Red Hat Ceph Storage 4

管理指南

管理 Red Hat Ceph Storage

Last Updated: 2021-08-26

Red Hat Ceph Storage 4 管理指南

管理 Red Hat Ceph Storage

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本文档描述了如何管理进程、监控集群状态、管理用户，以及添加和删除红帽 Ceph 存储的守护进程。 红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始：master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 CTO Chris Wright 信息。

目录

第 1 章 CEPH 管理

红帽 Ceph 存储集群是所有 Ceph 部署的基础。在部署红帽 Ceph 存储集群后，可以通过管理操作保持红帽 Ceph 存储集群正常运行并保持最佳运行。

红帽 Ceph 存储管理指南可帮助存储管理员执行以下任务：

- 如何检查我的红帽 Ceph 存储群集的健康状态？
- 如何启动和停止红帽 Ceph 存储群集服务？
- 如何为正在运行的红帽 Ceph 存储集群添加或删除 OSD？
- 如何管理红帽 Ceph 存储集群中存储的对象的用户身份验证和访问控制？
- 我想了解如何通过红帽 Ceph 存储集群使用覆盖。
- 我想监控红帽 Ceph 存储集群的性能。

基本的 Ceph 存储集群由两种类型的守护进程组成：

- Ceph 对象存储设备(OSD)将数据存储为对象存储在分配给 OSD 的 PG 中
- Ceph monitor 维护集群映射的主副本

生产系统将具有三个或更多 Ceph 监控器来实现高可用性，通常至少有 50 个 OSD 用于接受的负载平衡、数据重新平衡和数据恢复。

其它资源

- Red Hat Ceph Storage 安装指南

第 2 章 了解 CEPH 的进程管理

作为存储管理员，您可以通过类型或实例、裸机或容器中操作各种 Ceph 守护进程。通过操控这些守护进程，您可以根据需要启动、停止和重新启动所有 Ceph 服务。

2.1. 先决条件

- 安装红帽 Ceph 存储软件.

2.2. CEPH 进程管理

在红帽 Ceph 存储中，所有流程管理都是通过 Systemd 服务完成的。每次您想要 **start**、**restart** 和 **stop** Ceph 守护进程时，您必须指定守护进程类型或守护进程实例。

其它资源

- 有关使用 Systemd 的更多信息，请参阅 Red Hat Enterprise Linux *系统管理员指南中的使用 systemd 管理服务* 一章。

2.3. 启动、停止和重启所有 CEPH 守护进程

以 **admin** 用户身份从节点启动、停止和重启所有 Ceph 守护进程。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 对节点具有 **root** 访问权限。

流程

1. 启动所有 Ceph 守护进程：

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

2. 停止所有 Ceph 守护进程：

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

3. 重启所有 Ceph 守护进程：

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

2.4. 根据类型启动、停止和重启 CEPH 守护进程

若要启动、停止或重新启动特定类型的所有 Ceph 守护进程，请在运行 Ceph 守护进程的节点上遵循以下步骤。

先决条件

- 正在运行的红帽 Ceph 存储群集.

- 对节点具有 **root** 访问权限。

流程

- 在 **Ceph 监控** 节点上：

  开始：

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

  停止：

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

  重启：

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

- 在 **Ceph Manager** 节点上：

  开始：

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

  停止：

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

  重启：

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

- 在 **Ceph OSD** 节点上：

  开始：

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

  停止：

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

  重启：

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

- 在 **Ceph 对象网关** 节点上：

  开始：

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

停止：

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

重启：

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

2.5. 通过实例启动、停止和重启 CEPH 守护进程

若要通过实例启动、停止或重新启动 Ceph 守护进程，请在运行 Ceph 守护进程的节点上遵循下列步骤。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 对节点具有 **root** 访问权限。

流程

- 在 **Ceph 监控** 节点上：

  开始：

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

  停止：

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

  重启：

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

  replace

  - ***MONITOR_HOST_NAME*** 使用 Ceph 监控节点的名称。

- 在 **Ceph Manager** 节点上：

  开始：

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

  停止：

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

重启：

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

replace

- ***MANAGER_HOST_NAME*** 使用 Ceph 管理器节点的名称。

- 在 **Ceph OSD** 节点上：

  开始：

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

  停止：

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

  重启：

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

  replace

  - ***OSD_NUMBER*** Ceph OSD 的 **ID** 数量。
    例如，当查看 **ceph osd tree** 命令输出时，**osd.0** 的 **ID** 为 **0**。

- 在 **Ceph 对象网关** 节点上：

  开始：

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

  停止：

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

  重启：

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

  replace

  - ***OBJ_GATEWAY_HOST_NAME*** 使用 Ceph 对象网关节点的名称。

2.6. 启动、停止和重启容器中运行的 CEPH 守护进程

使用 **systemctl** 命令启动、停止或重启容器中运行的 Ceph 守护进程。

先决条件

- 安装红帽 Ceph 存储软件.
- 节点的根级别访问权限。

流程

1. 要启动、停止或重启容器中运行的 Ceph 守护进程，以 **root** 用户身份运行 **systemctl** 命令，其格式如下：

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

replace

- *ACTION* 是要执行的操作； **start**、**stop** 或 **restart**。
- *DAEMON* 是守护进程，**osd**、**mon**、**mds** 或 **rgw**。
- *ID* 是：
  - **ceph-mon**、**ceph-mds** 或 **ceph-rgw** 守护进程运行的短主机名。
  - 如果部署了 **ceph-osd** 守护进程的 ID。

例如：要重启 ID 为 **osd01** 的 **ceph-osd** 守护进程：

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

启动在 **ceph-monitor01** 主机上运行的 **ceph-mon** 演示：

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

停止在 **ceph-rgw01** 主机上运行的 **ceph-rgw** 守护进程：

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

2. 验证该操作是否已成功完成。

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

例如：

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

其它资源

- 有关更多信息，请参见 *《红帽 Ceph 存储管理指南》中的了解 Ceph 进程管理* 一章。

2.7. 查看容器中运行的 CEPH 守护进程的日志文件

使用容器主机的 **journald** 守护进程，查看容器中 Ceph 守护进程的日志文件。

先决条件

- 安装红帽 Ceph 存储软件.
- 节点的根级别访问权限。

流程

1. 要查看整个 Ceph 日志文件，以 **root** 用户身份运行一个 **journalctl** 命令，其格式如下：

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

replace

- *DAEMON* 是 Ceph 守护进程； **osd**、**mon** 或 **rgw**。
- *ID* 是：
  - **ceph-mon**、**ceph-mds** 或 **ceph-rgw** 守护进程运行的短主机名。
  - 如果部署了 **ceph-osd** 守护进程的 ID。

例如，要查看 ID 为 **osd01** 的 **ceph-osd** 守护进程的整个日志：

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

2. 要只显示最近的日志条目，使用 **-f** 选项。

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

例如，只查看在 **ceph-monitor01** 主机上运行的 **ceph-mon** 守护进程的最新日志条目：

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

注意

您还可以使用 **sosreport** 工具查看 **journald** 日志。有关 SOS 报告的详情，请查看 什么是 sosreport 以及如何在 Red Hat Enterprise Linux 中创建？红帽客户门户上的解决方案.

其它资源

- **journalctl(1)** man page。

2.8. 关闭并重启 RED HAT CEPH STORAGE 集群

按照以下步骤关闭和重新引导 Ceph 集群：

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 具有 **root** 访问权。

流程

关闭 Red Hat Ceph Storage 集群

1. 停止客户端在此集群和任何其他客户端上使用 RBD 镜像和 RADOS 网关。

2. 在继续操作前，集群必须处于健康状态（**Health_OK** 和所有 PG **active+clean**）。使用客户端密钥环（如 Ceph 监控器或 OpenStack 控制器节点）在节点上运行 **ceph status**，以确保集群正常运行。

3. 如果使用 Ceph 文件系统(**CephFS**)，则必须关闭 **CephFS** 集群。关闭 **CephFS** 集群的方法是将等级数量减少到 **1**，设置 **cluster_down** 标志，然后失败最后一个等级。

   例如：

   ```
   [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
   ```

   设置 **cluster_down** 标志可防止待机接管失败的等级。

4. 设置 **noout**、**norecover**、**norebalance**、**nobackfill**、**nodown** 和 **pause** 标志。使用客户端密钥环在节点上运行以下命令：例如，Ceph 监控器或 OpenStack 控制器节点：

   ```
   [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. 逐一关闭 OSD 节点：

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

6. 逐一关闭监控节点：

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

重启 Red Hat Ceph Storage 集群

1. 打开管理节点。

2. 打开监控节点：

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

3. 打开 OSD 节点：

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

4. 等待所有节点出现。验证所有服务均已启动，并且节点之间连接正常。

5. 取消设置 **noout**、**norecover**、**norebalance**、**nobackfill**、**nodown** 和 **pause** 标志。使用客户端密钥环在节点上运行以下命令：例如，Ceph 监控器或 OpenStack 控制器节点：

```
[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. 如果使用 Ceph 文件系统(**CephFS**)，则必须通过将 **cluster_down** 标志设置为 **false** 来激活 **CephFS** 集群：

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

7. 验证集群处于健康状态（**Health_OK** 和所有 PG **active+clean**）。使用客户端密钥环在节点上运行 **ceph status**。例如，Ceph 监控器或 OpenStack 控制器节点，以确保集群正常运行。

2.9. 其它资源

- 有关安装 Ceph 的更多信息，请参阅《*Red Hat Ceph 存储安装指南*》

第 3 章 监控 CEPH 存储集群

作为存储管理员，您可以监控红帽 Ceph 存储集群的整体健康状况，以及监控 Ceph 各个组件的健康状况。

在您拥有正在运行的红帽 Ceph 存储集群后，您可以开始监控存储集群，以确保 Ceph monitor 和 Ceph OSD 守护进程正在运行。Ceph 存储集群客户端连接到 Ceph 监控器，并接收最新版本的存储集群映射，然后才能将数据读取和写入到存储群集内的 Ceph 池。因此，监控集群必须在 Ceph 客户端可以读取和写入数据之前就群集状态达成一致。

Ceph OSD 必须对Primary OSD 上的放置组进行对等，以及次要 OSD 上的 PG 副本。如果出现故障，对等将反映 **active + clean** 状态以外的内容。

3.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

3.2. CEPH 存储集群的高级监控

作为存储管理员，您可以监控 Ceph 守护进程的运行状况，确保它们已启动并在运行。高级别监控还需要检查存储集群容量，以确保存储集群没有超过其 **full ratio**。红帽 Ceph 存储控制面板 是进行高级别监控的最常见方式。但是，您也可以使用命令行界面、Ceph 管理 socket 或 Ceph API 来监控存储集群。

3.2.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

3.2.2. 以交互方式使用 Ceph 命令界面

您可以使用 **ceph** 命令行工具与 Ceph 存储集群交互接口。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 以互动模式运行 **ceph** 工具。

   a. **裸机** 部署：

   示例

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

   b. **容器** 部署：

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
```

replace

- 带有 Ceph *监控容器名称的 MONITOR_NAME*，分别通过运行 **docker ps** 或 **podman ps** 命令找到。

  示例

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

  本例在 **mon01** 上打开了一个交互式终端会话，您可以在其中启动 Ceph 互动 shell。

3.2.3. 检查存储集群运行状况

启动 Ceph 存储集群后，在开始读取或写入数据之前，请先检查存储集群的运行状况。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 您可以使用以下命令检查 Ceph 存储集群的健康状况：

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

2. 如果您为配置或密钥环指定了非默认位置，您可以指定它们的位置：

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

启动 Ceph 集群后，您可能会遇到 **HEALTH_WARN XXX num placement groups stale** 等健康警告。等待几分钟，然后再次检查。当存储集群就绪时，**ceph health** 应该返回信息，如 **HEALTH_OK**。此时，开始使用集群没有问题。

3.2.4. 监视存储集群事件

您可以使用命令行界面观察 Ceph 存储集群上发生的事件。

先决条件

- 正在运行的红帽 Ceph 存储群集.

- 节点的根级别访问权限。

流程

1. 要在命令行中监控集群持续事件，请打开一个新终端，然后输入：

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

Ceph 将打印每个事件。例如，一个 tiny Ceph 集群由一个 monitor 组成，两个 OSD 可能会打印以下内容：

```
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
```

输出提供：

- 集群 ID
- 集群健康状态
- monitor map epoch 以及 monitor 仲裁的状态
- OSD map epoch 以及 OSD 的状态
- 放置组映射版本
- 放置组和池的数量
- 存储的数据 *的概念量* 和存储的对象数量
- 存储的数据总数

3.2.5. Ceph 如何计算数据使用量

used 值反映了使用的原始存储 *的实际*数量。**xxx GB / xxx GB** 值表示集群总存储容量的可用数量（两个数字越小）。概念数反映了数据在复制、克隆或快照之前的大小。因此，实际存储的数据量通常超过存储的概念量，因为 Ceph 会创建数据副本，也可能会使用存储容量进行克隆和快照。

3.2.6. 了解存储集群用量统计

要检查集群的数据使用情况和数据分布，请使用 **df** 选项。它与 Linux **df** 命令类似。执行以下命令：

```
[root@mon ~]# ceph df
RAW STORAGE:
    CLASS     SIZE        AVAIL       USED        RAW USED     %RAW USED
    hdd       662 TiB     611 TiB     51 TiB      51 TiB          7.74
    TOTAL     662 TiB     611 TiB     51 TiB      51 TiB          7.74

POOLS:
    POOL                    ID    STORED    OBJECTS    USED     %USED    MAX AVAIL
    default.rgw.users.keys   276       0 B          0       0 B      0     193 TiB
    default.rgw.data.root    277       0 B          0       0 B      0     193 TiB
    .rgw.root                278   5.7 KiB         12   2.2 MiB      0     193 TiB
    default.rgw.control      279       0 B          8       0 B      0     193 TiB
    default.rgw.gc           280       0 B          0       0 B      0     193 TiB
```

输出 **的 RAW STORAGE** 部分概述了存储集群用于数据的存储量。

- **CLASS：** 所用设备的类型。
- **SIZE：** 存储集群的总体存储容量。
- **AVAIL：** 存储集群中可用的可用空间量。
- **USED：** 存储集群中的已用空间量。
- **RAW USED：** **USED** 空间总和分配 **db** 和 **wal** BlueStore 分区的空间。
- **% RAW USED：** RAW **USED** 的百分比.将此号码与 **full ratio** 和 **near full ratio** 一起使用，以确保您没有达到存储集群的容量。

输出的 **POOLS** 部分提供了池列表以及每个池的概念用法。本节的输出 **不** 反映副本、克隆或快照。例如，如果您存储了 1 MB 数据的对象，则概念性使用为 1 MB，但实际使用量可能为 3 MB 或更多，具体取决于副本数量，如 **size = 3**、克隆和快照。

- **POOL：** 池的名称。
- **id：** 池 ID。
- **STORED：** 用户存储的数据量。
- **OBJECTS：** 每个池存储的对象的概念数量。
- **USED：** 以 KB 为单位存储的数据概念量，除非该数字附加 **M** 代表兆字节或 **G** （千兆字节）。
- **%USED：** 每个池使用的存储的理念百分比。
- **MAX AVAIL：** 池中最大可用空间。

注意

POOLS 部分中的数字是概念性的。它们不包含副本、快照或克隆的数量。因此，**USED** 和 **%USED** 金额的总和不会计入输出的 **GLOBAL** 部分中的 **RAW USED** 和 **%RAW USED** 金额。

其它资源

- 详情请参阅 Ceph 如何计算数据使用量。
- 详情请参阅 了解 OSD 使用量统计。

3.2.7. 了解 OSD 使用量统计

使用 **ceph osd df** 命令查看 OSD 利用率统计。

```
[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 的名称。
- **CLASS：** OSD 使用的设备类型。
- **WEIGHT：** CRUSH map 中 OSD 的权重。
- **REWEIGHT：** 默认的重新加权值。
- **SIZE：** OSD 的总体存储容量。
- **USE：** OSD 容量。
- **DATA：** 用户数据使用的 OSD 容量大小。
- **OMAP：** 用于存储对象映射(**omap**)数据（存储在 **rocksdb**中的键值对）的 **bluefs** 存储的估算值。
- **META：** 为内部元数据分配的 **bluefs** 空间或 **bluestore_bluefs_min** 参数中设置的值（以较大为准）计算为 **bluefs** 中分配的总空间减去估计的 **omap** 数据大小。
- **AVAIL：** OSD 上的可用空间量。
- **%USE：** OSD 使用的存储概念百分比
- **VAR：** 平均利用率上或低于平均利用率的变化。
- **PGS：** OSD 中的 PG 数量。
- **MIN/MAX VAR：** 所有 OSD 之间的最小和最大变化。

其它资源

- 详情请参阅 *Ceph 如何计算数据使用量*。
- 详情请参阅 *了解 OSD 使用量统计*。
- 详情请参阅 *红帽 Ceph 存储策略指南* 中的 *CRUSH Weights*。

3.2.8. 检查 Red Hat Ceph Storage 集群状态

您可以从命令行界面检查红帽 Ceph 存储集群的状态。**status** 子命令或 **-s** 参数将显示存储集群的当前状态。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要检查存储集群的状态，请执行以下操作：

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

或者：

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

2. 在互动模式中输入 **status** 并按 **Enter** ：

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

例如，一个 tiny Ceph 集群由一个 monitor 组成，两个 OSD 可以打印以下内容：

```
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. 检查 Ceph monitor 状态

如果存储集群具有多个 Ceph 监控（这是生产红帽 Ceph 存储集群要求）的要求，那么在启动存储集群后以及进行任何读取或写入数据之前，请检查 Ceph monitor 仲裁状态。

当多个监视器正在运行时，必须存在仲裁。

定期检查 Ceph 监控状态，确保它们正在运行。如果 Ceph monitor 出现问题，阻止对存储集群状态达成一致，则故障可能会阻止 Ceph 客户端读取和写入数据。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要显示 monitor map，请执行以下操作：

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

或者

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

2. 要检查存储集群的仲裁状态，请执行以下操作：

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

Ceph 将返回仲裁状态。由三个监视器组成的 Red Hat Ceph Storage 集群可能会返回以下内容：

示例

```
{ "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. 使用 Ceph 管理套接字

使用管理 socket 文件直接与给定守护进程交互。例如，套接字可让您：

- 在运行时列出 Ceph 配置
- 在运行时直接设置配置值，而不依赖于 monitor。这在监控器为 **down** 时非常有用。
- 转储历史操作

- 转储操作优先级队列状态
- 在不重启的情况下转储操作
- 转储性能计数器

此外，在对 monitor 或 OSD 相关的问题进行故障排除时，使用套接字也很有帮助。

重要

管理 socket 仅在守护进程运行时可用。当您正确关闭守护进程时，管理套接字会被删除。但是，如果守护进程意外终止，管理套接字可能持久存在。

不管守护进程没有运行，在尝试使用管理套接字时会返回以下错误：

```
Error 111: Connection Refused
```

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 使用套接字：

   语法

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

   替换：

   - ***TYPE*** 使用 Ceph 守护进程的类型（**mon**、**osd**、**mds**）。
   - ***ID*** 使用守护进程 ID
   - ***COMMAND*** 使用 命令来运行。使用 **help** 列出给定守护进程的可用命令。

     示例

     查看名为 **mon.0** 的 Ceph monitor 的 monitor 状态：

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

2. 或者，也可使用其套接字文件指定 Ceph 守护进程：

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

3. 查看名为 **osd.2** 的 Ceph OSD 的状态：

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

4. 列出 Ceph 进程的所有套接字文件：

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

其它资源

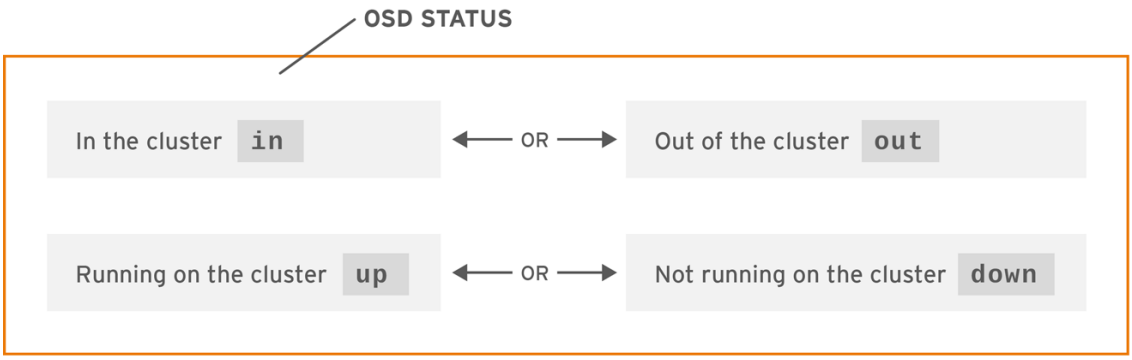
- 如需更多信息，*请参阅红帽 Ceph 存储故障排除指南*。

3.2.11. 了解 Ceph OSD 状态

OSD 的状态是集群中的 **in**，或来自集群中的 **out**。它可启动并运行 **up**，或者它停机且未运行，或 **down**。如果 OSD 是 **up**，它可以是 **in** 存储集群，其中的数据可以被读取和写入，或者是存储集群的 **out**。如果是 **in** 集群，并且最近移动了集群的 **out**，Ceph 会将放置组迁移到其他 OSD。如果 OSD 是集群的 **out**，CRUSH 不会分配 PG 到 OSD。如果 OSD 是 **down**，它也应是 **out**。

注意

如果 OSD 是 **down** 和 **in**，则会出现一个问题，集群不会处于健康状态。



如果您执行 **ceph health**、**ceph -s** 或 **ceph -w** 等命令，您可能会注意到集群并不总是回显 **HEALTH OK**。不要 panic。对于 OSD，您应该预计集群 **不会** 在几个预期情况下回显 **HEALTH OK** ：

- 您尚未启动集群，也不会响应。
- 您刚刚启动或重新启动集群，但还没有就绪，因为 PG 已创建好，并且 OSD 正在对等。
- 您刚刚添加或删除了 OSD。
- 您刚刚修改了 cluster map。

监控 OSD 的一个重要方面是确保集群启动并运行 **in** 集群的所有 OSD 都为 **up** 并运行。

要查看所有 OSD 是否都在运行，请执行：

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

或者

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

结果应该告诉您 map epoch **eNNNN**、OSD 总数 **x**、数量为 **y**、以及 **up** 的数量是 **z** ： **in**

-

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

如果 **in** 的 OSD 数量超过 **up** OSD 的数量。执行以下命令，以确定未运行的 **ceph-osd** 守护进程：

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

示例

```
# 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
```

提示

通过能够按照设计良好的 CRUSH 层次结构搜索，可以帮助您更快地识别物理位置，从而对存储集群进行故障排除。

如果 OSD 为 **down**，请连接到节点并启动它。您可以使用红帽存储控制台重新启动 OSD 节点，也可以使用命令行。

示例

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

3.2.12. 其它资源

- *红帽 Ceph 存储仪表板指南.*

3.3. CEPH 存储集群的低级别监控

作为存储管理员，您可以从低层次监控红帽 Ceph 存储集群的健康状况。低级监控通常涉及确保 Ceph OSD 正确对等。当发生对等故障时，放置组以降级状态运行。这种降级状态可能是由许多不同的因素导致的，如硬件故障、挂起或崩溃的 Ceph 守护进程、网络延迟或完整站点中断。

3.3.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

3.3.2. 监控放置组集

当 CRUSH 将 PG 分配到 OSD 时，它会查看池的副本数量，并将 PG 分配到 OSD，使得 PG 的每一副本分配到不同的 OSD。例如，如果池需要 PG 的三个副本，CRUSH 可以分别将它们分配到 **osd.1**、**osd.2** 和 **osd.3**。CRUSH 实际上寻求伪随机放置，它将考虑您在 CRUSH map 中设置的故障域，因此很少会看到分配给大型集群中最接近邻居 OSD 的放置组。我们引用了应当包含特定放置组副本的 OSD **集合** 作为操作集合。在某些情况下，Acting Set 中的 OSD 是 **down**，否则无法服务 PG 中对象的请求。出现这些情况时，请不要 panic。常见示例包括：

- 您添加了或删除了 OSD。然后，CRUSH 将 PG 重新分配给其他 OSD- 从而更改操作集的构成并通过"回填"进程生成数据迁移。
- OSD 曾 **down** 重启，现在为 **recovering**。
- 操作集合中的 OSD 是 **down** 或无法为请求服务，另一个 OSD 暂时承担了自己的职责。

Ceph 利用 **Up Set** 处理客户端请求，这是将实际处理请求的 OSD 集合。在大多数情况下，启动集合和操作集几乎相同。如果没有，这可能表明 Ceph 正在迁移数据，或者 OSD 正在恢复，或者存在问题，即 Ceph 通常在这种情况下使用"stuck stale"消息回显 **HEALTH WARN** 状态。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 检索 PG 列表：

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

2. 查看操作集合中或给定 PG 的启动集合中有哪些 OSD：

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

结果应告知您 osdmap epoch、**eNNN**、PG 号、***PG_NUM***、Up Set **up[]** 中的 OSD，以及执行集合中的 OSD **acting[]** ：

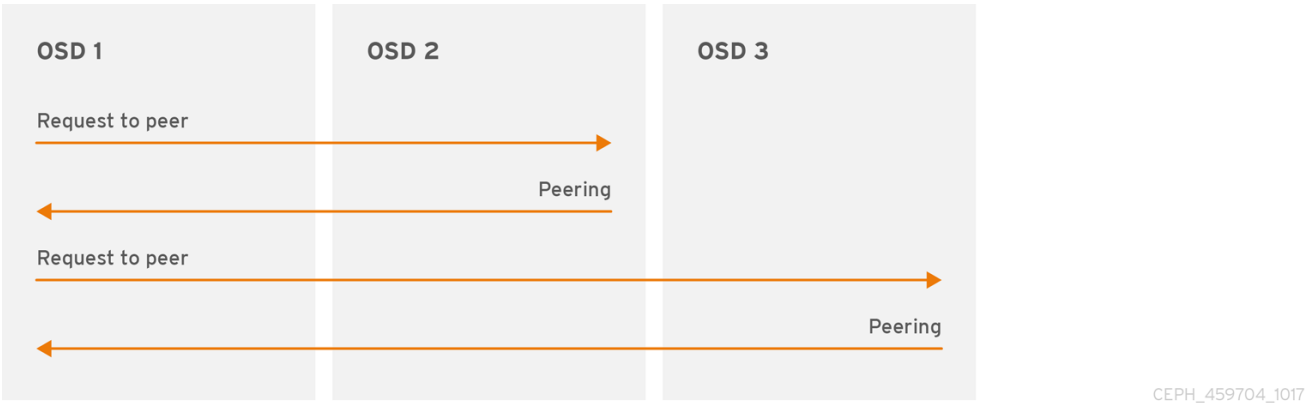
```
[root@mon ~]# ceph osdmap eNNN pg PG_NUM-> up [0,1,2] acting [0,1,2]
```

注意

如果 Up Set 和 Acting Set 不匹配，这可能表示集群重新平衡其自身或集群存在潜在问题。

3.3.3. Ceph OSD 对等

在将数据写入放置组前，它必须处于 **active** 状态，**它应该** 处于 **clean** 状态。供 Ceph 要确定 PG 的当前状态，即 PG 的Primary OSD，即操作集合中的第一个 OSD，以及与次要和第三 OSD 的对等 OSD，就 PG 的当前状态建立共识。假设池有 3 个 PG 副本。



3.3.4. 放置组状态

如果您执行 **ceph health**、**ceph -s** 或 **ceph -w** 等命令，您可能会注意到集群并不总是回显 **HEALTH OK**。在检查 OSD 是否在运行后，您也应检查 PG 状态。您应预计，在多个与 peering 相关的放置组中，集群 **不会** 回显 **HEALTH OK** ：

- 您刚刚创建了一个池和放置组，但尚未创建对等组。
- PG 正在恢复。
- 您刚刚将 OSD 添加到集群中或从集群中删除了 OSD。
- 您刚刚修改过 CRUSH map，而 PG 正在迁移。
- 放置组的不同副本中的数据不一致。
- Ceph 正在清理 PG 的副本。
- Ceph 没有足够的存储容量来完成回填操作。

如果其中一个情况导致 Ceph 回显 **HEALTH WARN**，请不要 panic。在很多情况下，集群会自行恢复。在某些情况下，您可能需要采取行动。监控 PG 的一个重要方面是确保在集群启动并运行时，所有放置组都是 **active**，最好是处于 **clean** 状态。

要查看所有 PG 的状态，请执行：

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

结果应该告诉您放置组映射版本 **vNNNNNN**、放置组总数 **x** 以及放置组 **y** 处于特定状态，如 **active+clean**:

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

注意

Ceph 通常报告 PG 的多个状态。

快照 Trimming PG 状态

存在快照时，将报告两个额外的 PG 状态。

- **snaptrim** : PG 目前被修剪
- **snaptrim_wait** : PG 等待修剪

输出示例：

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

除了放置组状态外，Ceph 还将回显所使用的数据量、**aa**、剩余存储容量量、**bb** 以及放置组的总存储容量。这些数据在少数情况下可能很重要：

- 您将到达 **near full ratio** 或 **full ratio**。

- 由于 CRUSH 配置中出现错误，您的数据不会在集群中分布。

放置组 ID

放置组 ID 由池编号而不是池名称组成，后跟句点(.)和放置组 ID-a 十六进制数字。您可以从 **ceph osd lspools** 的输出中查看池号及其名称。默认池名称 **data**、**metadata** 和 **rbd** 分别对应于池号 **0**、**1** 和 **2**。完全限定 PG ID 具有以下格式：

```
POOL_NUM.PG_ID
```

输出示例：

```
0.1f
```

- 检索 PG 列表：

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

- 以 JSON 格式格式化输出并将其保存到文件中：

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

- 查询特定放置组：

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

  JSON 格式的输出示例：

  ```
  {
    "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"
      }
    ]
}
```


其它资源

- 如需了解有关快照修剪设置的更多详细信息，请参阅红帽 Ceph 存储 4 *配置指南* 中的 *对象存储守护进程(OSD)配置选项*。

3.3.5. 放置组创建状态

创建池时，它将创建您指定的 PG 数量。在创建一个或多个 PG 时，Ceph 将回显 **creating**。创建之后，属于放置组操作集合一部分的 OSD 将对它们进行对等操作。完成 peering 后，放置组状态应为 **active+clean**，这意味着 Ceph 客户端可以开始写入 PG。

3.3.6. 放置组 peering 状态

当 Ceph 对 PG 进行对等时，Ceph 将存储 PG 副本的 OSD 放入 **关于 PG 中对象状态和元数据状态的协议**。当 Ceph 完成对等操作时，这表示存储 PG 的 OSD 同意 PG 的当前状态。但是，完成 peering 进程 **并不意味着** 每个副本都有最新的内容。

权威历史记录

Ceph 不会 **确认** 对客户端的写入操作，直到操作集的所有 OSD 都永久保留写入操作。这种做法可确保自上次成功对等操作以来，操作集合中至少有一个已确认写入操作的记录。

利用每个确认的写入操作的准确记录，Ceph 可以构建和重建 PG 的新权威历史记录。完整、全顺序的操作集合，如果执行此操作，可使 OSD 的副本保持最新状态。

3.3.7. 放置组活跃状态

当 Ceph 完成 peering 进程后，放置组可能会变为 **active**。**active** 状态意味着放置组中的数据通常位于主放置组中，以及用于读写操作的副本。

3.3.8. 放置组清理状态

当 PG 处于 **clean** 状态时，Primary OSD 和副本 OSD 已成功对等，且 PG 没有伪装的副本。Ceph 复制 PG 中所有对象的正确次数。

3.3.9. 放置组降级状态

当客户端将对象写入到Primary OSD 时，Primary OSD 负责将副本写入到副本 OSD。Primary OSD 将对象写入存储后，PG 将保持 **degraded** 状态，直到 Primary OSD 收到 Ceph 成功创建副本对象的副本 OSD 的确认。

PG 可以是 **active+degraded** 的原因是 OSD 可能为 **active**，即使它还没有保存所有对象。如果 OSD 传出 **down**，Ceph 会将分配给 OSD 的每个 PG 标记为 **degraded**。OSD 重新上线时，OSD 必须再次对等点。但是，如果客户端是 **active**，客户端仍然可以将新对象写入 **degraded** PG。

如果 OSD 是 **down**，且 **degraded** 条件仍然存在，Ceph 可以将 **down** OSD 标记为集群的 **out**，并将 **down** OSD 数据从 OSD 重新 map 到另一个 OSD。 **mon_osd_down_out_interval** 控制标记 **down** 和标记为 **out** 之间的时间，默认为 **600** 秒。

PG 也可以是 **degraded**，因为 Ceph 无法找到 Ceph 认为应在 PG 中的一个或多个对象。虽然您无法读取或写入未找到的对象，但您仍然可以访问 **degraded** PG 中的所有其他对象。

假设副本池中有 9 个 OSD。如果 OSD 编号 9 停机，分配给 OSD 9 的 PG 处于降级状态。如果 OSD 9 没有恢复，它会退出集群和集群重新平衡。在这种情况下，PG 会被降级，然后恢复到活动状态。

3.3.10. 放置组恢复状态

Ceph 专为容错而设计，其规模在于硬件和软件问题持续存在的规模。当 OSD 到达 **down** 时，其内容可能会低于 PG 中其他副本的当前状态。当 OSD 恢复 **up** 时，必须更新 PG 的内容来反映当前状态。在该期间内，OSD 可能会反映 **recovering** 状态。

恢复并不总是简单，因为硬件故障可能会导致多个 OSD 的级联故障。例如，一个机架或机柜的网络交换机可能会失败，这会导致多台主机计算机的 OSD 落于集群的当前状态后。故障解决后，每一 OSD 必须恢复。

Ceph 提供多个设置，以平衡新服务请求与恢复数据对象并将 PG 恢复到当前状态的需要之间的资源争用。**osd recovery delay start** 设置允许 OSD 在开始恢复过程前重启、重复甚至处理一些请求。**osd recovery threads** 设置限制恢复过程的线程数量，默认为一个线程。**osd recovery thread timeout** 设置线程超时，因为多个 OSD 可能会以加号的速度失败、重启和重新创建。**osd recovery max active** 设置限制 OSD 同时进入的恢复请求数量，以防止 OSD 无法服务。**osd recovery max chunk** 设置限制恢复的数据块的大小，以防止网络拥塞。

3.3.11. 回退状态

当新 OSD 加入集群时，CRUSH 会将 PG 从集群中的 OSD 重新分配到新添加的 OSD。强制新 OSD 立即接受重新分配的 PG 可能会给新 OSD 带来过多的负载。将 OSD 与 PG 回填可让此过程在后台开始。完成回填后，新 OSD 将在准备好时开始为请求提供服务。

在回填操作中，您可能看到以下状态之一：* **backfill_wait** 表示回填操作处于待处理状态，但 * **backfill** 表示正在进行回填操作。**backfill_too_full** 表示请求了回填操作，但因为存储容量不足而无法完成。

当无法回填 PG 时，可以将其视为 **incomplete**。

Ceph 提供了多个设置来管理与将 PG 重新分配到 OSD 相关的负载高峰，特别是新 OSD。默认情况下，**osd_max_backfills** 将 OSD 的最大并发回填操作数设置为 10。如果 OSD 接近满比率，则 OSD 能够拒绝回填请求，默认为 85%。**osd backfill full ratio**如果 OSD 拒绝回填请求，**osd backfill retry interval** 允许 OSD 在 10 秒后默认重试请求。OSD 也可以将 **osd backfill scan min** 和 **osd backfill scan max** 设置为管理扫描间隔，默认为 64 和 512。

对于某些工作负载，最好完全避免常规恢复，并使用回填。由于回填操作在后台进行，因此 I/O 能够继续执行 OSD 中的对象。要强制回填而不是恢复，将 **osd_min_pg_log_entries** 设置为 **1**，并将 **osd_max_pg_log_entries** 设置为 **2**。有关何时适合您的工作负载的详细信息，请联系您的红帽支持客户团队。

3.3.12. 更改恢复或回填操作的优先级

您可能会遇到这样的情形：某些放置组(PG)需要恢复和/或回填，其中一些 PG 包含的数据比其他放置组包含更重要的数据。使用 **pg force-recovery** 或 **pg force-backfill** 命令，确保优先级较高的 PG 首先进行恢复或回填。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 发出 **pg force-recovery** 或 **pg force-backfill** 命令，并为具有较高优先级数据的 PG 指定优先级顺序：

   语法

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

   示例

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

   此命令可使红帽 Ceph 存储首先在指定放置组(PG)上执行恢复或回填，然后再处理其他放置组。发出 命令不会中断当前执行的回填或恢复操作。在当前运行的操作完成后，将针对指定的 PG 尽快执行恢复或回填。

3.3.13. 在指定放置组上更改或取消恢复或回填操作

如果您取消了存储集群中某些 PG(PG)上的高优先级 **force-recovery** 或 **force-backfill** 操作，这些 PG 的操作将恢复到默认的恢复或回填设置。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 更改或取消指定 PG 上的恢复或回填操作：

   语法

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

   示例

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

   这会取消 **force** 标志，并以默认顺序处理 PG。

   完成指定 PG 的恢复或回填操作后，处理顺序将恢复为默认值。

其它资源

- 如需有关 RADOS 中恢复和回填操作优先级顺序的更多信息，请参阅 RADOS 中的 PG 优先级恢复和回填。

3.3.14. 为池强制高优先级恢复或回填操作

如果池中的所有 PG 需要高优先级恢复或回填，请使用 **force-recovery** 或 **force-backfill** 选项来启动操作。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

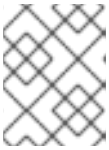
1. 在指定池中的所有 PG 上强制进行高优先级恢复或回填：

   语法

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

   示例

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



注意

请谨慎使用 **force-recovery** 和 **force-backfill** 命令。更改这些操作的优先级可能会破坏 Ceph 内部优先级计算的顺序。

3.3.15. 取消池的高优先级恢复或回填操作

如果您取消池中所有 PG 的高优先级 **force-recovery** 或 **force-backfill** 操作，则池中 PG 的操作将恢复到默认的恢复或回填设置。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 对指定池中所有放置组取消高优先级恢复或回填操作：

   语法

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

   示例

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

3.3.16. 重新安排池的恢复或回填操作的优先级

如果您有多个池当前使用相同的底层 OSD 和某些池包含高优先级数据，您可以重新调整执行操作的顺序。使用 **recovery_priority** 选项为优先级较高的池分配更高的优先级值。这些池将在优先级较低的池或设置为默认优先级的池之前执行。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 重新排列池的恢复/回填优先级：

   语法

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

   示例

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

   VALUE 设置优先级顺序。例如，如果您有 10 个池，优先级为 10 的池会首先被处理，后面是优先级为 9 的池，以此类推。如果只有某些池具有高优先级，您可以仅为这些池设置优先级值。没有设置优先级值的池按照默认顺序处理。

3.3.17. RADOS 中 PG 恢复的优先级

本节介绍 RADOS 中 PG 的恢复和回填的相对优先级值。首先处理更高的值。Inactive PG 的优先级高于活跃或降级 PG。

操作	值	描述
OSD_RECOVERY_PRIORITY_MIN	0	最小恢复值
OSD_BACKFILL_PRIORITY_BASE	100	MBackfillReserve 的回填优先级
OSD_BACKFILL_DEGRADED_PRIORITY_BASE	140	MBackfillReserve（降级 PG）的基础回填优先级.
OSD_RECOVERY_PRIORITY_BASE	180	MBackfillReserve 的基本恢复优先级
OSD_BACKFILL_INACTIVE_PRIORITY_BASE	220	MBackfillReserve（不活跃 PG）的基准回填优先级.
OSD_RECOVERY_INACTIVE_PRIORITY_BASE	220	MRecoveryReserve（不活跃 PG）的基本恢复优先级.
OSD_RECOVERY_PRIORITY_MAX	253	Max 为 MBackfillReserve 手动/自动设置恢复优先级
OSD_BACKFILL_PRIORITY_FORCED	254	手动强制时，MBackfillReserve 的回填优先级
OSD_RECOVERY_PRIORITY_FORCED	255	MRecoveryReserve 恢复优先级，当强制时
OSD_DELETE_PRIORITY_NORMAL	179	OSD 不完整时删除 PG 的优先级
OSD_DELETE_PRIORITY_FULLISH	219	OSD 接近满时删除 PG 的优先级
OSD_DELETE_PRIORITY_FULL	255	OSD 满时删除的优先级

3.3.18. 放置组重新 map 状态

当 Acting Set 为放置组提供服务时，数据将从旧操作集迁移到新的操作集。可能需要过些时间，新的 Primary OSD 才能为请求服务。因此，它可能会要求旧主继续服务请求，直到 PG 迁移完成。数据迁移完成后，映射将使用新操作集合的 Primary OSD。

3.3.19. 放置组过时状态

虽然 Ceph 使用 heartbeats 来确保主机和守护进程正在运行，但 **ceph-osd** 守护进程也可能进入 **stuck**

状态，它们无法及时报告统计数据。例如，临时网络故障。默认情况下，OSD 守护进程每半秒报告其放置组、thru、引导和失败统计信息，即 **0.5**，比心跳阈值更频繁。如果 PG 执行集合的 **Primary OSD** 无法报告给 monitor，或者其他 OSD 报告了Primary OSD **down**，则 monitor 将标记 PG **stale**。

当您启动存储集群时，通常会查看 **stale** 状态，直到对等过程完成为止。在存储集群运行一段时间后，查看处于 **stale** 状态的 PG 表示这些 PG 的 Primary OSD 为 **down** 或者没有向 monitor 报告 PG 统计信息。

3.3.20. 放置组错误设置的状态

有一些临时的回填方案是 PG 临时映射到 OSD。当 **temporary** 情况应该不再如此时，PG 可能仍然驻留在临时位置，而不是位于正确的位置。在这种情况下，它们被称为 **misplaced**。这是因为实际存在正确数量的额外副本，但一个或多个副本位于错误的位置。

例如，有 3 个 OSD：0、1、2 和所有 PG map 到这三个 OSD 的一些变异。如果添加另一个 OSD(OSD 3)，一些 PG 现在将 map 到 OSD 3，而不是另一个 PG。但是，在 OSD 3 回填之前，PG 将具有一个临时映射，使得 PG 能够继续从旧 map 为 I/O 服务。在该时间里，PG 是 **misplaced**，因为它有一个临时映射，而非 **degraded**，因为存在 3 个副本。

示例

```
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] 是一个临时映射，因此 **up** 集不等于 **acting** 集，而且 PG 为 **misplaced**，但不是 **degraded**，因为 [0,1,2] 仍有三个副本。

示例

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

OSD 3 现已回填，临时映射已被删除，没有降级且不会被错误地放置。

3.3.21. 放置组不完整状态

当内容不完整且 peering 失败时，PG 进入 **incomplete** 状态，即当没有足够完整的 OSD 来执行恢复时。

我们假设 OSD 1、2 和 3 是活动的 OSD 集，并且它切换到 OSD 1、4 和 3，那么 **osd.1** 将请求在回填 4 时请求 OSD 1、2 和 3 的临时操作集合。在此过程中，如果 OSD 1、2 和 3 都停机，**osd.4** 将是可能没有完全回填所有数据的唯一遗留数据。目前，PG 将转到 **incomplete**，表示没有足够完整的 OSD 来执行恢复。

或者，如果 **osd.4** 未涉及，并且操作的集合只是 OSD 1、2 和 3 时，当 OSD 1、2 和 3 停机时，PG 可能会转到 **stale**，表示该 PG 自操作集更改后没有任何消息。造成不存在向新 OSD 通知的 OSD 的原因。

3.3.22. 识别卡住 PG

如前文所述，放置组不一定只是一个问题，因为它的状态不是 **active+clean**。通常，Ceph 自我修复的功能在 PG 卡住时可能无法正常工作。卡住的状态包括：

- **unclean** ：放置组包含不会复制所需次数的对象。它们应该正在恢复。
- **Inactive** ：放置组无法处理读取和写入，因为它们正在等待具有最新数据的 OSD 返回 **up**。

- **stale** ： PG 处于未知状态，因为托管它们的 OSD 暂时尚未报告给 monitor 集群，并且可以使用 **mon osd report timeout** 设置进行配置。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要识别卡住的 PG，请执行以下操作：

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

3.3.23. 查找对象的位置

Ceph 客户端检索最新的群集映射，CRUSH 算法计算如何将对象 map 到 PG，然后计算如何将 PG 动态分配到 OSD。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要查找对象位置，您需要的是对象名称和池名称：

```
ceph osd map POOL_NAME OBJECT_NAME
```

第 4 章 覆盖 CEPH 行为

作为存储管理员，您需要了解如何将覆盖用于红帽 Ceph 存储群集，以在运行时期间更改 Ceph 选项。

4.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

4.2. 设置和取消设置 CEPH 覆盖选项

您可以设置和取消设置 Ceph 选项来覆盖 Ceph 的默认行为。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要覆盖 Ceph 的默认行为，请使用 **ceph osd set** 命令和您要覆盖的行为：

   ```
   ceph osd set FLAG
   ```

   设置行为后，**ceph health** 将反映您为集群设置的覆盖。

2. 要停止覆盖 Ceph 的默认行为，请使用 **ceph osd unset** 命令和您要停止的覆盖。

   ```
   ceph osd unset FLAG
   ```

标志	描述
noin	防止 OSD 被视为 **in** 集群。
noout	防止 OSD 被视为集群的 **out**。
noup	防止 OSD 被视为 **up** 并在运行。
nodown	防止 OSD 被视为 **down**。
full	使集群似乎已到达其 **full_ratio**，从而防止写入操作。
pause	Ceph 将停止处理读写操作，但不会影响 OSD **in**、**out**、**up** 或 **down** 状态。
nobackfill	Ceph 将阻止新的回填操作。
norebalance	Ceph 将阻止新的重新平衡操作。
norecover	Ceph 将阻止新的恢复操作。

标志	描述
noscrub	Ceph 将阻止新的清理操作。
nodeep-scrub	Ceph 将防止新的深度清理操作。
notieragent	Ceph 将禁用正在寻找冷/脏对象来清空和驱除的进程。

4.3. CEPH 覆盖用例

- **noin**：通常与 **noout** 搭配使用，以解决闪烁 OSD。
- **noout**: 如果超过 **mon osd report timeout** 并且 OSD 没有向 monitor 报告，则 OSD 将获得标记为 **out**。如果出现错误，您可以设置 **noout** 来防止 OSD 在您对问题进行故障排除时被标记为 **out**。
- **noup**：通常与 **nodown** 搭配使用，以解决闪烁 OSD。
- **nodown**: 网络问题可能会中断 Ceph 'heartbeat' 进程，而 OSD 可能是 **up**，但仍被标记为 down。您可以设置 **nodown** 来防止 OSD 在进行故障排除时被标记为 down。
- **full**: 如果集群要到达其 **full_ratio**，您可以抢占地将集群设置为 **full** 并扩展容量。

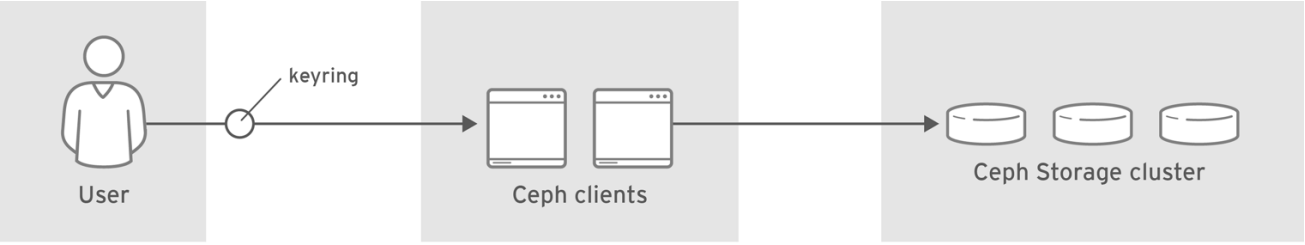
注意

将集群设置为 **full** 将阻止写入操作。

- **pause**: 如果您需要在没有客户端读写数据的情况下对正在运行的 Ceph 集群进行故障排除，您可以将集群设置为 **pause**，以防止客户端操作。
- **nobackfill**: 如果您需要临时取 OSD 或节点 **down**，例如升级守护进程，您可以设置 **nobackfill**，以便 Ceph 不会在 OSD 为 **down** 时回填。
- **norecover**：如果您需要替换 OSD 磁盘，并且不希望 PG 在热插拔磁盘期间恢复到另一个 OSD，您可以设置 **norecover** 以防止其他 OSD 将新 PG 复制到其他 OSD。
- **noscrub** 和 **nodeep-scrubb** ：例如，如果要防止清理，降低高负载、恢复、回填和重新平衡期间的开销，您可以设置 **noscrub** 和/或 **nodeep-scrub** 来防止集群清理 OSD。
- **notieragent**: 如果您要停止层代理进程，从查找冷对象到清除到后备存储层，您可以设置 **notieragent**。

第 5 章 CEPH 用户管理

作为存储管理员，您可以通过为红帽 Ceph 存储集群中的对象提供身份验证、密钥环管理和访问控制来管理 Ceph 用户基础。



5.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.
- 访问 Ceph 监控器或 Ceph 客户端节点.

5.2. CEPH 用户管理背景

当 Ceph 在启用身份验证和授权的情况下运行时，您必须指定包含指定用户的 secret key 的用户名和密钥环。如果不指定用户名，Ceph 将使用 **client.admin** 管理用户作为默认用户名。如果没有指定密钥环，Ceph 将使用 Ceph 配置中的 **keyring** 设置来查找密钥环。例如：如果您在没有指定用户或密钥环的情况下执行 **ceph health** 命令：

```
# ceph health
```

Ceph 将该命令解释如下：

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

另外，您可以使用 **CEPH_ARGS** 环境变量以避免重新输入用户名和 secret。

无论 Ceph 客户端的类型如何，如块设备、对象存储、文件系统、原生 API 或 Ceph 命令行，Ceph 将所有数据存储为池内的对象。Ceph 用户必须有权访问池，才能读取和写入数据。此外，管理 Ceph 用户必须具有执行 Ceph 管理命令的权限。

以下概念可帮助您了解 Ceph 用户管理：

存储集群用户

红帽 Ceph 存储集群的用户是个人或应用程序。通过创建用户，您可以控制谁可以访问存储集群、其池以及这些池中的数据。

Ceph 具有用户 **type** 的概念。出于用户管理的目的，类型将始终为 **client**。Ceph 以句点(.)分隔形式标识用户，其由用户类型和用户 ID 组成。例如 **TYPE.ID**、**client.admin** 或 **client.user1**。用户键入的原因在于 Ceph 监控器和 OSD 也使用 Cephx 协议，但它们不是客户端。区分用户类型有助于区分客户端用户和其他用户 - 提取访问控制、用户监控和可追溯性。

有时，Ceph 的用户类型可能会看上去有些混淆，因为 Ceph 命令行允许您根据命令行使用情况来指定包含或不使用类型的用户。如果指定了 **--user** 或 **--id**，您可以省略 类型。因此 **client.user1** 只需输入为 **user1**。如果指定了 **--name** 或 **-n**，您必须指定类型和名称，如 **client.user1**。红帽建议尽可能将类型和

名称用作最佳实践。

注意

红帽 Ceph 存储集群用户与 Ceph 对象网关用户不同。对象网关使用红帽 Ceph 存储集群用户在网关守护进程和存储集群之间进行通信，但该网关对其最终用户有自己的用户管理功能。

授权功能

Ceph 使用术语"功能"（容量）来描述授权经过身份验证的用户来练习 Ceph 监视器和 OSD 的功能。功能还可以限制对池中数据或池中命名空间的数据的访问。Ceph 管理用户在创建或更新用户时设置用户的功能。功能语法采用以下形式：

语法

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

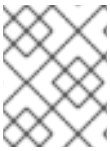
- **monitor Caps：** 监控器功能包括 **r**、**w**、**x**、**allow profile *CAP*** 和 **profile rbd**。

  示例

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

- **OSDCaps：** OSD 功能包括 **r**、**w**、**x**、**class-read**、**class-write**、**profile osd**、**profile rbd** 和 **profile rbd-read-only**。此外，OSD 功能也允许对池和命名空间设置。

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



注意

Ceph 对象网关守护进程(**radosgw**)是 Ceph 存储集群的客户端，因此它不表示为 Ceph 存储集群守护进程类型。

以下条目描述了每种功能：

allow	在守护进程的访问设置之前。
r	授予用户读取访问权限。需要使用 monitor 来检索 CRUSH map。
w	授予用户对对象的写入访问权限。
x	为用户提供调用类方法（即读取和写入）并对 monitor 执行 **auth** 操作的功能。
class-read	赋予用户调用类读取方法的功能。**x** 的子集。
class-write	赋予用户调用类写入方法的功能。**x** 的子集。
*	授予用户特定守护进程或池的读取、写入和执行权限，以及执行 admin 命令的功能。

profile osd	授予用户权限，以 OSD 身份连接到其他 OSD 或 monitor。调用 OSD，使 OSD 能够处理复制心跳流量和状态报告。
profile bootstrap-osd	为用户授予引导 OSD 的权限，以便他们在引导 OSD 时具有添加密钥的权限。
profile rbd	授予用户对 Ceph 块设备的读写访问权限。
profile rbd-read-only	授予用户对 Ceph 块设备的只读访问权限。

池

池为 Ceph 客户端定义存储策略，并充当该策略的逻辑分区。

在 Ceph 部署中，通常要创建一个池来支持不同类型的用例。例如，云卷或镜像、对象存储、热存储和冷存储等。将 Ceph 部署为 OpenStack 的后端时，典型的部署具有卷、镜像、备份和虚拟机等用户的池，以及 **client.glance**、**client.cinder** 等用户。

命名空间

池中的对象可以关联到池中的 namespace-a 逻辑对象组。用户对池的访问权限可以与命名空间关联，这样用户的读取和写入只能在命名空间内进行。写入到池中命名空间的对象只能由有权访问命名空间的用户访问。

注意

目前，命名空间仅适用于在 **librados** 上写入的应用程序。块设备和对象存储等 Ceph 客户端目前不支持此功能。

命名空间的理由是池可以是按用例分隔数据的计算方式，因为每个池都会创建一组映射到 OSD 的 PG。如果多个池使用相同的 CRUSH 层次结构和规则集，OSD 性能可能会随着负载的增加而降级。

例如，池应当为每个 OSD 拥有大约 100 个 PG。因此，具有 1000 个 OSD 的示范集群将具有 100,000 个池的放置组。映射到同一 CRUSH 层次结构和规则集的每个池都会在示例集群中再创建 100,000 个放置组。相比之下，将对象写入命名空间只是将命名空间与对象名称关联，与单独池的计算开销无关。您可以使用命名空间，而不是为用户或一组用户创建单独的池。

注意

目前只能使用 **librados**。

其它资源

- 有关 *配置身份验证的详细信息，请参阅红帽 Ceph 存储配置指南*。

5.3. 管理 CEPH 用户

作为存储管理员，您可以通过创建、修改、删除和导入用户来管理 Ceph 用户。Ceph 客户端用户可以是个人或应用，它们使用 Ceph 客户端与红帽 Ceph 存储集群守护进程交互。

5.3.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.
- 访问 Ceph 监控器或 Ceph 客户端节点.

5.3.2. 列出 Ceph 用户

您可以使用命令行界面列出存储集群中的用户。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要列出存储集群中的用户，请执行以下操作：

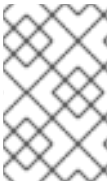
```
[root@mon ~]# ceph auth list
```

Ceph 将列出存储群集中的所有用户。例如，在一个双节点示例存储集群中，**ceph auth list** 将输出类似如下的内容：

示例

```
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
```



注意

对用户应用 **TYPE.ID** 表示法，**osd.0** 是类型为 **osd** 的用户，其 ID 为 **0**，**client.admin** 是类型为 **client** 的用户，其 ID 为 **admin**，即默认的 **client.admin** 用户。另请注意，每个条目都有 **key:** ***VALUE*** 条目，以及一个或多个 **caps:** 条目。

您可以在 **ceph auth list** 中使用 **-o *FILE_NAME*** 选项将输出保存到文件中。

5.3.3. 显示 Ceph 用户信息

您可以使用 命令行界面显示 Ceph 的用户信息。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要检索特定用户、密钥和功能，请执行以下操作：

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

示例

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

2. 您还可以在 **ceph auth get** 中使用 **-o *FILE_NAME*** 选项将输出保存到文件中。开发人员也可以执行以下操作：

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

示例

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

auth export 命令与 **auth get** 相同，但也会打印出与最终用户无关的内部 **auid**。

5.3.4. 添加新 Ceph 用户

添加用户会创建一个用户名，即 **TYPE.ID**，这是 secret 密钥以及您用来创建用户的命令中包含的任何功能。

用户的密钥使用户能够通过 Ceph 存储集群进行身份验证。用户的功能授权用户读取、写入或执行 Ceph 监视器(**mon**)、Ceph OSD(**osd**)或 Ceph 元数据服务器(**mds**)。

添加用户有几个方法：

- **ceph auth add**：此命令是添加用户的规范方式。它将创建用户，生成密钥并添加任何指定的功能。
- **ceph auth get-or-create**：此命令通常是创建用户的最简便方式，因为它返回了含有用户名（括在方括号中）和密钥的密钥文件格式。如果用户已存在，此命令只需以 keyfile 格式返回用户名和密钥。您可以使用 **-o *FILE_NAME*** 选项将输出保存到文件中。
- **ceph auth get-or-create-key**: 此命令是创建用户并仅返回用户的密钥的一种便捷方式。这只对只需要密钥的客户端有用，例如 **libvirt**。如果用户已存在，此命令只需返回密钥。您可以使用 **-o *FILE_NAME*** 选项将输出保存到文件中。

在创建客户端用户时，您可以创建没有能力的用户。除了仅仅通过身份验证外，没有功能的用户是无用的，因为客户端无法从监控器检索集群映射。但是，如果您希望以后使用 **ceph auth caps** 命令延迟添加功能，您可以创建没有功能的用户。

典型的用户至少具有 Ceph 监控器的读取功能，以及 Ceph OSD 的读取和写入功能。此外，用户的 OSD 权限通常仅限于访问特定的池。

```
[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

如果您为用户提供 OSD 的功能，但您不限制对特定池的访问，该用户将有权访问集群中的 ALL 池！

5.3.5. 修改 Ceph 用户

ceph auth caps 命令允许您指定用户并更改用户的功能。设置新功能将覆盖当前的功能。因此，首先查看当前功能并在您添加新功能时包括它们。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 查看当前功能：

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

   示例

   ```
   [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. 要添加功能，请使用表单：

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

   示例

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

在示例中，添加了 OSD 上的执行能力。

3. 验证添加的功能：

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

示例

```
[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"
```

在示例中，可以看到 OSD 上的执行能力。

4. 要移除某一功能，请设置所有当前功能，但您要移除的功能除外。

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

示例

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

在示例中，OSD 上的执行能力未包含在内，因此将被移除。

5. 验证删除的功能：

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

示例

```
[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"
```

在示例中，OSD 上的执行能力不再列出。

其它资源

- 有关 *功能的更多详细信息，请参阅授权* 功能。

5.3.6. 删除 Ceph 用户

您可以使用命令行界面从 Ceph 存储集群中删除用户。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要删除用户，请使用 **ceph auth del** ：

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

其中 ***TYPE*** 是 **client**、**osd**、**mon** 或 **mds** 之一，***ID*** 是守护进程的用户名或 ID。

5.3.7. 打印 Ceph 用户密钥

您可以使用命令行界面显示 Ceph 用户的关键信息。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要将用户的身份验证密钥打印到标准输出，请执行以下操作：

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

其中 ***TYPE*** 是 **client**、**osd**、**mon** 或 **mds** 之一，***ID*** 是守护进程的用户名或 ID。

2. 当您需要使用用户的密钥（例如 **libvirt** ）填充客户端软件时，打印用户的密钥非常有用。

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

5.3.8. 导入 Ceph 用户

您可以使用命令行界面导入 Ceph 用户。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要导入一个或多个用户，请使用 **ceph auth import** 并指定一个密钥环：

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

示例

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

注意

Ceph 存储集群将添加新用户、其密钥和功能，并将更新现有用户、其密钥及其功能。

5.4. 管理 CEPH 密钥环

作为存储管理员，管理 Ceph 用户密钥对于访问红帽 Ceph 存储集群非常重要。您可以创建密钥环，将用户添加到密钥环，并使用密钥环修改用户。

5.4.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.
- 访问 Ceph 监控器或 Ceph 客户端节点.

5.4.2. 创建密钥环

您需要向 Ceph 客户端提供用户密钥，以便 Ceph 客户端能够检索指定用户的密钥，并与 Ceph 存储群集进行身份验证。Ceph 客户端通过访问密钥环来查找用户名并检索用户的密钥。

ceph-authtool 工具允许您创建密钥环。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要创建空密钥环，请使用 **--create-keyring** 或 **-C**。

   示例

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

   使用多个用户创建密钥环时，我们建议使用集群名称。例如：用于密钥环文件名的 ***CLUSTER_NAME*.keyring`**，并将其保存在 **/etc/ceph/** 目录中，以便 **keyring** 配置默认设置将获取文件名而无需您在 Ceph 配置文件的本地副本中指定它。

2. 通过执行以下操作来创建 **ceph.keyring** ：

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

当使用单个用户创建密钥环时，我们建议使用集群名称、用户类型和用户名，并将其保存在 **/etc/ceph/** 目录中。例如，**client.admin** 用户的 **ceph.client.admin.keyring**。

要在 **/etc/ceph/** 中创建密钥环，您必须使用 **root**。这意味着该文件将只为 **root** 用户具有 **rw** 权限，这在密钥环包含管理员密钥时适用。但是，如果您要为特定用户或用户组使用密钥环，请确保执行 **chown** 或 **chmod** 来建立适当的密钥环所有权和访问权限。

5.4.3. 将用户添加到密钥环

当您向 Ceph 存储集群中添加用户时，您可以使用 **get** 流程检索用户、密钥和功能，然后将用户保存到密钥环文件中。当您只想在每个密钥环使用一个用户时，带有 **-o** 选项的 *Display Ceph 用户信息* 过程将以 keyring 文件格式保存输出。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要为 **client.admin** 用户创建密钥环，请执行以下操作：

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

   请注意，我们为单个用户使用推荐的文件格式。

2. 当您要将用户导入到密钥环时，您可以使用 **ceph-authtool** 指定目标密钥环和源密钥环。

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

5.4.4. 创建具有密钥环的 Ceph 用户

Ceph 提供直接在红帽 Ceph 存储集群中创建用户的功能。但是，您也可以直接在 Ceph 客户端密钥环上创建用户、密钥和功能。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 将用户导入到密钥环中：

   示例

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

2. 创建密钥环并同时添加新用户到密钥环中：

   例如：

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

在以下情况下，新用户 **client.ringo** 只在密钥环中。

3. 将新用户添加到 Ceph 存储集群：

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

其它资源

- 有关功能的更多详细信息，请参阅 Ceph 用户管理背景。

5.4.5. 使用密钥环修改 Ceph 用户

您可以使用命令行界面修改 Ceph 用户及其密钥环。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 要在密钥环中修改用户记录的功能，请指定密钥环和用户后面的功能，例如：

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

1. 要将用户更新至 Red Hat Ceph Storage 集群，您必须将密钥环中的用户更新为 Red Hat Ceph Storage 集群中的用户条目：

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

您还可以直接在存储集群中修改用户功能，将结果保存到密钥环文件中，然后将密钥环导入到主 **ceph.keyring** 文件中。

其它资源

- 如需有关从密钥环更新 Red Hat Ceph Storage 集群用户的详细信息，*请参阅导入* 用户。

5.4.6. Ceph 用户的命令行使用

Ceph 支持将以下用户名和 secret 用于用户名和 secret：

--id | **--user**

描述

Ceph 使用类型和 ID 识别用户。例如 **TYPE.ID** 或 **client.admin**、**client.user1**。**id**、**name** 和 **-n** 选项允许您指定用户名的 ID 部分。例如 **admin**、**user1** 或 **foo**。您可以使用 **--id** 指定用户并省略类型。例如，要指定用户 **client.foo**，请输入以下内容：

-

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

--name | **-n**

描述

Ceph 使用类型和 ID 识别用户。例如 **TYPE.ID** 或 **client.admin**、**client.user1**。**--name** 和 **-n** 选项允许您指定完全限定用户名。您必须使用用户 ID 指定用户类型（通常为 **client**）。例如：

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

--keyring

描述

包含一个或多个用户名和 secret 的密钥环的路径。**--secret** 选项提供相同的功能，但它不适用于 Ceph RADOS 网关，它使用 **--secret** 来满足另一个目的。您可以使用 **ceph auth get-or-create** 检索密钥环，并将其存储在本地。这是首选的方法，因为您可以切换用户名而不切换密钥环路径。例如：

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

5.4.7. Ceph 用户管理限制

cephx 协议相互验证 Ceph 客户端和服务器。它不是为了处理人工用户或应用程序程序代表他们运行的验证。如果需要这种效果才能处理访问控制需求，您必须有另一种机制，这可能特定于用于访问 Ceph 对象存储的前端。这种其他机制的作用是确保只有可接受的用户和程序才能在 Ceph 允许访问其对象存储的计算机上运行。

用于对 Ceph 客户端和服务器进行身份验证的密钥通常存储在具有受信任主机中适当权限的纯文本文件中。

重要

将密钥存储在纯文本文件中存在安全缺点，但是考虑到 Ceph 在后台使用的基本身份验证方法，它们很难避免。这些设置 Ceph 系统应清楚这些缺点。

特别是，任意用户计算机（尤其是便携式计算机）不应配置为直接与 Ceph 交互，因为使用模式需要在不安全的计算机上存储纯文本身份验证密钥。欺骗该计算机或获取对其无与伦比的访问的任何人都可以获取允许其自己的计算机身份验证到 Ceph 的密钥。

用户不必允许潜在的不安全计算机直接访问 Ceph 对象存储，而是要求用户使用为这些用途提供足够安全性的方法登录环境中受信任的计算机。该可信计算机将存储人类用户的纯文本 Ceph 密钥。Ceph 的未来版本可能会更加彻底地解决这些特定的身份验证问题。

目前，任何 Ceph 身份验证协议都没有为传输中的消息提供保密。因此，窃听线路可以听到和理解 Ceph 中客户端和服务器之间发送的所有数据，即使他无法创建或更改它们。在 Ceph 中存储敏感数据的用户应考虑加密其数据，然后将其提供给 Ceph 系统。

例如，Ceph 对象网关提供 S3 API 服务器端加密，它将加密从 Ceph 对象网关客户端接收的未加密数据，然后再将它存储在 Ceph 存储集群中，并且类似地解密从 Ceph 存储集群检索的数据，再将其发回到客户端。为确保客户端和 Ceph 对象网关之间的传输加密，Ceph 对象网关应配置为使用 SSL。

第 6 章 CEPH-VOLUME 工具

作为存储管理员，您可以使用 **ceph-volume** 实用程序准备、创建和激活 Ceph OSD。 **ceph-volume** 工具是一个目的命令行工具，用于将逻辑卷部署为 OSD。它使用插件类型框架来部署具有不同设备技术的 OSD。**ceph-volume** 实用程序遵循与 **ceph-disk** 实用程序类似的工作流，用于部署 OSD，采用可预测且强大的方法来准备、激活和启动 OSD。目前，**ceph-volume** 工具只支持 **lvm** 插件，并计划将来支持其他技术。

重要

ceph-disk 命令已弃用。

6.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

6.2. CEPH 卷 LVM 插件

通过使用 LVM 标签，**lvm** 子命令可以通过查询与 OSD 关联的设备来存储和重新发现它们，以便激活它们。这包括对基于 lvm 的技术（如 **dm-cache** ）的支持。

使用 **ceph-volume** 时，使用 **dm-cache** 是透明的，把 **dm-cache** 视为逻辑卷。使用 **dm-cache** 时的性能增益和损失取决于特定工作负载。通常，随机和顺序读取会在较小的块大小下看到性能增加。虽然随机和顺序写入的性能在较大块大小时会降低。

要使用 LVM 插件，在 **ceph-volume** 命令中添加 **lvm** 作为子命令：

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

lvm 子命令有三个子命令，如下所示：

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

注意

使用 **create** 子命令，将 **prepare** 和 **activate** 子命令组合为一个子命令。

其它资源

- 详情请查看 **create** 子命令 部分。

6.3. 为什么 CEPH-VOLUME 替换 CEPH-DISK？

早期版本的红帽 Ceph 存储使用 **ceph-disk** 实用程序来准备、激活和创建 OSD。从红帽 Ceph 存储 4 开始，**ceph-disk** 被 **ceph-volume** 工具替代，它旨在作为单一目的命令行工具将逻辑卷部署为 OSD，同时在准备、激活和创建 OSD 时维护一个与 **ceph-disk** 类似的 API。

ceph-volume 是如何工作的？

ceph-volume 是一个模块化工具，目前支持通过两种方式置备硬件设备：传统的 **ceph-disk** 设备和 LVM（逻辑卷管理器）设备。**ceph-volume lvm** 命令使用 LVM 标签存储关于 Ceph 特定设备及其与 OSD 的关系的信息。它使用这些标签来稍后重新发现和查询与 OSDS 关联的设备，以便它可以激活它们。它还支持基于 LVM 和 **dm-cache** 的技术。

ceph-volume 工具以 **dm-cache** 透明方式使用，并将其视为逻辑卷。使用 **dm-cache** 时，您可能会考虑性能的提高和损失，具体取决于您要处理的特定工作负载。通常，随机和顺序读取操作的性能在块大小较小时会增加；而随机和顺序写操作的性能在更大的块大小上会降低。使用 **ceph-volume** 不会引入显著的性能限制。

重要

ceph-disk 工具已弃用。

注意

如果这些设备仍在使用，**ceph-volume simple** 命令可以处理旧的 **ceph-disk** 设备。

ceph-disk 是如何工作的？

ceph-disk 工具需要支持许多不同类型的 init 系统，如 **upstart** 或 **sysvinit**，同时可以发现设备。因此，**ceph-disk** 只关注 GUID 分区表(GPT)分区。特别在 GPT GUID 中，该 GUID 以独特的方式标记设备来回答以下问题，例如：

- 这个设备是 **journal** 吗？
- 该设备是否加密数据分区？
- 设备是否部分准备就绪？

要解决这些问题，**ceph-disk** 使用 UDEV 规则来匹配 GUID。

使用 ceph-disk 有哪些缺点？

使用 UDEV 规则调用 **ceph-disk** 可能会导致 **ceph-disk systemd** 单元和 **ceph-disk** 可执行文件之间的后退。这个过程非常不可靠且耗时，并可能导致 OSD 在节点的引导过程中根本不出现。此外，根据 UDEV 的异步行为，调试甚至无法复制这些问题。

因为 **ceph-disk** 只可用于 GPT 分区，所以不支持其他技术，如逻辑卷管理器(LVM)卷或类似的设备映射程序设备。

要确保 GPT 分区与设备发现工作流可以正常工作，**ceph-disk** 需要使用大量特殊标记。此外，这些分区要求设备完全归 Ceph 所有。

6.4. 使用准备 CEPH OSD CEPH-VOLUME

prepare 子命令为 OSD 数据和日志准备 OSD 后端对象存储，并使用逻辑卷(LV)。除了使用 LVM 添加一些额外的元数据标签外，它不会修改逻辑卷。这些标签使卷更易于发现，它们也会将卷识别为 Ceph 存储集群的一部分，以及这些卷在存储集群中的角色。

BlueStore OSD 后端支持以下配置：

- 块设备、**block.wal** 设备和 **block.db** 设备

- 块设备和 **block.wal** 设备
- 块设备和 **block.db** 设备
- 单个块设备

prepare 子命令接受整个设备或分区，或 **block** 的逻辑卷。

先决条件

- OSD 节点的根级别访问权限.
- （可选）创建逻辑卷。如果您提供物理设备的路径，子命令将设备转换为逻辑卷。这种方法更为简单，但您无法配置或更改逻辑卷的创建方式。

流程

1. 准备 LVM 卷：

   语法

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

   示例

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

   a. 另外，如果您要为 RocksDB 使用单独的设备，请指定 **--block.db** 和 **--block.wal** 选项：

      语法

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

      示例

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

   b. 另外，要加密数据，请使用 **--dmcrypt** 标志：

      语法

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

      示例

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

其它资源

- 如需了解更多详细信息，请参阅《*红帽 Ceph 存储管理指南》中的使用"ceph-volume"激活 CephOSD 部分*。
- 如需了解更多详细信息，*请参阅《红帽 Ceph 存储管理指南》中的使用"ceph-volume"创建 Ceph OSD* 部分。

6.5. 使用激活 CEPH OSD CEPH-VOLUME

激活过程在引导时启用 **systemd** 单元，允许启用并挂载正确的 OSD 标识符及其 UUID。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- Ceph OSD 节点的根级别访问权限.
- Ceph OSD 由 **ceph-volume** 实用程序准备。

流程

1. 从 OSD 节点获取 OSD ID 和 UUID：

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

2. 激活 OSD：

语法

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

示例

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

要激活准备激活的所有 OSD，使用 **--all** 选项：

示例

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

其它资源

- 如需了解更多详细信息，请参阅《*红帽 Ceph 存储管理指南》中的使用"ceph-volume"准备 CephOSD 部分*。
- 如需了解更多详细信息，*请参阅《红帽 Ceph 存储管理指南》中的使用"ceph-volume"创建 Ceph OSD* 部分。

6.6. 使用 创建 CEPH OSD CEPH-VOLUME

create 子命令调用 **prepare** 子命令，然后调用 **activate** 子命令。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 对 Ceph OSD 节点的根级别访问权限.

注意

如果您更希望控制创建过程，可以单独使用 **prepare** 和 **activate** 子命令来创建 OSD，而不使用 **create**。您可以使用两个子命令将新 OSD 逐渐引入存储集群，同时避免重新平衡大量数据。这两种方法方法都相同，但使用 **create** 子命令会使 OSD 在完成 *之后立即* 启动和 *启动*。

流程

1. 创建新 OSD：

语法

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

示例

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

其它资源

- 如需了解更多详细信息，请参阅《*红帽 Ceph 存储管理指南》中的使用"ceph-volume"准备 CephOSD 部分*。
- 如需了解更多详细信息，请参阅《*红帽 Ceph 存储管理指南》中的使用"ceph-volume"激活 CephOSD 部分*。

6.7. 将批处理模式用于 CEPH-VOLUME

在提供单一设备时，**batch** 子命令可自动创建多个 OSD。

ceph-volume 命令根据驱动器类型决定创建 OSD 的最佳方法。Ceph OSD 优化取决于可用的设备：

- 如果所有设备都是传统的硬盘驱动器，**batch** 为每个设备创建一个 OSD。
- 如果所有设备都是固态状态驱动器，**batch** 会在每个设备中创建两个 OSD。
- 如果结合了传统的硬盘驱动器和固态驱动器，**batch** 将使用传统的硬盘驱动器进行数据，并在固态驱动器上创建最大可能的日志(**block.db**)。

注意

batch 子命令不支持为 write-ahead-log(**block.wal**)设备创建单独的逻辑卷。

先决条件

- 正在运行的红帽 Ceph 存储群集.

- 对 Ceph OSD 节点的根级别访问权限.

流程

1. 在几个驱动器上创建 OSD：

   语法

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

   示例

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

其它资源

- 如需了解更多详细信息，*请参阅《红帽 Ceph 存储管理指南》中的使用"ceph-volume"创建 Ceph OSD* 部分。

第 7 章 CEPH 性能基准

作为存储管理员，您可以对红帽 Ceph 存储集群的性能进行基准测试。本节的目的是让 Ceph 管理员能够基本了解 Ceph 的原生基准测试工具。通过这些工具，可以部分了解 Ceph 存储群集的运行状况。这不是 Ceph 性能基准测试的最终指南，也是有关如何相应地调优 Ceph 的指南。

7.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

7.2. 性能基准

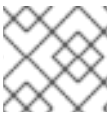
OSD（包括日志、磁盘和网络吞吐量）都应具有可比较的性能基准。您可以通过比较基准性能数据与 Ceph 原生工具中的数据，确定潜在的调优机会。红帽企业 Linux 有许多内置工具，以及大量开源社区工具，可用于帮助完成这些任务。

其它资源

- 有关一些可用工具的详情，请查看知识库 文章。

7.3. CEPH 性能基准测试

Ceph 包含 **rados bench** 命令，用于对 RADOS 存储集群执行性能基准测试。命令将执行写入测试和两种类型的读取测试。在测试读写性能时使用 **--no-cleanup** 选项。默认情况下，**rados bench** 命令将删除它写入存储池中的对象。离开这些对象后，有两个读取测试可以测量顺序和随机读取性能。



注意

在运行这些性能测试前，运行以下命令丢弃所有文件系统缓存：

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

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 创建新存储池：

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

2. 对新创建的存储池执行 10 秒写入测试：

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

   输出示例

   ```
   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. 对存储池执行 10 秒的后续读取测试：

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

输出示例

```
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. 对存储池执行 10 秒随机读取测试：

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

-

输出示例

```
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. 要增加并发读写的数量，使用 **-t** 选项，默认是 16 个线程。另外，**-b** 参数可以调整要写入的对象的大小。默认对象大小为 4 MB。安全的最大对象大小为 16 MB。红帽建议在不同的池中运行这些基准测试的多个副本。这样做可显示多个客户端的性能变化。
添加 **--run-name <label>** 选项来控制在基准测试过程中写入的对象的名称。通过更改每个正在运行的命令实例的 **--run-name** 标签，可以同时运行多个 **rados bench** 命令。这可以防止多个客户端尝试访问同一对象并允许不同的客户端访问不同的对象时可能会出现 I/O 错误。**--run-name** 选项在尝试模拟实际工作负载时也很有用。例如：

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

输出示例

```
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. 删除 **rados bench** 命令创建的数据：

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

7.4. 对 CEPH 块性能进行基准测试

Ceph 包含 **rbd bench-write** 命令，用于测试后续写入到块设备的吞吐量和延迟。默认字节大小为 4096，默认 I/O 线程数为 16，写入的默认字节总数为 1 GB。这些默认值可分别通过 **--io-size**、**--io-threads** 和 **--io-total** 选项进行修改。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 如果还没有载入，载入 **rbd** 内核模块：

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

2. 在 **testbench** 池中创建一个 1 GB **rbd** 镜像文件：

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

3. 将镜像文件映射到设备文件中：

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

4. 在块设备中创建 **ext4** 文件系统：

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

5. 创建新目录：

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

6. 将块设备挂载到 **/mnt/ceph-block-device/** ：

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

7. 针对块设备执行写入性能测试

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

示例

```
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
    ..
```

其它资源

- 有关 **rbd** 命令的详情，请查看 *Red Hat Ceph Storage Block Device Guide* 中的块设备 *命令* 部分。

第 8 章 CEPH 性能计数器

作为存储管理员，您可以收集红帽 Ceph 存储集群的性能指标。Ceph 性能计数器是内部基础架构指标的集合。此指标数据的收集、聚合和图表可通过工具分类来完成，对于性能分析非常有用。

8.1. 先决条件

- 正在运行的红帽 Ceph 存储群集.

8.2. 访问 CEPH 性能计数器

性能计数器通过 Ceph 监控器和 OSD 的套接字接口提供。默认情况下，每个守护进程的套接字文件位于 **/var/run/ceph** 下。性能计数器被分组为集合名称。这些集合名称代表子系统或子系统的实例。

以下是 monitor 和 OSD 集合名称类别的完整列表，其中含有每个 的简短描述：

监控集合名称类别

- Cluster Metrics - 显示存储集群的信息：监控器、OSD、池和 PG
- 级别数据库指标 - 显示后端 **KeyValueStore** 数据库的信息
- monitor Metrics - 显示常规监控器信息
- Paxos Metrics - 显示集群仲裁管理的信息
- throttle Metrics - 显示有关 monitor 如何节流的统计信息

OSD 集合名称类别

- 写回 Throttle Metrics - 显示有关写回节流如何跟踪未清空 IO 的统计信息
- 级别数据库指标 - 显示后端 **KeyValueStore** 数据库的信息
- Objecter Metrics - 显示有关各种基于对象的操作的信息
- 读取和写入操作指标 - 显示有关各种读写操作的信息
- 恢复状态指标 - 显示 - 显示各种恢复状态上的延迟
- OSD Throttle Metrics - 显示有关 OSD 如何节流的统计信息

RADOS 网关集合名称类别

- 对象网关客户端指标 - 显示 GET 和 PUT 请求的统计信息
- Objecter Metrics - 显示有关各种基于对象的操作的信息
- 对象网关 Throttle Metrics - 显示有关 OSD 如何节流的统计信息

8.3. 显示 CEPH 性能计数器

ceph daemon .. perf schema 命令输出可用的指标。每个指标都有关联的位字段值类型。

先决条件

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 查看指标的 schema：

```
ceph daemon DAEMON_NAME perf schema
```

注意

您必须从运行守护进程的节点运行 **ceph daemon** 命令。

2. 从 monitor 节点执行 **ceph daemon .. perf schema** 命令：

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

示例

```
{
    "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. 从 OSD 节点执行 **ceph daemon .. perf schema** 命令：

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

示例

```
...
"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. 位字段值定义

位	含义
1	浮点值
2	未签名 64 位整数值
4	平均（已 + 计数）
8	计数

每个值的位 1 或 2 设置来指示类型，可以是浮动点，也可以是整数值。设置位 4 时，读取有两个值，即 sum 和 count。设定位 8 时，先前间隔的平均时间间隔为总和 delta，因为上一读取后，除以计数 delta 即可。或者，分隔直销的值可以提供生命周期平均值。它们通常用于测量延迟、请求数和请求延迟总和。某些位值被组合使用，如 5、6 和 10。位值 5 是 1 位和位 4 的组合。这意味着平均值为浮动点值。位值 6 是 2 位和位 4 的组合。这意味着平均值是一个整数。位值 10 是 2 位和位 8 的组合。这意味着计数器值是一个整数值。

其它资源

- 如需了解更多 *详细信息，请参阅平均值和总和*。

8.4. 转储 CEPH 性能计数器

ceph daemon .. perf dump 命令输出当前的值，并将指标分组到各个子系统的集合名称下。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

流程

1. 查看当前的指标数据：

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

注意

您必须从运行守护进程的节点运行 **ceph daemon** 命令。

2. 从 monitor 节点执行 **ceph daemon .. perf dump** 命令：

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

示例

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

3. 从 OSD 节点执行 **ceph daemon .. perf dump** 命令：

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

示例

```
...
"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
        },
...
```

其它资源

- 要查看各个可用的 monitor 指标的简短描述，请参阅 *Ceph monitor 指标表*。

8.5. 平均计数和总和

所有延迟数字的位字段值为 5。此字段包含平均计数和总和的浮动点值。**avgcount** 是这个范围内的操作数量，**sum** 是总延迟（以秒为单位）。当使用 **avgcount** 分隔 **sum** 时，这将让您了解每个操作的延迟。

其它资源

- 若要查看可用各个 OSD 指标的简短描述，请参阅 *Ceph OSD 表*。

8.6. CEPH 监控指标

表 8.2. 集群指标表

集合名称	指标名称	位字段值	简短描述
cluster	**num_mon**	2	monitor 数量
	num_mon_quorum	2	仲裁中的监视器数量
	num_osd	2	OSD 总数
	num_osd_up	2	正常运行的 OSD 数量
	num_osd_in	2	集群中的 OSD 数量
	osd_epoch	2	OSD map 的当前 epoch
	osd_bytes	2	以字节为单位的集群总容量
	osd_bytes_used	2	集群中使用的字节数
	osd_bytes_avail	2	集群中可用字节数
	num_pool	2	池数量
	num_pg	2	放置组总数
	num_pg_active_clean	2	处于 active+clean 状态的 PG 数量
	num_pg_active	2	处于活跃状态的 PG 数量
	num_pg_peering	2	处于 peering 状态的放置组数量
	num_object	2	集群中的对象总数
	num_object_degraded	2	降级（缺少副本）对象的数量
	num_object_misplaced	2	错误放置（群集中的 Grong 位置）对象数

集合名称	指标名称	位字段值	简短描述
	num_object_unfound	2	未找到的对象数量
	num_bytes	2	所有对象的字节总数
	num_mds_up	2	在线 MDS 的数量
	num_mds_in	2	集群中 MDS 的数量
	num_mds_failed	2	失败的 MDS 数
	mds_epoch	2	MDS 映射的当前时期

表 8.3. 级别数据库指标表

集合名称	指标名称	位字段值	简短描述
leveldb	**leveldb_get**	10	get
	leveldb_transaction	10	事务
	leveldb_compact	10	紧凑
	leveldb_compact_range	10	按范围划分的紧凑
	leveldb_compact_queue_merge	10	紧凑队列中的范围合并
	leveldb_compact_queue_len	2	压缩队列的长度

表 8.4. 常规监控指标表

集合名称	指标名称	位字段值	简短描述
mon	**num_sessions**	2	当前打开的监控会话数量
	session_add	10	创建的监控会话数量
	session_rm	10	monitor 中的 remove_session 调用数
	session_trim	10	修剪监控会话数量

集合名称	指标名称	位字段值	简短描述
	num_elections	10	参加的选举监控器数量
	election_call	10	由 monitor 启动的选举数
	election_win	10	监控器赢得的选举数
	election_lose	10	监控器丢失的选举数

表 8.5. Paxos 指标表

集合名称	指标名称	位字段值	简短描述
paxos	**start_leader**	10	在领导角色中启动
	start_peon	10	以 peon 角色开始
	restart	10	重启
	refresh	10	refreshes
	refresh_latency	5	刷新延迟
	begin	10	已启动并处理
	begin_keys	6	开始事务中的密钥
	begin_bytes	6	开始事务中的数据
	begin_latency	5	延迟开始操作
	commit	10	commits
	commit_keys	6	提交时事务中的键
	commit_bytes	6	提交时事务中的数据
	commit_latency	5	提交延迟
	collect	10	Ppeon 收集
	collect_keys	6	peon 收集事务中的密钥
	collect_bytes	6	有关 peon 收集的数据

集合名称	指标名称	位字段值	简短描述
	collect_latency	5	peon 收集延迟
	collect_uncommitted	10	已启动和处理中的未提交值
	collect_timeout	10	收集超时
	accept_timeout	10	接受超时
	lease_ack_timeout	10	租期确认超时
	lease_timeout	10	租期超时
	store_state	10	在磁盘上存储共享状态
	store_state_keys	6	处于存储状态的事务键
	store_state_bytes	6	处于存储状态的数据
	store_state_latency	5	存储状态延迟
	share_state	10	共享状态
	share_state_keys	6	处于共享状态的密钥
	share_state_bytes	6	处于共享状态的数据
	new_pn	10	新提议号查询
	new_pn_latency	5	新推荐数量获得延迟

表 8.6. 节流指标表

集合名称	指标名称	位字段值	简短描述
throttle-*	**val**	10	当前可用的节流
	max	10	throttle 的最大值
	get	10	get
	get_sum	10	获取数据
	get_or_fail_fail	10	get_or_fail 期间被阻止

集合名称	指标名称	位字段值	简短描述
	get_or_fail_success	10	get_or_fail 期间成功获得
	take	10	参加
	take_sum	10	捕获的数据
	put	10	PUT
	put_sum	10	放置数据
	wait	5	等待延迟

其它资源

- *集群指标表*
- *级别数据库指标表*
- *常规监控指标表*
- *Paxos 指标表*
- *节流指标表*

8.7. CEPH OSD 指标

表 8.7. 写回 Throttle 指标表

集合名称	指标名称	位字段值	简短描述
WBThrottle	**bytes_dirtied**	2	脏数据
	bytes_wb	2	写入数据
	ios_dirtied	2	脏操作
	ios_wb	2	写入操作
	inodes_dirtied	2	等待写入的条目
	inodes_wb	2	写入条目

表 8.8. 级别数据库指标表

集合名称	指标名称	位字段值	简短描述
leveldb	**leveldb_get**	10	get
	leveldb_transaction	10	事务
	leveldb_compact	10	紧凑
	leveldb_compact_range	10	按范围划分的紧凑
	leveldb_compact_queue_merge	10	紧凑队列中的范围合并
	leveldb_compact_queue_len	2	压缩队列的长度

表 8.9. Objecter 指标表

集合名称	指标名称	位字段值	简短描述
objecter	**op_active**	2	活跃操作
	op_laggy	2	Laggy 操作
	op_send	10	发送的操作
	op_send_bytes	10	发送的数据
	op_resend	10	resent 操作
	op_ack	10	提交回调
	op_commit	10	操作提交
	op	10	操作
	op_r	10	读取操作
	op_w	10	写入操作
	op_rmw	10	read-modify-write 操作
	op_pg	10	PG 操作
	osdop_stat	10	stat 操作

集合名称	指标名称	位字段值	简短描述
	osdop_create	10	创建对象操作
	osdop_read	10	读取操作
	osdop_write	10	写入操作
	osdop_writefull	10	编写完整的对象操作
	osdop_append	10	附加操作
	osdop_zero	10	将对象设置为零操作
	osdop_truncate	10	中继对象操作
	osdop_delete	10	删除对象操作
	osdop_mapext	10	映射扩展操作
	osdop_sparse_read	10	稀疏读操作
	osdop_clonerange	10	克隆范围操作
	osdop_getxattr	10	获取 xattr 操作
	osdop_setxattr	10	设置 xattr 操作
	osdop_cmpxattr	10	xattr 比较操作
	osdop_rmxattr	10	删除 xattr 操作
	osdop_resetxattrs	10	重置 xattr 操作
	osdop_tmap_up	10	TMAP 更新操作
	osdop_tmap_put	10	TMAP 推出操作
	osdop_tmap_get	10	TMAP get 操作
	osdop_call	10	调用（执行）操作
	osdop_watch	10	通过对象操作监控
	osdop_notify	10	通知对象操作
	osdop_src_cmpxattr	10	多操作中的扩展属性比较

集合名称	指标名称	位字段值	简短描述
	osdop_other	10	其他操作
	linger_active	2	活跃的闲置操作
	linger_send	10	已发送的闲置操作
	linger_resend	10	重新闲置操作
	linger_ping	10	发送 ping 以闲置操作
	poolop_active	2	活跃池操作
	poolop_send	10	发送的池操作
	poolop_resend	10	重新池操作
	poolstat_active	2	Active get pool stat 操作
	poolstat_send	10	已发送池统计信息操作
	poolstat_resend	10	Resent 池统计信息
	statfs_active	2	statfs 操作
	statfs_send	10	发送的 FS 统计
	statfs_resend	10	Resent FS stats
	command_active	2	活动命令
	command_send	10	发送的命令
	command_resend	10	resent 命令
	map_epoch	2	OSD map epoch
	map_full	10	接收的完整 OSD map
	map_inc	10	收到增量 OSD map
	osd_sessions	2	开放会话
	osd_session_open	10	已打开会话
	osd_session_close	10	会话已关闭

集合名称	指标名称	位字段值	简短描述
	osd_laggy	2	Laggy OSD 会话

表 8.10. 读取和写入操作指标表

集合名称	指标名称	位字段值	简短描述
osd	**op_wip**	2	当前正在处理的复制操作（主要）
	op_in_bytes	10	客户端操作总写入大小
	op_out_bytes	10	客户端操作总读取大小
	op_latency	5	客户端操作的延迟（包括队列时间）
	op_process_latency	5	客户端操作的延迟（不包括队列时间）
	op_r	10	客户端读取操作
	op_r_out_bytes	10	客户端数据读取
	op_r_latency	5	读取操作的延迟（包括队列时间）
	op_r_process_latency	5	读取操作的延迟（队列时间除外）
	op_w	10	客户端写入操作
	op_w_in_bytes	10	写入的客户端数据
	op_w_rlat	5	客户端写入操作可读/应用延迟
	op_w_latency	5	写入操作的延迟（包括队列时间）
	op_w_process_latency	5	写入操作的延迟（队列时间除外）
	op_rw	10	客户端读-modify-write 操作
	op_rw_in_bytes	10	客户端读写操作写入

集合名称	指标名称	位字段值	简短描述
	op_rw_out_bytes	10	客户端读写操作读取出
	op_rw_rlat	5	客户端 read-modify-write 操作可读/应用延迟
	op_rw_latency	5	读写操作的延迟（包括队列时间）
	op_rw_process_latency	5	读-modify-write 操作的延迟（不包括队列时间）
	subop	10	Suboperations
	subop_in_bytes	10	子操作总大小
	subop_latency	5	Suboperations 延迟
	subop_w	10	复制的写入
	subop_w_in_bytes	10	复制写入数据大小
	subop_w_latency	5	复制的写入延迟
	subop_pull	10	子操作拉取请求
	subop_pull_latency	5	Suboperations pull latency
	subop_push	10	Suboperations push 消息
	subop_push_in_bytes	10	Suboperations push size
	subop_push_latency	5	Suboperations push latency
	pull	10	发送的拉取请求
	push	10	发送的推送消息
	push_out_bytes	10	推送的大小
	push_in	10	入站推送消息
	push_in_bytes	10	入站推送大小

集合名称	指标名称	位字段值	简短描述
	recovery_ops	10	开始恢复操作
	loadavg	2	CPU 负载
	buffer_bytes	2	分配的缓冲区大小总数
	numpg	2	放置组
	numpg_primary	2	此 osd 的主要 PG
	numpg_replica	2	此 osd 是副本的 PG
	numpg_stray	2	准备好从这个 osd 删除的 PG
	heartbeat_to_peers	2	心跳(ping)对等点发送至
	heartbeat_from_peers	2	心跳(ping)对等点，
	map_messages	10	OSD map 消息
	map_message_epochs	10	OSD map epoch
	map_message_epoch_dups	10	OSD map 重复
	stat_bytes	2	OSD 大小
	stat_bytes_used	2	已使用空间
	stat_bytes_avail	2	可用空间
	copyfrom	10	RADOS 'copy-from' 操作
	tier_promote	10	级别提升
	tier_flush	10	等级清除
	tier_flush_fail	10	失败的层次清除
	tier_try_flush	10	分层刷新尝试

集合名称	指标名称	位字段值	简短描述
	tier_try_flush_fail	10	层刷新尝试失败
	tier_evict	10	等级驱除
	tier_whiteout	10	层次白皮书
	tier_dirty	10	脏层标志集
	tier_clean	10	脏层标志已清理
	tier_delay	10	层次延迟（代理等待）
	tier_proxy_read	10	分层代理读取
	agent_wake	10	分层代理唤醒
	agent_skip	10	代理跳过的对象
	agent_flush	10	分层代理清除
	agent_evict	10	分层代理驱除
	object_ctx_cache_hit	10	对象上下文缓存命中
	object_ctx_cache_total	10	对象上下文缓存查找

表 8.11. 恢复状态指标表

集合名称	指标名称	位字段值	简短描述
recoverystate_perf	**initial_latency**	5	初始恢复状态延迟
	started_latency	5	开始恢复状态延迟
	reset_latency	5	重置恢复状态延迟
	start_latency	5	启动恢复状态延迟
	primary_latency	5	主要恢复状态延迟
	peering_latency	5	对等恢复状态延迟
	backfilling_latency	5	回填恢复状态延迟

集合名称	指标名称	位字段值	简短描述
	waitremotebackfillreserved_latency	5	等待远程回填保留恢复状态延迟
	waitlocalbackfillreserved_latency	5	等待本地回填保留恢复状态延迟
	notbackfilling_latency	5	Notbackfilling 恢复状态延迟
	repnotrecovering_latency	5	Repnotrecovering 恢复状态延迟
	repwaitrecoveryreserved_latency	5	REP 等待恢复保留的恢复状态延迟
	repwaitbackfillreserved_latency	5	REP 等待回填保留恢复状态延迟
	RepRecovering_latency	5	RepRecovering 恢复状态延迟
	activating_latency	5	激活恢复状态延迟
	waitlocalrecoveryreserved_latency	5	等待本地恢复保留恢复状态延迟
	waitremoterecoveryreserved_latency	5	等待远程恢复保留恢复状态延迟
	recovering_latency	5	恢复恢复状态延迟
	recovered_latency	5	恢复恢复状态延迟
	clean_latency	5	清理恢复状态延迟
	active_latency	5	活跃恢复状态延迟
	replicaactive_latency	5	Replicaactive 恢复状态延迟
	stray_latency	5	低延迟恢复状态延迟
	getinfo_latency	5	Getinfo 恢复状态延迟
	getlog_latency	5	Getlog 恢复状态延迟

集合名称	指标名称	位字段值	简短描述
	waitactingchange_latency	5	Waitactingchange 恢复状态延迟
	incomplete_latency	5	不完整的恢复状态延迟
	getmissing_latency	5	Getmissing 恢复状态延迟
	waitupthru_latency	5	Waitupthru 恢复状态延迟

表 8.12. OSD Throttle 指标表

集合名称	指标名称	位字段值	简短描述
throttle-*	**val**	10	当前可用的节流
	max	10	throttle 的最大值
	get	10	get
	get_sum	10	获取数据
	get_or_fail_fail	10	get_or_fail 期间被阻止
	get_or_fail_success	10	get_or_fail 期间成功获得
	take	10	参加
	take_sum	10	捕获的数据
	put	10	PUT
	put_sum	10	放置数据
	wait	5	等待延迟

其它资源

- *写回 Throttle 指标表*
- *级别数据库指标表*
- *Objecter 指标表*
- *读取和写入操作指标表*

- *恢复状态指标表*
- *OSD Throttle 指标表*

8.8. CEPH 对象网关指标

表 8.13. RADOS 客户端指标表

集合名称	指标名称	位字段值	简短描述
client.rgw.<rgw_node_name>	**req**	10	requests
	failed_req	10	中止的请求
	get	10	get
	get_b	10	得到的大小
	get_initial_lat	5	获取延迟
	put	10	PUT
	put_b	10	放置大小
	put_initial_lat	5	将延迟
	qlen	2	队列长度
	qactive	2	活跃请求队列
	cache_hit	10	缓存点击
	cache_miss	10	缓存未命中
	keystone_token_cache_hit	10	Keystone 令牌缓存命中
	keystone_token_cache_miss	10	Keystone 令牌缓存未命中

表 8.14. Objecter 指标表

集合名称	指标名称	位字段值	简短描述
objecter	**op_active**	2	活跃操作
	op_laggy	2	Laggy 操作

集合名称	指标名称	位字段值	简短描述
	op_send	10	发送的操作
	op_send_bytes	10	发送的数据
	op_resend	10	resent 操作
	op_ack	10	提交回调
	op_commit	10	操作提交
	op	10	操作
	op_r	10	读取操作
	op_w	10	写入操作
	op_rmw	10	read-modify-write 操作
	op_pg	10	PG 操作
	osdop_stat	10	stat 操作
	osdop_create	10	创建对象操作
	osdop_read	10	读取操作
	osdop_write	10	写入操作
	osdop_writefull	10	编写完整的对象操作
	osdop_append	10	附加操作
	osdop_zero	10	将对象设置为零操作
	osdop_truncate	10	中继对象操作
	osdop_delete	10	删除对象操作
	osdop_mapext	10	映射扩展操作
	osdop_sparse_read	10	稀疏读操作
	osdop_clonerange	10	克隆范围操作
	osdop_getxattr	10	获取 xattr 操作

集合名称	指标名称	位字段值	简短描述
	osdop_setxattr	10	设置 xattr 操作
	osdop_cmpxattr	10	xattr 比较操作
	osdop_rmxattr	10	删除 xattr 操作
	osdop_resetxattrs	10	重置 xattr 操作
	osdop_tmap_up	10	TMAP 更新操作
	osdop_tmap_put	10	TMAP 推出操作
	osdop_tmap_get	10	TMAP get 操作
	osdop_call	10	调用（执行）操作
	osdop_watch	10	通过对象操作监控
	osdop_notify	10	通知对象操作
	osdop_src_cmpxattr	10	多操作中的扩展属性比较
	osdop_other	10	其他操作
	linger_active	2	活跃的闲置操作
	linger_send	10	已发送的闲置操作
	linger_resend	10	重新闲置操作
	linger_ping	10	发送 ping 以闲置操作
	poolop_active	2	活跃池操作
	poolop_send	10	发送的池操作
	poolop_resend	10	重新池操作
	poolstat_active	2	Active get pool stat 操作
	poolstat_send	10	已发送池统计信息操作

集合名称	指标名称	位字段值	简短描述
	poolstat_resend	10	Resent 池统计信息
	statfs_active	2	statfs 操作
	statfs_send	10	发送的 FS 统计
	statfs_resend	10	Resent FS stats
	command_active	2	活动命令
	command_send	10	发送的命令
	command_resend	10	resent 命令
	map_epoch	2	OSD map epoch
	map_full	10	接收的完整 OSD map
	map_inc	10	收到增量 OSD map
	osd_sessions	2	开放会话
	osd_session_open	10	已打开会话
	osd_session_close	10	会话已关闭
	osd_laggy	2	Laggy OSD 会话

表 8.15. RADOS 网关 Throttle Metrics Table

集合名称	指标名称	位字段值	简短描述
throttle-*	**val**	10	当前可用的节流
	max	10	throttle 的最大值
	get	10	get
	get_sum	10	获取数据
	get_or_fail_fail	10	get_or_fail 期间被阻止
	get_or_fail_success	10	get_or_fail 期间成功获得

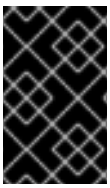
集合名称	指标名称	位字段值	简短描述
	take	10	参加
	take_sum	10	捕获的数据
	put	10	PUT
	put_sum	10	放置数据
	wait	5	等待延迟

其它资源

- RADOS 网关客户端表
- Objecter 指标表
- RADOS 网关 Throttle Metrics Table

第 9 章 BLUESTORE

自红帽 Ceph 存储 4 开始，BlueStore 是 OSD 守护进程的默认对象存储。较早的对象存储 FileStore 需要在原始块设备之上使用文件系统。然后，对象写入文件系统。BlueStore 不需要初始文件系统，因为 BlueStore 将对象直接放置在块设备中。



重要

BlueStore 为生产环境中的 OSD 守护进程提供高性能后端。默认情况下，BlueStore 配置为自我调节。如果您确定手动调优的蓝卡环境性能更好，请联系 红帽支持 并共享您的配置详情，以帮助我们提高自动调节功能。红帽期待您的反馈并感谢您的建议。

9.1. CEPH BLUESTORE

以下是使用 BlueStore 的一些主要功能：

直接管理存储设备

BlueStore 使用原始块设备或分区。这可避免任何可能会限制性能或增加复杂性的本地文件系统（如 XFS）的抽象层。

使用 RocksDB 进行元数据管理

BlueStore 使用 RocksDB 的键值数据库来管理内部元数据，如从对象名称到磁盘上块位置的映射。

完整数据和元数据校验和

默认情况下，写入 BlueStore 的所有数据和元数据都受到一个或多个校验和的保护。不进行验证，不会从磁盘读取数据或元数据，也不会返回到用户。

写时高效复制

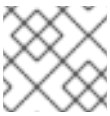
Ceph 块设备和 Ceph 文件系统快照依赖于一种写时复制克隆机制，该克隆机制在 BlueStore 中有效地实施。这会为常规快照和纠删代码池带来高效的 I/O，依靠克隆来实现高效的双阶段提交。

没有大型重复写入

BlueStore 首先将任何新数据写入块设备上的未分配空间，然后提交 RocksDB 事务，该事务更新对象元数据以引用磁盘的新区域。只有写入操作低于可配置的大小阈值时，它才会返回到写入日志方案，类似于 FileStore 的运行方式。

多设备支持

BlueStore 可以使用多个块设备来存储不同的数据。例如：数据的硬盘驱动器(HDD)、用于元数据的 Solid-state Drive(SSD)、非易失性内存(NVM)或非易失性随机访问内存(NVRAM)或 RocksDB 写入日志(WAL)的持久内存。详情请查看 *Ceph BlueStore 设备*。



注意

ceph-disk 工具还没有置备多个设备。若要使用多个设备，必须手动设置 OSD。

有效的块设备使用

由于 BlueStore 不使用任何文件系统，因此可以最大程度减少清除存储设备缓存的需要。

9.2. CEPH BLUESTORE 设备

本节解释了 BlueStore 后端使用的块设备。

BlueStore 管理一个、两个或三个存储设备。

- 主要
- WAL
- DB

在最简单的情形中，BlueStore 会消耗一个（主要）存储设备。存储设备被分区为包含以下内容的两个部分：

- **OSD 元数据** ：使用 XFS 格式化的小型分区，其中包含 OSD 的基本元数据。此数据目录包含有关 OSD 的信息，如其标识符、它所属的集群及其私钥环。
- **数据** ：一个大型分区，用于巩固直接由 BlueStore 管理且包含所有 OSD 数据的设备的其余部分。此主设备通过数据目录中的块符号链接来标识。

您还可以使用两个额外的设备：

- **WAL(write-ahead-log)设备** ：存储 BlueStore 内部日志或写入日志的设备。它由数据目录中的 **block.wal** 符号链接标识。只有在设备比主设备快时才考虑使用 WAL 设备。例如，WAL 设备使用 SSD 磁盘，而主设备使用 HDD 磁盘。
- **DB 设备** ：存储 BlueStore 内部元数据的设备。嵌入的 RocksDB 数据库将元数据放置在 DB 设备中，而不是放置在主设备上，以提高性能。如果 DB 设备已满，它开始向主设备添加元数据。只有在设备比主设备快时才考虑使用 DB 设备。

警告

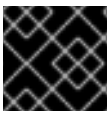
如果您在快速设备中仅提供千兆字节的存储。红帽建议将它用作 WAL 设备。如果您有更快的设备可用，请考虑将其用作 DB 设备。BlueStore 日志始终放在最快的设备上，因此使用 DB 设备可以获得与 WAL 设备相同的益处，同时也允许存储额外的元数据。

9.3. CEPH BLUESTORE 缓存

BlueStore 缓存是缓冲区的集合，可以根据配置填充，如 OSD 守护进程从磁盘读取或写入到磁盘时一样。默认情况下，在 Red Hat Ceph Storage 中，BlueStore 会在读取时缓存，但不进行写入。这是因为 **bluestore_default_buffered_write** 选项被设置为 **false**，以避免与缓存驱除相关的潜在开销。

如果 **bluestore_default_buffered_write** 选项被设置为 **true**，数据会首先写入缓冲区，然后提交到磁盘。之后，会向客户端发送写入确认，以便后续读取缓存中已有数据的速度，直到该数据被驱除。

读取密集型工作负载不会立即从 BlueStore 缓存中受益。随着更多的读取完成，缓存将随着时间增加，后续读取的性能也会得到提升。缓存填充的速度取决于 BlueStore 块和数据库磁盘类型，以及客户端的工作负载要求。



重要

在启用 **bluestore_default_buffered_write** 选项前，请联系 红帽支持。

9.4. CEPH BLUESTORE 的大小注意事项

使用 BlueStore OSD 混合传统和固态硬盘时，适当调整 RocksDB 逻辑卷(**block.db**)大小非常重要。红帽建议 RocksDB 逻辑卷不少于块大小的 4% 和对象、文件和混合工作负载。红帽支持 1% 的 BlueStore 块大小及 RocksDB 和 OpenStack 块工作负载。例如，如果对象工作负载的块大小为 1 TB，则至少创建一个 40 GB RocksDB 逻辑卷。

如果没有混合驱动器类型，则无需使用单独的 RocksDB 逻辑卷。BlueStore 将自动管理 RocksDB 的大小。

BlueStore 的缓存内存用于 RocksDB、BlueStore 元数据和对象数据的键值对元数据。

注意

BlueStore 缓存内存值是 OSD 已消耗的内存足迹之外的。

9.5. 添加 CEPH BLUESTORE OSD

本节介绍如何使用 BlueStore 后端对象存储安装新的 Ceph OSD 节点。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 节点的根级别访问权限。

流程

1. 将新的 OSD 节点添加到 Ansible 清单文件中的 **[osds]** 部分中，默认位于 **/etc/ansible/hosts**。

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

替换：

- ***HOST_NAME*** OSD 节点的名称

示例

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

2. 进入 **/usr/share/ceph-ansible/** 目录。

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

3. 创建 **host_vars** 目录。

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

4. 在 **host_vars** 中为新添加的 OSD 创建 配置文件。

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

替换：

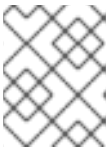
- ***HOST_NAME*** 新添加的 OSD 的主机名

示例

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

5. 在新创建的文件中添加以下设置：

```
osd_objectstore: bluestore
```



注意

要将 BlueStore 用于所有 OSD，请将 **osd_objectstore:bluestore** 添加到 **group_vars/all.yml** 文件中。

6. 在 **host_vars/*HOST_NAME*.yml** 中配置 BlueStore OSD：

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

替换：

- ***DATALV*** 使用数据逻辑卷名称
- ***DATAVG*** 使用数据逻辑卷组群名称

示例

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

7. 可选。如果要将 **block.wal** 和 **block.db** 存储在专用逻辑卷中，请按如下方式编辑 **host_vars/*HOST_NAME*.yml** 文件：

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

替换：

- 带有应包含数据的逻辑卷的 *DATALV*

- 包含 write-ahead-log 的逻辑卷的 *WALLV*
- 带有卷组 WAL 和/或 DB 设备 LV 的 *VG* 位于
- 带有逻辑卷的 *DBLV* 应包含 BlueStore 内部元数据

示例

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

注意

当将 **lvm_volumes:** 与 **osd_objectstore: bluestore** 搭配使用时，**lvm_volumes** YAML 字典必须至少包含 **data**。定义 **wal** 或 **db** 时，它必须同时具有 LV 名称和 VG 名称（**db** 和 **wal** 不需要）。这允许四个组合：仅数据、数据和 wal、数据和 wal 和 wal 和 db，或者数据和 db。数据可以是裸设备、lv 或 分区。**wal** 和 **db** 可以是 lv 或 分区。当指定原始设备或者分区 **ceph-volume** 时，会在其上面放置逻辑卷。

注意

目前，**ceph-ansible** 不会创建卷组或者逻辑卷。这必须在运行 Anisble playbook 之前完成。

8. 打开并编辑 **group_vars/all.yml** 文件，并取消注释 **osd_memory_target** 选项。调整您希望 OSD 使用的内存量的值。

注意

osd_memory_target 选项的默认值是 **4000000000**，即 4 GB。这个选项将 BlueStore 缓存固定在内存中。

重要

osd_memory_target 选项只适用于由 BlueStore 支持的 OSD。

9. 运行以下 Ansible playbook:

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

10. 从 Ceph 监控节点，验证新 OSD 是否已成功添加：

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

9.6. 为小型写入调优 CEPH BLUESTORE

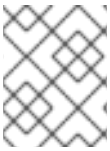
在 BlueStore 中，原始分区以 **bluestore_min_alloc_size** 的块的形式进行配置和管理。默认情况

下，**bluestore_min_alloc_size** 用于 HDD，16 KB 代表 SSD。当每个块中的未写入区域写入原始分区时，它会被填充为零。如果工作负载没有正确定义未使用空间（例如编写小对象时），这会导致浪费空间。

最佳实践是将 **bluestore_min_alloc_size** 设置为与最小写入操作相匹配，从而避免产生放大的罚款。

例如，如果您的客户端频繁写入 4 KB 对象，请使用 **ceph-ansible** 在 OSD 节点上配置以下设置：

bluestore_min_alloc_size = 4096



注意

bluestore_min_alloc_size_ssd 和 **bluestore_min_alloc_size_hdd** 的设置分别特定于 SSD 和 HDD，但不需要设置它们，因为设置 **bluestore_min_alloc_size** 会覆盖它们。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- 新服务器可以重新调配为 OSD 节点，或者：
- 可以重新部署的 OSD 节点。
- 如果要重新部署现有的 Ceph OSD 节点，Ceph 监控节点的管理密钥环。

流程

1. 可选：如果重新部署现有的 OSD 节点，请使用 **shrink-osd.yml** Ansible playbook 从集群中删除该 OSD。

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

示例

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

2. 如果重新部署现有的 OSD 节点，请擦除 OSD 驱动器并重新安装操作系统。
3. 利用 Ansible，为节点做好 OSD 调配准备。准备任务示例包括启用红帽 Ceph 存储存储库、添加 Ansible 用户和启用免密码 SSH 登录。
4. 将 **bluestore_min_alloc_size** 添加到 **group_vars/all.yml** Ansible playbook 的 **ceph_conf_overrides** 部分：

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

5. 如果部署新节点，将其添加到 Ansible 清单文件中，通常： **/etc/ansible/hosts**

```
[osds]
OSD_NODE_NAME
```

示例

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

6. 如果重新部署现有的 OSD，请将 Ceph 监控节点上的 admin keyring 文件复制到您要部署 OSD 的节点。

7. 使用 Ansible 置备 OSD 节点：

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

示例

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

8. playbook 完成后，使用 **ceph daemon** 命令验证设置：

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

示例

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

您可以看到 **bluestore_min_alloc_size** 设置为 4096 字节，相当于 4 KiB。

其它资源

- 如需更多信息，请参阅《*红帽 Ceph 存储安装指南*》。

9.7. BLUESTORE 分段工具

作为存储管理员，您需要定期检查 BlueStore OSD 的碎片级别。您可以通过一个简单的命令检查碎片级别，以查看离线或在线 OSD。

9.7.1. 先决条件

- 正在运行的红帽 Ceph 存储 3.3 或更高版本的存储集群。
- BlueStore OSDs.

9.7.2. 什么是 BlueStore 碎片工具？

对于 BlueStore OSD，可用空间随时间分散到底层存储设备上。有些碎片通常是正常的，但是当过度碎片化时，这会导致性能下降。

BlueStore 碎片工具会在 BlueStore OSD 的碎片级别生成分数。此分段分数以范围 0 到 1 的形式指定。0 分表示没有碎片，1 分数表示严重碎片。

表 9.1. 分段分数的含义

分数	碎片量
0.0 - 0.4	无可减少碎片.
0.4 - 0.7	小且可接受的碎片.
0.7 - 0.9	相当大，但安全碎片.
0.9 - 1.0	严重碎片，导致性能问题.

重要

如果您有严重的碎片，并且需要某些帮助来解决此问题，请 联系红帽支持团队。

9.7.3. 检查碎片

可以在线或脱机检查 BlueStore OSD 的碎片级别。

先决条件

- 正在运行的红帽 Ceph 存储 3.3 或更高版本的存储集群。
- BlueStore OSDs.

在线 BlueStore 分段分数

1. 检查正在运行的 BlueStore OSD 进程：
   a. 简单报告：

      语法

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

      示例

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

   b. 更详细的报告：

      语法

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

      示例

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

offline BlueStore 分段分数

1. 检查非运行的 BlueStore OSD 进程：

   a. 简单报告：

   语法

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

   示例

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

   b. 更详细的报告：

   语法

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

   示例

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

其它资源

- 如需了解碎片分数的详细信息，请参阅 Red Hat Ceph Storage 4.1 *BlueStore 片段工具*。

9.8. 如何将对象存储从 FILESTORE 迁移到 BLUESTORE

作为存储管理员，您可以从传统的对象存储 FileStore 迁移到新对象存储 BlueStore。

9.8.1. 先决条件

- 个健康且运行中的红帽 Ceph 存储集群。

9.8.2. 从 FileStore 迁移到 BlueStore

与传统的 FileStore 相比，BlueStore 提高了性能和稳健性.单个红帽 Ceph 存储集群可以同时包含 FileStore 和 BlueStore 设备。

转换单个 OSD 不能就地或隔离进行。转换过程将依赖于存储集群的正常复制和修复过程，或者将 OSD 内容从旧(FileStore)设备复制到新(BlueStore)设备的工具和策略。可以通过两种方法从 FileStore 迁移到 BlueStore。

第一种方法

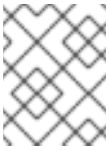
第一种方法是依次标记各个设备，等待数据在存储集群之间复制，重新调配 OSD，再将它重新标记为"in"。以下是这种方法的优势和缺点：

优点

- 简单.
- 可以逐个设备完成。
- 不需要备用设备或节点。

缺点

- 通过网络复制数据会发生两次。



注意

个副本到存储集群中的某些其他 OSD，允许您维护所需的副本数，然后另一副本返回到重新调配的 BlueStore OSD。

第二种方法

第二种方法是进行整个节点替换。您需要有一个没有数据的空节点。

有两种方法可以做到这一点：* 从不属于存储集群的新空节点开始。* 通过从存储集群中的现有节点卸载数据。

优点

- 数据仅通过网络复制一次。
- 一次性转换整个节点的 OSD。
- 可以并行化 以一次转换多个节点。
- 每个节点不需要备用设备。

缺点

- 需要备用节点。
- 整个节点的 OSD 的价值一次迁移数据。这可能会影响集群的整体性能。
- 所有迁移的数据仍然在网络上形成一个完整的跃点。

9.8.3. 使用 Ansible 从 FileStore 迁移到 BlueStore

使用 Ansible 从 FileStore 迁移到 BlueStore 将缩小并重新部署节点上的所有 OSD。在开始迁移前，Ansible playbook 先进行容量检查。然后 **ceph-volume** 实用程序重新部署 OSD。

先决条件

- 正常运行的红帽 Ceph 存储 4 集群。
- 用于 Ansible 应用程序的 **ansible** 用户帐户。

流程

1. 以 **ansible** 用户身份登录 Ansible 管理节点。
2. 编辑 **group_vars/osd.yml** 文件，添加并设置以下选项：

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

3. 运行以下 Ansible playbook:

语法

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

示例

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

4. 等待迁移完成，然后从存储集群中的下一个 OSD 节点开始。

9.8.4. 使用标记从 FileStore 迁移到 BlueStore，并替换方法

从 FileStore 迁移到 BlueStore 的最简单方法是依次标记各个设备，等待数据在存储集群之间复制，重新调配 OSD，再将它重新标记为"in"。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- **root** 访问该节点。

流程

将以下变量 ***OSD_ID*** 替换为强制标识号。

1. 查找要替换的 FileStore OSD。
   a. 获取 OSD 标识号：

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

   b. 确定 OSD 使用的是 FileStore 还是 BlueStore：

语法

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

示例

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

c. 查看 FileStore 设备与 BlueStore 设备之间的当前计数：

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

2. 将 FileStore OSD 标记为 out：

```
ceph osd out OSD_ID
```

3. 等待数据迁移出 OSD：

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

4. 停止 OSD：

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

5. 捕获此 OSD 使用的设备：

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

6. 卸载 OSD：

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

7. 使用第 5 步中的值作为 ***DEVICE*** 销毁 OSD 数据：

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

重要

Be **EXTREMELY CAREFUL**，因为这会破坏设备的内容。在继续操作前，请确定设备中的数据（即存储集群处于健康状态）。

注意

如果 OSD 已加密，则卸载 **osd-lockbox** 并移除加密，然后再使用 **dmsetup remove** 来部署应用 OSD。

注意

如果 OSD 包含逻辑卷，则使用 **ceph-volume lvm zap** 命令中的 **--destroy** 选项。

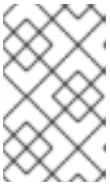
8. 使存储集群意识到 OSD 已销毁：

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

9. 使用第 5 步中的 ***DEVICE*** 重新调配 OSD 作为 BlueStore OSD，相同 ***OSD_ID*** ：

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

10. 重复此过程。



注意

重新填充新的 BlueStore OSD 可能会同时排空下一个 FileStore OSD，只要您在销毁任何 OSD 之前确保存储集群为 **HEALTH_OK**。如果不这样做，将减少数据的冗余性，并增加数据丢失的风险或潜在的数据丢失风险。

9.8.5. 使用整个节点替换方法从 FileStore 迁移到 BlueStore

从 FileStore 迁移到 BlueStore 可以针对节点进行迁移，方法是将每个存储的数据副本只传输一次。此迁移可通过存储集群中的备用节点完成，或者有足够的可用空间从存储群集中撤离整个节点，以便将其用作备用。理想情况下，节点必须具有与您要迁移的其他节点大致相同的容量。

先决条件

- 正在运行的红帽 Ceph 存储群集.
- **root** 访问该节点。
- 个空节点，没有数据。

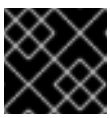
流程

- 将以下变量 ***NEWNODE*** 替换为新节点名称。
- 将以下变量 ***EXISTING_NODE_TO_CONVERT*** 替换为存储集群中已存在的节点名称。
- 将以下变量 ***OSD_ID*** 替换为 OSD 标识号。

1. 使用不在存储集群中的新节点。对于在存储集群中使用已存在的节点，请跳至第 3 步。

   a. 将节点添加到 CRUSH 层次结构中：

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

   重要

   不要将它附加到 root。

   b. 安装 Ceph 软件包：

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

   注意

   默认情况下，复制 Ceph 配置文件 **/etc/ceph/ceph.conf**，并将密钥环复制到新节点。

2. 跳到第 5 步。

3. 如果您在存储集群中使用已有节点，请使用以下命令：

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

注意

其中 **default** 是 CRUSH map 中的即时祖先。

4. 跳到第 8 步。

5. 为所有设备置备新的 BlueStore OSD:

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

6. 验证 OSD 是否加入集群：

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

您应该会在节点名称下看到新节点名称下的所有 OSD，但该节点 **不能** 嵌套到层次结构中任何其他节点的下方。

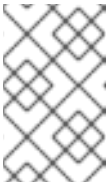
示例

```
[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. 将新节点切换到集群中的旧节点位置：

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

此时，***EXISTING_NODE_TO_CONVERT*** 中的所有数据都将开始迁移到 ***NEWNODE*** 上的 OSD。



注意

如果旧节点和新节点的总容量有差异，您可能也看到一些数据迁移到存储集群中或从存储集群中的其他节点迁移，但只要节点的大小类似，则该数量相对较少。

8. 等待数据迁移完成：

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

9. 登录到 ***EXISTING_NODE_TO_CONVERT***，停止并卸载现在空 ***EXISTING_NODE_TO_CONVERT*** 上的所有旧 OSD：

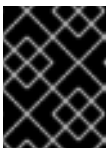
```
[root@mon ~]# systemctl stop ceph-osd@OSD_ID
[root@mon ~]# umount /var/lib/ceph/osd/ceph-OSD_ID
```

10. 销毁并清除旧的 OSD：

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

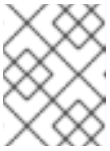
11. 擦除旧 OSD 设备.这要求您识别要手动擦除的设备。为每个设备运行以下命令：

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



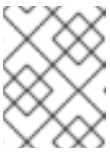
重要

Be **EXTREMELY CAREFUL**，因为这会破坏设备的内容。在继续操作前，请确定设备中的数据（即存储集群处于健康状态）。



注意

如果 OSD 已加密，则卸载 **osd-lockbox** 并移除加密，然后再使用 **dmsetup remove** 来部署应用 OSD。



注意

如果 OSD 包含逻辑卷，则使用 **ceph-volume lvm zap** 命令中的 **--destroy** 选项。

12. 将现在空的旧节点用作新节点，再重复该过程。