。

BlueStore のキャッシュメモリーは、RocksDB、BlueStore のメタデータ、オブジェクトデータのキー/値のペアのメタデータで使用されます。

注記

BlueStore キャッシュのメモリー値は、OSD によってすでに使用されているメモリーフットプリントに追加されます。

9.5. CEPH BLUESTORE OSD の追加

本セクションでは、BlueStore バックエンドオブジェクトストアを使用して新たな Ceph OSD ノードをインストールする方法を説明します。

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. デフォルトでは、新しい OSD ノードを、(デフォルトでは **/etc/ansible/hosts** にある) Ansible インベントリーファイルの **[osds]** セクションに追加します。

```
[osds]
node1
node2
node3
HOST_NAME
```

以下を置き換えます。

- **HOST_NAME** は、OSD ノードの名前に置き換えます。

例

```
[osds]
node1
node2
node3
node4
```

2. **/usr/share/ceph-ansible** ディレクトリーに移動します。

```
[user@admin ~]$ cd /usr/share/ceph-ansible
```

3. **host_vars** ディレクトリーを作成します。

```
[root@admin ceph-ansible] mkdir host_vars
```

4. 新規に追加された OSD の設定ファイルを **host_vars** に作成します。

```
[root@admin ceph-ansible] touch host_vars/HOST_NAME.yml
```

以下を置き換えます。

- **HOST_NAME** を、新しく追加された Ceph ノードのホスト名に置き換えます。

例

```
[root@admin ceph-ansible] touch host_vars/node4.yml
```

5. 新規作成されたファイルに以下の設定を追加します。

```
osd_objectstore: bluestore
```

注記

すべての OSD に BlueStore を使用するには、**osd_objectstore:bluestore** を **group_vars/all.yml** ファイルに追加します。

6. **host_vars/HOST_NAME.yml** で BlueStore OSD を設定します。

構文

```
lvm_volumes:
  - data: DATALV
    data_vg: DATAVG
```

以下を置き換えます。

- **DATALV** を、データ論理ボリューム名に置き換えます。
- **DATAVG** を、データ論理ボリュームグループ名に置き換えます。

例

```
lvm_volumes:
  - data: data-lv1
    data_vg: vg1
```

7. オプション。**block.wal** と **block.db** を専用の論理ボリュームに保存する場合は、**host_vars/HOST_NAME.yml** ファイルを以下のように編集します。

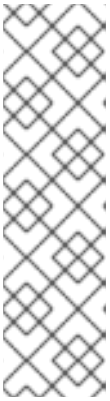
```
lvm_volumes:
  - data: DATALV
    wal: WALLV
    wal_vg: VG
    db: DBLV
    db_vg: VG
```

以下を置き換えます。

- DATALV を、データが含まれる論理ボリュームに置き換えます。
- **WALLV** を、write-ahead-log が含まれる論理ボリュームに置き換えます。
- **WALLV** を、WAL または DB デバイス、またはその両方の論理ボリュームが含まれるボリュームグループに置き換えます。
- **DBLV** を、BlueStore 内部メタデータが含まれるべき論理ボリュームに置き換えます。

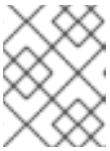
例

```
lvm_volumes:
  - data: data-lv3
    wal: wal-lv1
    wal_vg: vg3
    db: db-lv3
    db_vg: vg3
```



注記

lvm_volumes: を **osd_objectstore: bluestore** と一緒に使用する場合は、**lvm_volumes** の YAML 辞書には少なくとも **データ** が含まれている必要があります。**wal** または **db** を定義する際には、LV 名と VG 名の両方が必要になります (**db** と **wal** は必要ありません)。これにより、データのみ、データおよび wal、データおよび wal および db、またはデータおよび db の 4 つの組み合わせを使用できます。データは、生のデバイス、論理ボリューム、またはパーティションにすることができます。**wal** および **db** は、論理ボリュームまたはパーティションにすることができます。生のデバイスまたはパーティションの **ceph-volume** を指定すると、その上に論理ボリュームが配置されます。



注記

現在、**ceph-ansible** は、ボリュームグループまたは論理ボリュームを作成しません。Anisble Playbook を実行する前に、これを実行する必要があります。

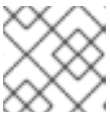
8. オプション: **group_vars/all.yml** ファイルで **block.db** のデフォルトサイズをオーバーライドできます。

構文

```
ceph_conf_overrides:
  osd:
    bluestore_block_db_size: VALUE
```

例

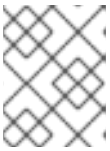
```
ceph_conf_overrides:
  osd:
    bluestore_block_db_size: 24336000000
```



注記

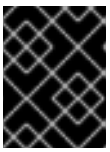
bluestore_block_db_size の値は 2 GB より大きくする必要があります。

9. **group_vars/all.yml** ファイルを開いて編集し、**osd_memory_target** オプションのコメント設定を解除します。OSD が消費するメモリー容量に応じて値を調整します。



注記

osd_memory_target オプションのデフォルト値は **4000000000** (4 GB) です。このオプションは、メモリー内の BlueStore キャッシュをピニングします。



重要

osd_memory_target オプションは、BlueStore がサポートする OSD にのみ適用されます。

10. 以下の Ansible Playbook を実行します。

```
[user@admin ceph-ansible]$ ansible-playbook site.yml
```

11. Ceph Monitor ノードから、新規 OSD が正常に追加されたことを確認します。

```
[root@mon ~]# ceph osd tree
```

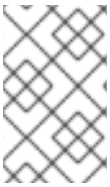
9.6. CEPH BLUESTORE をチューニングして小規模な書き込みを実現

BlueStore では、生のパーティションは **bluestore_min_alloc_size** のブロックで割り当て、管理されます。デフォルトでは、**bluestore_min_alloc_size** は HDD の場合は 64 KB、SSD では 4 KB になります。各チャンクの書き込みのない領域は、未加工パーティションに書き込まれる際にゼロで埋められます。これにより、小さいオブジェクトを書き込むなど、ワークロードのサイズが適切に設定されていない場合に未使用領域が無駄になる可能性があります。

書き込み増幅のペナルティーを回避できるように **bluestore_min_alloc_size** を最小書き込みに一致させることを推奨します。

たとえば、クライアントが 4 KB のオブジェクトを頻繁に書き込みする場合は、**ceph-ansible** を使用して OSD ノードで以下の設定を設定します。

bluestore_min_alloc_size = 4096



注記

bluestore_min_alloc_size_ssd 設定および **bluestore_min_alloc_size_hdd** 設定は、それぞれ SSD および HDD に固有のものですが、**bluestore_min_alloc_size** によりその設定が上書きされるため、設定する必要は必要ありません。

前提条件

- 稼働中の Red Hat Ceph Storage クラスターがある。
- OSD ノードとして新規にプロビジョニングできる新規サーバー、または
- 再デプロイできる OSD ノード。
- Ceph Monitor ノードの管理者キーリング (既存の Ceph OSD ノードを再デプロイする場合)。

手順

1. 必要に応じて、既存の OSD ノードを再デプロイする場合は、Ansible Playbook **shrink-osd.yml** を使用してクラスターから OSD を削除します。

```
ansible-playbook -v infrastructure-playbooks/shrink-osd.yml -e osd_to_kill=OSD_ID
```

例

```
[admin@admin ceph-ansible]$ ansible-playbook -v infrastructure-playbooks/shrink-osd.yml -e osd_to_kill=1
```

2. 既存の OSD ノードを再デプロイする場合には、OSD ドライブを削除して、OSD を再インストールします。

3. Ansible を使用して OSD プロビジョニング用のノードを準備します。準備タスクの例には、Red Hat Ceph Storage リポジトリーの有効化、Ansible ユーザーの追加、パスワードなしの SSH ログインの有効化が含まれます。

4. **bluestore_min_alloc_size** を Ansible Playbook **group_vars/all.yml** の **ceph_conf_overrides** セクションに追加します。

```
ceph_conf_overrides:
  osd:
    bluestore_min_alloc_size: 4096
```

5. 新規ノードをデプロイする場合は、これを Ansible インベントリーファイル (通常は **/etc/ansible/hosts**) に追加します。

```
[osds]
OSD_NODE_NAME
```

例

```
[osds]
osd1 devices="[ '/dev/sdb' ]"
```

6. 既存の OSD を再デプロイする場合は、Ceph Monitor ノードの管理キーリングファイルを OSD をデプロイするノードにコピーします。

7. Ansible を使用して OSD ノードをプロビジョニングします。

```
ansible-playbook -v site.yml -l OSD_NODE_NAME
```

例

```
[admin@admin ceph-ansible]$ ansible-playbook -v site.yml -l osd1
```

8. Playbook が完了したら、**ceph daemon** コマンドを使用して設定を確認します。

```
ceph daemon OSD.ID config get bluestore_min_alloc_size
```

例

```
[root@osd1 ~]# ceph daemon osd.1 config get bluestore_min_alloc_size
{
    "bluestore_min_alloc_size": "4096"
}
```

bluestore_min_alloc_size が 4096 バイトに設定されていることがわかります。これは 4 KiB に相当します。

関連情報

- 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。

9.7. BLUESTORE 断片化ツール

ストレージ管理者は、BlueStore OSD の断片化レベルを定期的にチェックする必要があります。オフライン OSD またはオンライン OSD の場合は、簡単な 1 つのコマンドを使用して断片化レベルを確認できます。

9.7.1. 前提条件

- 稼働中の Red Hat Ceph Storage 3.3 以上のストレージクラスター
- BlueStore OSD

9.7.2. BlueStore 断片化ツールとは

BlueStore OSD の場合は、基となるストレージデバイスの時間の経過とともに空き領域が断片化されます。一部の断片化は正常ですが、過剰な断片化が生じると、パフォーマンスが低下します。

BlueStore 断片化ツールは、BlueStore OSD の断片化レベルでスコアを生成します。この断片化スコアは 0 から 1 の範囲として指定されます。スコアが 0 の場合は断片化がなく、1 は深刻な断片化を意味します。

表9.1 断片化スコアの意味

スコア	断片化の量
0.0 - 0.4	なしから極小の断片化まで。
0.4 - 0.7	小さく、許容される断片化。
0.7 - 0.9	直感的ですが、安全な断片化です。
0.9 - 1.0	深刻な断片化があり、パフォーマンスの問題が発生することになります。

重要

深刻な断片化があり、問題の解決にサポートが必要な場合は、Red Hat サポート にお問い合わせください。

9.7.3. 断片化の確認

BlueStore OSD の断片化レベルのチェックは、オンラインまたはオフラインで行うことができます。

前提条件

- 稼働中の Red Hat Ceph Storage 3.3 以上のストレージクラスター
- BlueStore OSD

オンラインの BlueStore 断片化スコア

1. 実行中の BlueStore OSD プロセスを検証します。
   a. 簡単なレポート:

   構文

   ```
   ceph daemon OSD_ID bluestore allocator score block
   ```

例

```
[root@osd ~]# ceph daemon osd.123 bluestore allocator score block
```

b. より詳細なレポート:

構文

```
ceph daemon OSD_ID bluestore allocator dump block
```

例

```
[root@osd ~]# ceph daemon osd.123 bluestore allocator dump block
```

オフラインの BlueStore 断片化スコア

1. 実行していない BlueStore OSD プロセスを検証します。

a. 簡単なレポート:

構文

```
ceph-bluestore-tool --path PATH_TO_OSD_DATA_DIRECTORY --allocator block free-score
```

例

```
[root@osd ~]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-123 --allocator block free-score
```

b. より詳細なレポート:

構文

```
ceph-bluestore-tool --path PATH_TO_OSD_DATA_DIRECTORY --allocator block free-dump
```

例

```
[root@osd ~]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-123 --allocator block free-dump
```

関連情報

- 断片化スコアの詳細は、Red Hat Ceph Storage 4.1 の **BlueStore 断片化ツール** を参照してください。

9.8. オブジェクトストアを FILESTORE から BLUESTORE に移行する方法

ストレージ管理者は、従来のオブジェクトストアである FileStore から新しいオブジェクトストアである BlueStore に移行できます。

9.8.1. 前提条件

- 正常かつ稼働中の Red Hat Ceph Storage クラスター

9.8.2. FileStore から BlueStore への移行

BlueStore は従来の FileStore と比較すると、パフォーマンスと堅牢性が向上しています。1 つの Red Hat Ceph Storage クラスターには、FileStore デバイスと BlueStore デバイスの両方の組み合わせを含めることができます。

個々の OSD を変換することは、その場ではできませんし、単独でもできません。変換プロセスは、ストレージクラスターの通常のレプリケーションおよびヒーリングプロセス、または古い (FileStore) デバイスから新しい (BlueStore) デバイスに OSD コンテンツをコピーするツールや戦略のいずれかに依存します。FileStore から BlueStore に移行する方法は 2 つあります。

最初の操作

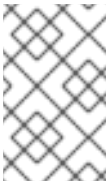
1 つ目は、各デバイスを順番に out とマークし、ストレージクラスター全体でデータの複製を待って、OSD を再プロビジョニングしてから再度 in のマークに戻すことです。以下は、この方法のメリットとデメリットです。

利点

- 簡単です。
- デバイスごとに実行できます。
- 予備のデバイスやノードは必要ありません。

デメリット

- ネットワーク上のデータのコピーは 2 回行われます。



注記

1 つのコピーをストレージクラスター内の他の OSD にコピーすることで、希望する数のレプリカを維持し、さらにもう 1 つのコピーを再プロビジョニングされた BlueStore OSD に戻すことができます。

2 つ目の方法

2 つ目は、ノード全体を置き換えます。データがない空のノードが必要です。

これを実行するには 2 つの方法を使用できます。*ストレージクラスターの一部ではない新規の空のノードから開始します。* ストレージクラスターの既存ノードからデータをオフロードする

利点

- データは一度だけネットワーク上でコピーされます。
- ノードの OSD 全体を一度に変換します。
- 一度に複数のノードを並行して変換できます。
- 各ノードでは、予備のデバイスは必要ありません。

デメリット

- スペアノードが必要です。
- ノード全体の OSD は、データを一度に移行します。これは、クラスター全体のパフォーマンスに影響を与える可能性が高くなります。
- 移行されたデータはすべて、ネットワーク上で 1 つの完全ホップを行います。

9.8.3. Ansible を使用した FileStore から BlueStore への移行

Ansible を使用して FileStore から BlueStore に移行すると、ノード上のすべての OSD が縮小され、再デプロイされます。Ansible Playbook は、移行を開始する前に容量チェックを実行します。**ceph-volume** ユーティリティーは OSD を再デプロイします。

前提条件

- 正常かつ稼働中の Red Hat Ceph Storage 4 クラスター
- Ansible アプリケーションで使用する **ansible** ユーザーアカウント。

手順

1. Ansible 管理ノードで **ansible** ユーザーとしてログインします。
2. **group_vars/osd.yml** ファイルを編集し、以下のオプションを追加し、設定します。

```
nb_retry_wait_osd_up: 50
delay_wait_osd_up: 30
```

3. 以下の Ansible Playbook を実行します。

構文

```
ansible-playbook infrastructure-playbooks/filestore-to-bluestore.yml --limit
OSD_NODE_TO_MIGRATE
```

例

```
[ansible@admin ~]$ ansible-playbook infrastructure-playbooks/filestore-to-bluestore.yml --
limit osd1
```

警告

Ceph 設定ファイルで **osd_crush_update_on_start = False** を明示的に設定すると、変換は失敗します。別の ID で新しい OSD を作成し、それを CRUSH ルールに置き忘れます。また、古い OSD データディレクトリーはクリアされません。

4. ストレージクラスターの次の OSD ノードで開始する前に、移行が完了するまで待ちます。

9.8.4. マークアウトと置換のアプローチを使用して FileStore から BlueStore への移行

FileStore から BlueStore に移行する最も簡単な方法は、各デバイスを順次マークアウトし、ストレージクラスター全体でデータが複製されるのを待ち、OSD を再プロビジョニングし、再度 in のマークに戻すことです。

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 **root** アクセス。

手順

以下の変数 **OSD_ID** を ODS の識別番号に置き換えます。

1. 置き換える FileStore OSD を検索します。
   a. OSD 識別番号を取得します。

```
[root@ceph-client ~]# ceph osd tree
```

   b. OSD が FileStore または BlueStore を使用しているかどうかを特定します。

   構文

```
ceph osd metadata OSD_ID | grep osd_objectstore
```

   例

```
[root@ceph-client ~]# ceph osd metadata 0 | grep osd_objectstore
    "osd_objectstore": "filestore",
```

   c. FileStore デバイスと BlueStore デバイス数の現在の数を表示するには、次のコマンドを実行します。

```
[root@ceph-client ~]# ceph osd count-metadata osd_objectstore
```

2. FileStore OSD にマークが付けられます。

```
ceph osd out OSD_ID
```

3. データが OSD から移行されるまで待機します。

```
while ! ceph osd safe-to-destroy OSD_ID ; do sleep 60 ; done
```

4. OSD を停止します。

```
systemctl stop ceph-osd@OSD_ID
```

5. この OSD が使用しているデバイスを取得します。

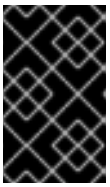
```
mount | grep /var/lib/ceph/osd/ceph-OSD_ID
```

6. OSD をアンマウントします。

```
umount /var/lib/ceph/osd/ceph-OSD_ID
```

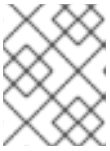
7. 手順 5 からの値を **DEVICE** として使用して、OSD データを破棄します。

```
ceph-volume lvm zap DEVICE
```



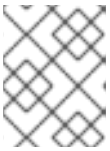
重要

デバイスの内容が破壊されますので、**非常に注意** してください。先に進む前に、デバイスのデータが不要で、ストレージクラスターが正常であることを確認してください。



注記

OSD が暗号化されている場合は、**dmsetup remove** を使用して OSD をザッピングする前に、**osd-lockbox** をアンマウントして暗号化を削除します。



注記

OSD に論理ボリュームが含まれている場合は、**ceph-volume lvm zap** コマンドで **--destroy** オプションを使用します。

8. OSD が破棄されていることをストレージクラスターに認識させます。

```
[root@ceph-client ~]# ceph osd destroy OSD_ID --yes-i-really-mean-it
```

9. ステップ 5 の **DEVICE** と、同じ **OSD_ID** を使用して、OSD を BlueStore OSD として再プロビジョニングします。

```
[root@ceph-client ~]# ceph-volume lvm create --bluestore --data DEVICE --osd-id OSD_ID
```

10. この手順を繰り返します。

注記

新規 BlueStore OSD の再入力は、OSD を破棄する前にストレージクラスターが **HEALTH_OK** であることを確認していれば、次の FileStore OSD のドレイン (枯渇) と同時に発生する可能性があります。これを行わないと、データの冗長性が軽減され、リスクが高まるか、データ損失の可能性が高まります。

9.8.5. ノード全体の置き換え方法を使用した FileStore から BlueStore への移行

FileStore から BlueStore への移行は、保存されているデータの各コピーを一度だけ転送することで、ノード単位で行うことができます。このマイグレーションは、ストレージクラスター内に予備のノードがある場合や、予備として使用するためにストレージクラスターからノード全体を退避させるのに十分な空き領域を持っている場合に行うことができます。理想的には、移行する他のノードとほぼ同じ容量のノードでなければなりません。

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 **root** アクセス。
- データがない空のノード。

手順

- 以下の変数 **NEWNODE** を、新しいノード名に置き換えます。
- 以下の変数 **EXISTING_NODE_TO_CONVERT** を、ストレージクラスターにすでに存在するノード名に置き換えます。
- 以下の変数 **OSD_ID** を、OSD 識別番号に置き換えます。

1. ストレージクラスターにない新規ノードを使用する。ストレージクラスターに存在するノードを使用するには、ステップ 3 に進みます。

   a. ノードを CRUSH 階層に追加します。

   ```
   [root@mon ~]# ceph osd crush add-bucket NEWNODE node
   ```

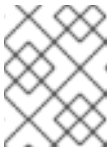
   

   重要

   ルートには接続しないでください。

   b. Ceph ソフトウェアパッケージをインストールします。

   ```
   [root@mon ~]# yum install ceph-osd
   ```

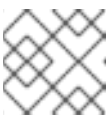
   

   注記

   Ceph 設定ファイル (デフォルトでは **/etc/ceph/ceph.conf**) およびキーリングを新しいノードにコピーします。

2. ステップ 5 に進みます。
3. ストレージクラスターで既存のノードを使用している場合は、以下のコマンドを使用します。

   ```
   [root@mon ~]# ceph osd crush unlink EXISTING_NODE_TO_CONVERT default
   ```

   注記

   ここでの **default** は、CRUSH マップの直系先祖です。

4. ステップ 8 に進みます。
5. すべてのデバイスに対して新しい BlueStore OSD をプロビジョニングします。

   ```
   [root@mon ~]# ceph-volume lvm create --bluestore --data /dev/DEVICE
   ```

6. OSD がクラスターに参加していることを確認します。

```
[root@mon ~]# ceph osd tree
```

新しいノード名の下にすべての OSD が表示されるはずですが、階層内の他のノードの下に入れ子に **なってはいけません**。

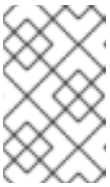
例

```
[root@mon ~]# ceph osd tree
ID CLASS WEIGHT  TYPE NAME     STATUS REWEIGHT PRI-AFF
-5            0 node newnode
10   ssd 1.00000     osd.10       up  1.00000 1.00000
11   ssd 1.00000     osd.11       up  1.00000 1.00000
12   ssd 1.00000     osd.12       up  1.00000 1.00000
-1       3.00000 root default
-2       3.00000     node oldnode1
0   ssd 1.00000         osd.0     up  1.00000 1.00000
1   ssd 1.00000         osd.1     up  1.00000 1.00000
2   ssd 1.00000         osd.2     up  1.00000 1.00000
```

7. 新規ノードを、クラスター内の古いノードの位置にスワップします。

```
[root@mon ~]# ceph osd crush swap-bucket NEWNODE
EXISTING_NODE_TO_CONVERT
```

この時点で、**EXISTING_NODE_TO_CONVERT** のすべてのデータは **NEWNODE** 上の OSD への移行を開始します。



注記

古いノードと新規ノードの合計容量に違いがある場合は、ストレージクラスター内の他のノードへのデータ移行も表示されますが、ノードのサイズが同じであれば、これは比較的少ないデータ量になります。

8. データの移行が完了するまで待ちます。

```
while ! ceph osd safe-to-destroy $(ceph osd ls-tree EXISTING_NODE_TO_CONVERT);
do sleep 60 ; done
```

9. **EXISTING_NODE_TO_CONVERT** にログインし、現時点で空の **EXISTING_NODE_TO_CONVERT** にある古い OSD をすべて停止およびアンマウントします。

```
[root@mon ~]# systemctl stop ceph-osd@OSD_ID
[root@mon ~]# umount /var/lib/ceph/osd/ceph-OSD_ID
```

10. 古い OSD を破棄してパージします。

```
for osd in ceph osd ls-tree EXISTING_NODE_TO_CONVERT; do ceph osd purge $osd -
-yes-i-really-mean-it ; done
```

11. 古い OSD デバイスを削除します。これには、手動で消去するデバイスを特定する必要があ

11. 古い OSD デバイスを削除します。これには、手動で消去するデバイスを特定する必要があります。各デバイスに対して、次のコマンドを実行します。

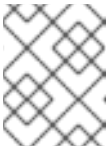
```
[root@mon ~]# ceph-volume lvm zap DEVICE
```

重要

デバイスの内容が破壊されますので、**非常に注意** してください。次のステップに進む前に、デバイスのデータが不要で、ストレージクラスターが正常であることを確認してください。

注記

OSD が暗号化されている場合は、**dmsetup remove** を使用して OSD をザッピングする前に、**osd-lockbox** をアンマウントして暗号化を削除します。



注記

OSD に論理ボリュームが含まれている場合は、**ceph-volume lvm zap** コマンドで **--destroy** オプションを使用します。

12. 現在空になっている古いノードを新しいノードとして使用し、処理を繰り返します。