Red Hat Ceph Storage 7

架构指南

Red Hat Ceph Storage 架构指南

Last Updated: 2024-06-14

Red Hat Ceph Storage 7 架构指南

Red Hat Ceph Storage 架构指南

法律通告

摘要

本文档为 Ceph Storage 集群及其客户端提供架构信息。 红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 信息。

目录

第 1 章 CEPH 架构

Red Hat Ceph Storage 集群是一个分布式数据对象存储，旨在提供卓越的性能、可靠性和可扩展性。分布式对象存储是未来的存储，因为它们适用于非结构化数据，并且因为客户端可以同时使用现代对象接口和旧接口。

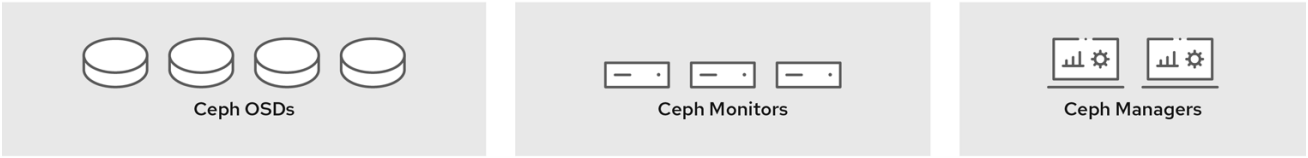
例如：

- 使用多种语言（C/C++、Java、Python）的 API
- RESTful 接口(S3/Swift)
- 块设备接口
- 文件系统接口

红帽 Ceph 存储集群的强大功能可以改变您的组织的 IT 基础架构，以及管理大量数据的能力，特别是适用于 Red Hat Enterprise Linux OSP 等云计算平台。Red Hat Ceph Storage 集群提供了**强大的**可扩展性，可以满足上千的客户访问 petabytes 到 exabytes 级别的数据。

各个 Ceph 部署的核心是 Red Hat Ceph Storage 集群。它由三种类型的守护进程组成：

- **Ceph OSD 守护进程：** Ceph OSD 代表 Ceph 客户端存储数据。此外，Ceph OSD 利用 Ceph 节点的 CPU、内存和网络来执行数据复制、纠删代码、重新平衡、恢复、监控和报告功能。
- **Ceph Monitor：** Ceph Monitor 为 Red Hat Ceph Storage 集群维护一个主副本，它包括 Red Hat Ceph Storage 集群的当前状态。monitor 需要高度一致性，并使用 Paxos 来确保与红帽 Ceph 存储集群的状态达成一致。
- **Ceph Manager：** Ceph Manager 维护关于放置组的详细信息，处理元数据和主机元数据，以大规模提高性能。Ceph 管理器处理许多只读 Ceph CLI 查询的执行，如放置组统计信息。Ceph 管理器还提供 RESTful 监控 API。



158_Ceph_0621

Ceph 客户端从 Red Hat Ceph Storage 集群读取和写入数据。客户端需要以下数据与 Red Hat Ceph Storage 集群通信：

- Ceph 配置文件或集群名称（通常是 **ceph**）和监控器地址。
- 池名称。
- 用户名和到 secret 密钥的路径。

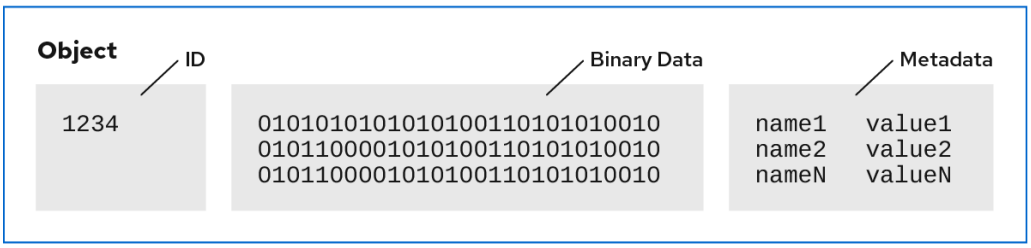
Ceph 客户端维护对象 ID 和用于存储对象的池名称。但是，它们不需要维护对象对 OSD 索引，或与中央对象索引通信来查找对象位置。为存储和检索数据，Ceph 客户端访问 Ceph 监控器并检索 Red Hat Ceph Storage 集群映射的最新副本。然后，Ceph 客户端为 **librados** 提供对象名称和池名称，它计算对象的 PG 和 Primary OSD，以使用 CRUSH（可扩展哈希下的受控复制）算法存储和检索数据。Ceph 客户端连接到执行读取和写入操作的Primary OSD。客户端和 OSD 之间没有中间服务器、代理或总线。

当 OSD 存储数据时，它可以从 Ceph 客户端接收数据 - 无论客户端是 Ceph 块设备、Ceph 对象网关、Ceph Filesystem 或另一个接口 -，它将数据作为对象保存。

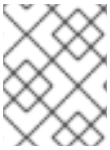
注意

对象 ID 在整个集群中是唯一的，而不仅仅是 OSD 的存储介质。

Ceph OSD 将所有数据作为对象存储在扁平命名空间中。它并不使用具有层次结构的目录。对象具有集群范围的唯一标识符、二进制数据和元数据，由一组名称/值对组成。



Ceph 客户端定义客户端数据格式的语义。例如，Ceph 块设备将块设备镜像映射到集群中存储的一系列对象。



注意

由唯一 ID、数据和名称/值对组成的对象可以表示结构化和非结构化数据，以及旧的和领先的边缘数据存储接口。

第 2 章 CEPH 核心组件

Red Hat Ceph Storage 集群可以拥有大量 Ceph 节点，以实现无限扩展、高可用性和性能。每个节点利用非专有硬件和智能 Ceph 守护进程，它们相互通信：

- 写和读数据
- 压缩数据
- 通过复制或纠删代码数据来确保持久性
- 监控和报告集群运行状况，也称为 'heartbeating（心跳）'
- 动态重新分发数据也称为"backfilling（回填）"
- 确保数据完整性；以及.
- 从故障中恢复。

对于读取和写入数据的 Ceph 客户端接口，Red Hat Ceph Storage 集群类似于存储数据的简单池。但是，**librados** 和存储集群通过对客户端接口完全透明的方式执行许多复杂的操作。Ceph 客户端和 Ceph OSD 都使用 CRUSH（可扩展哈希下的受控复制）算法。以下小节详细介绍了 CRUSH 如何使 Ceph 无缝执行这些操作。

先决条件

- 对分布式存储系统有基本了解。

2.1. CEPH 池

Ceph 存储集群将数据对象存储在名为"Pools"的逻辑分区中。Ceph 管理员可以创建用于特定类型的数据的池，例如用于块设备、对象网关或只是将一组用户与另一组隔离开来。

从 Ceph 客户端的角度来看，存储群集非常简单。当 Ceph 客户端使用 I/O 上下文读取或写入数据时，它**始**终连接到 Ceph 存储集群中的存储池。客户端指定池名称、用户和机密密钥，因此池似乎要充当含有对数据对象的访问控制的逻辑分区。

实际上，Ceph 池不只是用于存储对象数据的逻辑分区。池在 Ceph 存储集群分发和存储数据的方式中发挥着重要作用。但是，这些复杂的操作对 Ceph 客户端是完全透明的。

Ceph 池定义：

- **池类型：** 在早期版本的 Ceph 中，池仅维护对象的多个深度副本。现在，Ceph 可以维护一个对象的多个副本，或者可以使用纠删代码来确保持久性。数据持久性方法在池范围内，在创建池后不会更改。池类型定义了在创建池时的数据持久性方法。池类型对客户端完全透明。
- **放置组：** 在exabyte 级别的扩展存储集群中，Ceph 池可能会存储数百万的数据对象或更多。Ceph 必须处理许多类型的操作，包括通过副本或纠删代码区块数据持久性，通过清理或 CRC 检查、复制、重新平衡和恢复来数据完整性。因此，以每个对象为基础管理数据会带来可扩展性和性能瓶颈。Ceph 通过将池分片划分为放置组来解决这一瓶颈。CRUSH 算法计算用于存储对象并计算 PG 的 OSD 法集的 PG。CRUSH 将每个对象放入 PG 中。然后，CRUSH 将每个 PG 存储在一组 OSD 中。系统管理员在创建或修改池时设置放置组计数。
- **CRUSH Ruleset：** CRUSH 承担另一个重要角色：CRUSH 可以检测故障域和性能域。CRUSH 可以通过存储介质类型来识别 OSD，并将 OSD 分层整理到节点、机架和行中。CRUSH 使 Ceph OSD 能够跨故障域存储对象副本。例如，对象的副本可以存储在不同的服务器机房、aisles、机

架和节点中。如果集群的一个较大部分失败（如机架），集群仍然可以处于降级状态，直到集群恢复为止。

此外，CRUSH 支持客户端将数据写入特定类型的硬件，如 SSD、使用 SSD 日志的硬盘驱动器或硬盘驱动器，其与数据位于与数据相同的驱动器上。CRUSH 规则集确定池的故障域和性能域。管理员在创建池时设置 CRUSH 规则集。

注意

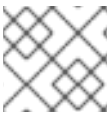
在创建池后，管理员将**无法**更改池的规则集。

- **持久化** ：在 exabyte 级别的扩展存储集群中，硬件故障是预料中的事件，而不是例外。当使用数据对象代表大型粒度存储接口（如块设备）时，为更大粒度接口丢失一个或多个数据对象可能会破坏更大粒度存储实体 - 不可取的渲染。因此，不能容忍数据丢失。Ceph 以两种方式提供高数据持久性：
  - 副本池将使用 CRUSH 故障域存储对象的多个深度副本，从而物理地分隔一个数据对象副本。也就是说，副本被分发到不同的物理硬件中。这会在硬件出现故障时提供了数据的持久性。
  - 纠删代码池将每个对象存储为 **K+M** 块，其中 **K** 代表数据区块，**M** 代表编码区块。sum 代表了用于存储对象的 OSD 的总数量，**M** 值代表可以失败的 OSD 数量，在有 **M** 的 OSD失败时仍可以恢复数据。

从客户端的角度来看，Ceph 是小巧的。客户端只是读取和写入池。但是，池在数据持久性、性能和高可用性方面扮演着重要角色。

2.2. CEPH 身份验证

为了识别用户和防止中间人攻击，Ceph 提供其 **cephx** 身份验证系统，用于验证用户和守护进程。



注意

cephx 协议不会处理通过存储在 OSD 中的网络或数据传输的数据加密。

cephx 使用共享密钥来进行身份验证，这意味着客户端和服务器均有客户端的机密密钥的副本。身份验证协议使双方能够证明其各自具有密钥副本，而无需实际发现它。这提供了 mutual 身份验证，这意味着集群是确保用户具有 secret 密钥，用户则确保集群具有 secret 密钥的副本。

Cephx

cephx 身份验证协议以类似于 Kerberos 的方式运行。

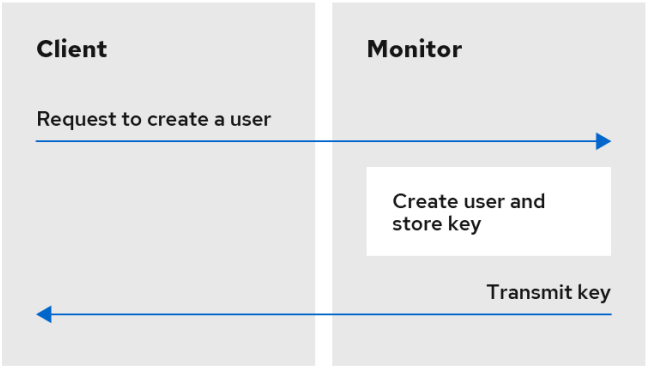
一个用户/actor 可以调用 Ceph 客户端以联系 monitor。与 Kerberos 不同，每个监控器可以验证用户和分发密钥，因此使用 **cephx** 时没有单点故障或瓶颈。monitor 返回与 Kerberos ticket 类似的身份验证数据结构，其中包含用于获取 Ceph 服务的会话密钥。此会话密钥本身通过用户的永久 secret 密钥加密，以便只有用户可以从 Ceph 监视器请求服务。然后，客户端使用 session 键从 monitor 请求其所需的服务，监控器为客户端提供一个 ticket，将客户端验证实际处理数据的 OSD。Ceph 监视器和 OSD 共享机密，因此客户端可以使用监控器提供的票据以及集群中的任何 OSD 或元数据服务器。与 Kerberos 一样，**cephx** 票据到期，因此攻击者无法使用过期的票据或会话密钥被大量获得。这种验证形式可防止攻击者访问通信介质的攻击者，可以在另一个用户的身份下创建虚假消息，或者更改其他用户的合法消息，只要用户的机密密钥在过期之前不会被篡改。

要使用 **cephx**，管理员必须首先设置用户。在以下示意图中，**client.admin** 用户从命令行调用 **ceph auth get-or-create-key** 来生成用户名和密钥。Ceph 的 **auth** 子系统生成用户名和密钥，使用 monitor(s)存储

副本，并将用户的机密信息传输回 **client.admin** 用户。这意味着客户端和 monitor 共享一个 secret 密钥。

注意

client.admin 用户必须以安全的方式向用户提供用户 ID 和密钥。



2.3. CEPH 放置组

在集群中存储数百万对象，单独管理是资源密集型。因此，Ceph 使用放置组(PG)更有效地管理大量对象。

PG 是池的一个子集，用于包含一组对象。Ceph 将池分片到一系列 PG 中。然后，CRUSH 算法将集群映射和集群的状态纳入帐户，并将 PG 平均和伪随机分发到集群中的 OSD。

下面是它的运作方式。

当系统管理员创建池时，CRUSH 会为池创建用户定义的 PG 数量。通常，PG 的数量应当是数据的一个合理的细粒度子集。例如，每个池每个 OSD 有 100 个 PG，这意味着每个 PG 大约包含池数据的 1%。

当 Ceph 从一个 OSD 从一个 OSD 移到另一个 OSD 时，PG 的数量会对性能产生影响。如果池中 PG 数量太少，Ceph 将同时迁移大量数据百分比，网络负载会对集群的性能造成负面影响。如果池太多 PG，Ceph 在移动数据的小百分比时将使用过多的 CPU 和 RAM，从而给集群的性能造成负面影响。有关计算用于实现最佳性能的 PG 数量的详情，请参阅 放置组计数。

Ceph 通过存储对象的副本或存储对象的纠删代码区块来确保数据丢失。由于 Ceph 将对象或纠删代码区块存储在 PG 中，因此 Ceph 将每个 PG 复制到一组 OSD 中，为对象的每个副本或对象的每个纠删代码区块复制了名为"Acting Set"的 OSD 中。系统管理员可以确定池中 PG 数量以及副本或纠删代码区块的数量。但是，CRUSH 算法会计算用于特定 PG 执行的 OSD。

CRUSH 算法和 PG 使 Ceph 动态化。集群映射或集群状态的更改可能会导致 Ceph 将 PG 从一个 OSD 移动到另一个 OSD。

以下是几个示例：

- **扩展集群：** 在向集群添加新主机及其 OSD 时，集群映射会改变。由于 CRUSH 平均而伪随机地将 PG 分发到整个集群中的 OSD，因此添加新的主机及其 OSD 意味着 CRUSH 会将部分池的 PG 重新分配到这些新 OSD。这意味着系统管理员不必手动重新平衡集群。此外，这意味着新 OSD 包含与其他 OSD 大约包含相同的数据量。这也意味着新 OSD 不包含新编写的 OSD，从而防止集群中的"热点"。

- **一个 OSD 失败：** 当 OSD 出现故障时，集群的状态会改变。Ceph 暂时丢失其中一个副本或纠删代码区块，需要制作另一个副本。如果执行集合中的 Primary OSD 失败，则执行集合中的下一个 OSD 将变为 primary，CRUSH 会计算新的 OSD 以存储额外的副本或纠删代码块。

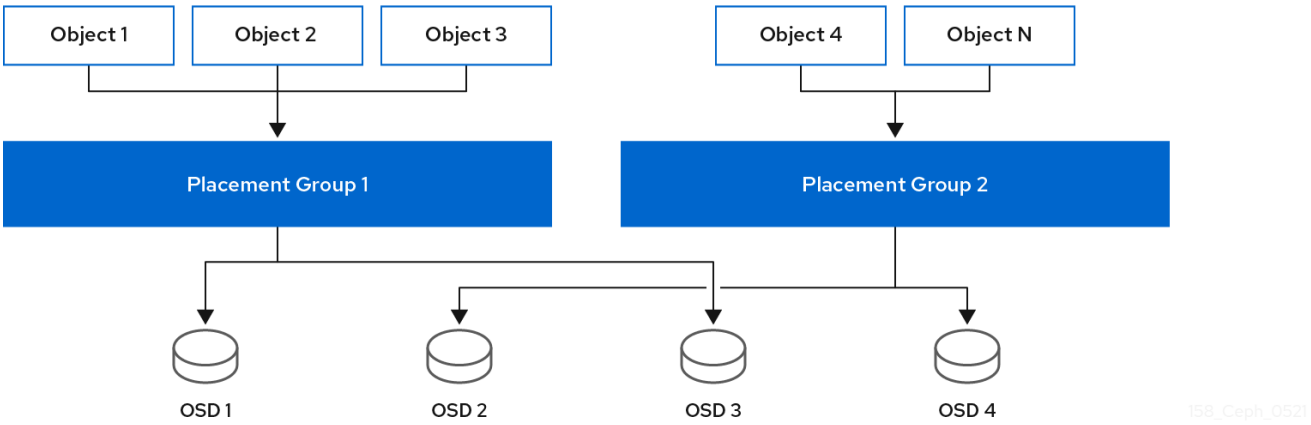
通过在数百个到数千 PG 的环境中管理数百万个对象，Ceph 存储集群可以有效地从故障中扩大、缩小和恢复。

对于 Ceph 客户端，通过 **librados** 的 CRUSH 算法使读和编写对象的过程非常简单。Ceph 客户端只是将对象写入池，或者从池中读取对象。执行集合中的 Primary OSD 可以将对象或纠删代码区块的副本写入代表 Ceph 客户端集合的次要 OSD。

如果 cluster map 或集群状态发生变化，则存储 PG 的 CRUSH 计算也会变化。例如，Ceph 客户端可以将对象 **foo** 写入池 **bar**。CRUSH 将对象分配到 PG **1.a**，并将它存储在 **OSD 5** 上，分别在 **OSD 10** 和 **OSD 15** 上制作副本。如果 **OSD 5** 失败，集群状态会改变。当 Ceph 客户端从池 **bar** 中读取对象 **foo** 时，通过 **librados** 的客户端将自动从 **OSD 10** 检索，因为新的 Primary OSD。

通过 **librados** 的 Ceph 客户端在编写和读取对象时直接连接到该集合内的 Primary OSD。因为 I/O 操作不使用集中式代理，所以网络超额订阅通常不是 Ceph 的问题。

下图显示了 CRUSH 如何分配对象到 PG，并将 PG 分配给 OSD。CRUSH 算法将 PG 分配给 OSD，使得执行集合中的每一 OSD 分配到单独的故障域中，这通常意味着 OSD 始终位于独立的服务器主机上，有时位于单独的机架中。



2.4. CEPH CRUSH 规则集

Ceph 为池分配 CRUSH 规则集。当 Ceph 客户端在池中存储或检索数据时，Ceph 会标识 CRUSH 规则集、规则集内的规则以及用于存储和检索数据的规则中的顶级 bucket。当 Ceph 处理 CRUSH 规则时，它会标识包含对象的 PG 的 Primary OSD。这使得客户端能够直接连接到 OSD，访问 PG 以及读取或写入对象数据。

若要将放置组映射到 OSD，CRUSH map 定义了 bucket 类型的层次结构。bucket 类型的列表位于生成的 CRUSH map 中的 **类型** 下。创建 bucket 层次结构的目的是通过故障域和/或性能域来隔离 leaf 节点，如驱动器类型、主机、机箱、机架、电源管理单元、pod、行、房间和数据中心。

除了代表 OSD 的叶节点外，层次结构的其余部分是任意的。如果默认类型不符合其要求，管理员可以根据自己的需要对其进行定义。CRUSH 支持向无形图建模 Ceph OSD 节点，通常采用层次结构。因此，Ceph 管理员可以通过单个 CRUSH 映射中的多个根节点来支持多个层次结构。例如，管理员可以创建一个层次结构，代表更高的成本 SSD，以及采用 SSD 日志实现性能降低的成本硬盘驱动器的单独层次结构。

2.5. CEPH 输入/输出操作

Ceph 客户端从 Ceph 监视器检索"Cluster Map"，并绑定到池，并在池中 PG 内对对象执行输入/输出 (I/O)。该池的 CRUSH 规则集和放置组数量是确定 Ceph 如何放置数据的主要因素。使用最新版本的 cluster map，客户端知道集群中所有 monitor 和 OSD 及其当前状态。**但是，客户端不知道对象位置的任何内容。**

客户端需要的唯一输入是对象 ID 和池名称。非常简单：Ceph 将数据存储在指定池中。当客户端想将指定对象存储在池中时，它会获取对象名称、哈希代码、池中的 PG 数量以及输入的 CRUSH（可扩展哈希下）计算 PG 的 ID，以及 PG 的 Primary OSD。

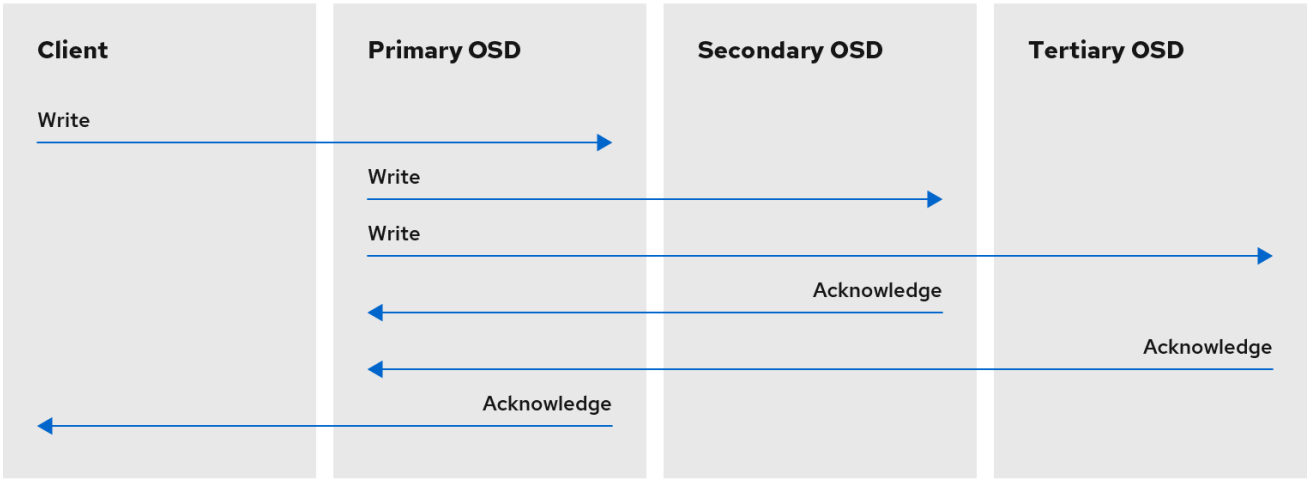
Ceph 客户端使用以下步骤来计算 PG ID。

1. 客户端输入池 ID 和对象 ID。例如，**pool = liverpool** 和 **object-id = john**。
2. CRUSH 采用对象 ID 并进行哈希。
3. CRUSH 计算 PG 数量的 hash modulo 来获取 PG ID。例如 **58**。
4. CRUSH 计算与 PG ID 对应的 Primary OSD。
5. 客户端获取指定池名称的池 ID。例如，池 **liverpool** 是池号 **4**。
6. 客户端将池 ID 添加到 PG ID 中。例如： **4.58**。
7. 客户端通过直接与 Acting 设置中的 Primary OSD 通信来执行对象操作，如 write、read 或 delete。

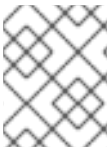
Ceph 存储集群的拓扑和状态在会话期间相对稳定。通过 **librados** 赋予 Ceph 客户端计算对象位置比要求客户端通过每个读/写操作的聊天性会话向存储集群进行查询要快。CRUSH 算法允许客户端计算对象应该存储的位置，并**允许客户端直接联系 acting set 中的 Primary OSD**以存储或检索对象中的数据。由于 exabyte 扩展的集群具有数千 OSD，因此客户端和 Ceph OSD 之间的订阅通过订阅并不是显著的问题。如果集群状态发生变化，客户端只需从 Ceph 监视器请求对 cluster map 的更新。

2.6. CEPH 复制

与 Ceph 客户端一样，Ceph OSD 也可联系 Ceph 监视器来检索 cluster map 的最新副本。Ceph OSD 也使用 CRUSH 算法，但它们使用它来计算对象存储对象的副本。在典型的写入场景中，Ceph 客户端使用 CRUSH 算法计算对象的 Acting 设置中的 PG ID 和 Primary OSD。当客户端将对象写入到 Primary OSD 时，Primary OSD 会找到它应当存储的副本数。该值在 **osd_pool_default_size** 设置中找到。然后，Primary OSD 采用对象 ID、池名称和 cluster map，并使用 CRUSH 算法计算用于操作集合的次要 OSD 的 ID。Primary OSD 将对象写入二级 OSD。当 Primary OSD 收到来自次要 OSD 和 Primary OSD 本身的确认，并且 Primary OSD 本身完成其写入操作时，它会确认对 Ceph 客户端的写入操作成功。



由于能够代表 Ceph 客户端执行数据复制功能，Ceph OSD 守护进程从这一操作过程中减轻 Ceph 客户端，同时确保高数据可用性和数据安全性。



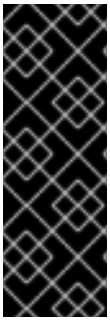
注意

Primary OSD 和二级 OSD 通常配置为位于单独的故障域中。CRUSH 通过考虑故障域来计算次要 OSD 的 ID。

数据副本

在复制池中，Ceph 需要对象的多个副本才能以降级状态运行。理想情况下，Ceph 存储集群可让客户端读取和写入数据，即使操作集合中的一个 OSD 失败。因此，Ceph 默认为对写操作进行至少两个副本的对象的三个副本。即使两个 OSD 失败，Ceph 仍会保留数据。但是，它将中断写操作。

在纠删代码池中，Ceph 需要在多个 OSD 之间存储对象的区块，以便它可以处于降级状态。与复制池类似，理想的纠删代码池使 Ceph 客户端能够以降级状态读取和写入。



重要

红帽支持对 *k* 和 *m* 的以下 *jerasure* 编码值：

- k=8 m=3
- k=8 m=4
- k=4 m=2

2.7. CEPH 纠删代码

Ceph 可以加载许多纠删代码算法之一。最早且最常用的是 **Reed-Solomon** 算法。纠删代码实际上是一个转发错误修正(FEC)代码。FEC 代码会将 **K** 块的消息转换为较长的消息，称为"代码字"的 **N** 块，以便 Ceph 可以从 **N** 块的子集恢复原始消息。

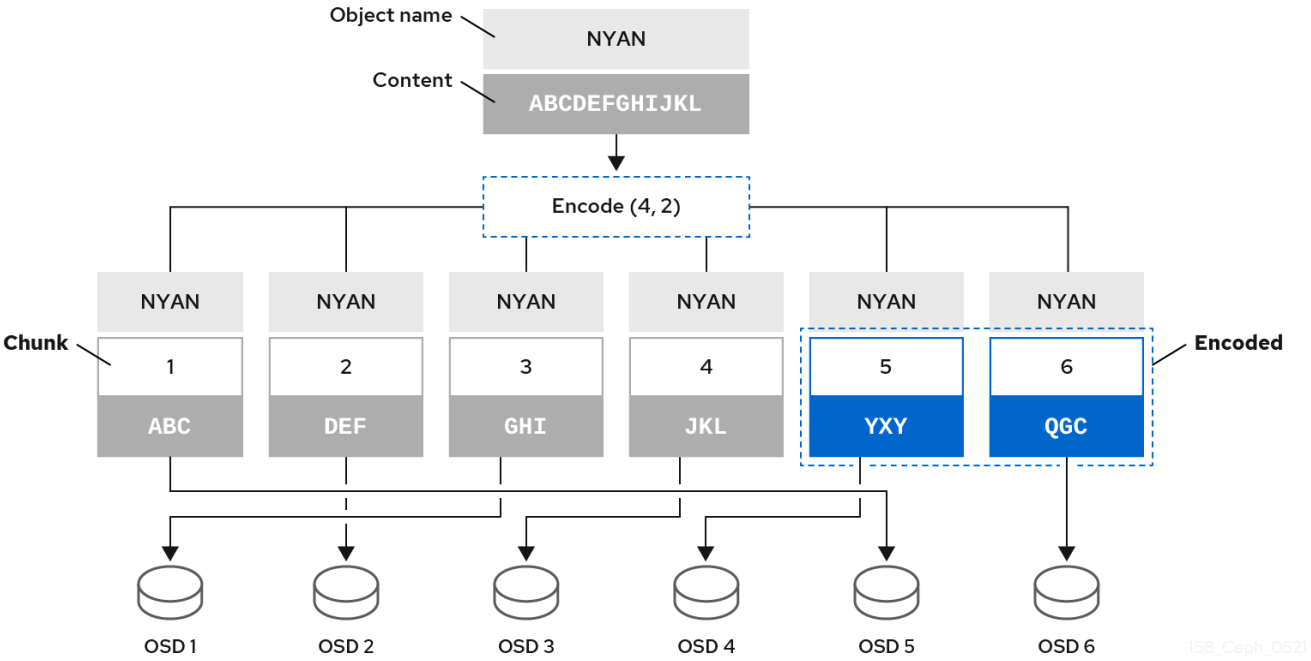
更具体地说，**N = K+M**，其中变量 **K** 是原始数据块的数量。变量 **M** 代表额外的或者冗余区块，它添加了纠删代码算法，以提供防故障的保护。变量 **N** 是纠删代码池进程后创建的区块总数。**M** 的值是 **N-K**，这意味着算法计算来自 **K** 原始数据块的 **N-K** 冗余块。此方法可确保 Ceph 可以访问所有原始数据。系统可应对任意 **N-K** 失败。例如，在 10 **K** 中是 16 **N** 配置，或者纠删代码 **10/16**，纠删代码算法向 10 个基本区

块 **K** 增加了六个额外的区块。例如，在 **M = K-N** 或 **16-10 = 6** 配置中，Ceph 会将 16 个块 **N** 分布到 16 个 OSD 中。即使有 6 个 OSD 无法丢失数据，也可以从 10 个验证的 **N** 块重新创建原始文件，从而确保非常高的容错能力。

与复制池一样，在纠删代码池中，设置中的 Primary OSD 会接收所有写入操作。在复制池中，Ceph 在集合的次要 OSD 上的 PG 中对 PG 中的各个对象进行深度副本。对于纠删代码，进程略有不同。纠删代码池将每个对象存储为 **K+M** 块。它被分为 **K** 数据区块和 **M** 编码区块。该池配置为具有 **K+M** 大小，以便 Ceph 将每个块存储到操作集的 OSD 中。Ceph 将块的排名存储为对象的属性。Primary OSD 负责将载荷编码到 **K+M** 区块，并将它们发送到其他 OSD。Primary OSD 还负责维护 PG 日志的权威版本。

例如，在典型的配置中，系统管理员会创建一个纠删代码池以使用六个 OSD 并保持丢失两个 OSD。也就是说，(**K+M = 6**)， (**M = 2**)。

当 Ceph 将包含 **ABCDEFGHIJKL** 的对象 **NYAN** 写入池时，通过简单地将内容划分为四个部分，即 **ABC**、**DEF**、**GHI** 和 **JKL**。如果内容长度不是 **K** 的倍数，则该算法会对内容进行 pad 处理。此函数还会创建两个编码块：第五个带有 **YXY**，以及第六个带有 **QGC** 的编码块。Ceph 将 OSD 上的各个块存储在执行集合中，其中将区块存储到具有相同名称 **NYAN** 但驻留于不同的 OSD 的对象中。除了其名称外，该算法还必须保留创建块作为对象 **shard_t** 的属性的顺序。例如，Chunk 1 包含 **ABC** 和 Ceph 存储在 **OSD5** 上，而块 5 包含 **YXY**，Ceph 则存储在 **OSD4** 上。

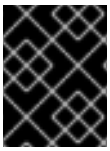


在恢复场景中，客户端尝试通过读取块 1 到 6 的块从纠删代码池中读取对象 **NYAN**。OSD 告知算法缺少块 2 和 6。这些缺少的块被称为 'erasures'。例如，由于 OSD6 不足而无法读取块 6，因此 **OSD6** 无法读取块 2，因为 **OSD2** 是最慢的，并且其块没有考虑。但是，当算法有四个块时，它会读取四个块：块 1，包含 **ABC**、块 3 包含 **GHI**、块 4 包含 **JKL**，以及包含 **YXY** 的块 5。然后，它会重建对象 **ABCDEFGHIJKL** 的原始内容，以及包含 **QGC** 的块 6 的原始内容。

将数据分割为块独立于对象放置。CRUSH 规则集和纠删代码池配置文件决定了 OSD 上的区块放置。例如，在纠删代码配置集中使用 Locally Repairable Code(**lrc**)插件会创建额外的块，需要较少的 OSD 从中恢复。例如，在 **lrc** 配置集配置 **K=4 M=2 L=3** 中，该算法会创建六个块(**K+M**)，就像 **jerasure** 插件一样，但本地的特性值(**L=3**)要求算法在本地创建 2 个块。该算法会创建额外的块，例如 **(K+M)/L**。如果包含块 0 的 OSD 失败，可以使用块 1、2 和第一个本地块来恢复此块。在这种情况下，算法只需要 3 个块而不是 5 个就能恢复。

注意

使用纠删代码的池禁用 Object Map。



重要

对于带有 2+2 配置的纠删代码池，请将 **ABCDEFGHIJKL** 的输入字符串替换为 **ABCDEF**，并将编码区块从 **4** 替换为 **2**。

其它资源

- 有关 CRUSH、纠删代码 profile 和插件的更多信息，请参见 Red Hat Ceph Storage 7 的存储策略指南。
- 如需有关 Object Map 的更多信息，请参阅 *Ceph 客户端对象映射*部分。

2.8. CEPH OBJECTSTORE

对**象存储** 为 OSD 的原始块设备提供低级接口。当客户端读取或写入数据时，它会与 **ObjectStore** 接口交互。Ceph 写入操作基本上是 ACID 事务：即，它们提供 **Atomicity**、**Consistency**、**Isolation** 和 **Durability**。对**象存储** 可确保 **交易**可以通过"全部或完全没有"的方式保证 **原子性**。**ObjectStore** 也会处理对象语义。存储集群中存储的对象具有唯一标识符、对象数据和元数据。因此，**ObjectStore** 通过确保 Ceph 对象语义正确来提供**一致性**。**ObjectStore** 通过在写操作中调用 **Sequencer** 来提供 ACID 事务的 **Isolation** 部分，以确保 Ceph 的写入操作会按顺序进行。与之相反，OSD 复制或擦除编码功能提供了 ACID 事务的 **Durability** 组件。由于 **ObjectStore** 是存储介质的低级别接口，因此还提供性能统计数据。

Ceph 实施多种统一方法来存储数据：

- **BlueStore：**使用原始块设备来存储对象数据的生产评分实施。
- **Memstore：** 开发人员在 RAM 中直接测试读写操作。
- **k/V 存储：** Ceph 使用键/值数据库的内部实施。

由于管理员通常仅解决 **BlueStore**，因此以下小节将更详细地描述这些实施。

2.9. CEPH BLUESTORE

BlueStore 是 Ceph 的当前存储实施。它在小分区上使用非常轻量的 **BlueFS** 文件系统用于其 k/v 数据库，并消除了代表放置组的目录范式，一个代表元数据的文件和文件 XATTRs。

BlueStore 存储数据，如下所示：

- **对象数据：** 在 **BlueStore** 中，Ceph 将对象存储在原始块设备上作为块。存储对象数据的原始块设备的一部分不包含文件系统。文件系统的遗漏消除了间接的层，从而提高了性能。但是，其中大部分的 **BlueStore** 性能改进来自 block 数据库和 write-ahead 日志。
- **Block Database：** 在 **BlueStore** 中，块数据库处理对象语义，以保证 **一致性**。对象的唯一标识符是 block 数据库中的一个键。block 数据库中的值由一系列块地址组成，它们引用所存储的对象数据、对象的 PG 和对象元数据。块数据库可能驻留在存储对象数据的同一原始块设备的 **BlueFS** 分区上，或者它可能驻留在单独的块设备上，通常当主块设备是硬盘，并且 SSD 或 NVMe 将提高性能。**BlueStore** 的键/值语义不会影响文件系统 XATTRs 的限制。**BlueStore** 可以在块数据库中快速将对象分配给其他放置组，而无需将文件从一个目录移动到另一个目录。块

数据库可以存储所存储对象数据的校验和及其元数据，允许每个读取的完整数据校验和操作比定期清理更加高效，从而检测位漫长。**BlueStore** 可以压缩对象，而块数据库可以存储用于压缩对象理解的算法，读取操作可选择用于解压缩的相应算法。

- write **-ahead Log:** 在 **BlueStore** 中，write-ahead 日志确保 **Atomicity**，并记录每个事务的所有方面。**BlueStore** write-ahead 日志或 WAL 可以同时执行此功能。BlueStore 可以在同一设备上部署 WAL 来存储对象数据，或者可以在另一个设备上部署 WAL，通常是主块设备是硬盘，而 SSD 或 NVMe 将提高性能。

注意

如果不同的设备比主存储设备快于主存储设备，则只有在单独的块设备中保存块数据库或 write-ahead 日志会很有用。例如，SSD 和 NVMe 设备通常比 HDD 快。由于工作负载中的差异，将块数据库和 WAL 放置到单独的设备中也可能具有性能优势。

2.10. CEPH 自我管理操作

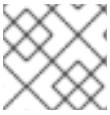
Ceph 集群自动执行许多自我监控和管理操作。例如，Ceph OSD 可以检查集群运行状况，再返回到 Ceph 监视器。通过使用 CRUSH 将对象分配到放置组和放置组到一组 OSD，Ceph OSD 可以使用 CRUSH 算法来重新平衡集群，或者动态从 OSD 故障中恢复。

2.11. CEPH 心跳

Ceph OSD 加入集群，并报告给 Ceph Monitor 的状态。在最低级别，Ceph OSD 的状态为 **up** 或 **down**，反映它是否正在运行并且能够服务 Ceph 客户端请求。如果 Ceph OSD 的状态为 **down**，且在 **（in）** Ceph 存储集群中，这可能代表 Ceph OSD 失败。例如，如果 Ceph OSD 未运行，它会崩溃 Ceph OSD 无法通知 Ceph 监控器它的状态已变为 **down**。Ceph Monitor 可以定期 Ping Ceph OSD 守护进程，以确保它正在运行。但是，心跳功能会帮助 Ceph OSD 来确定邻居 OSD 的状态是否为 **down**，以更新 cluster map，并将它报告给 Ceph 监控器。这意味着 Ceph 监控器可以保持轻巧的进程。

2.12. CEPH 对等点

Ceph 在多个 OSD 上存储放置组副本。PG 的每个副本都处于状态。这些 OSD "peer" 相互检查，以确保它们同意 PG 的每个副本的状态。对等问题通常会自己解决。



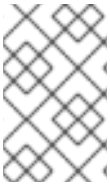
注意

当 Ceph 监视器同意存储 PG 的 OSD 状态时，这并不意味着放置组具有最新的内容。

当 Ceph 在一个 OSD 的操作组中保存放置时，会将它们称为 *Primary, Secondary* 等。按照惯例，*Primary* 是*操作组*中的第一个 OSD。存储 PG 的第一个副本的 *Primary* 负责协调该 PG 的对等进程*Primary* 是**唯一一个**可以接受客户端发起的对于一个给定放置组的写入对象的 OSD（充当 *Primary*）。

Acting Set 是一系列 OSD，它负责存储 PG。*Acting Set* 可能会引用当前负责 PG 的 Ceph OSD 守护进程，或者以一些时期负责特定放置组的 Ceph OSD 守护进程。

作为 *Acting Set* 一部分的 Ceph OSD 守护进程可能并不总是为 **up**。当 *Acting Set* 中的一个 OSD 为 **up** 时，它从属于 *Up Set*。*Up Set* 是一个重要的区别，因为在一个 OSD 失败时，Ceph 可以重新将 PG 映射到其他 Ceph OSD。



注意

在包含 **osd.25**、**osd.32** 和 **osd.61** 的 PG 的 *Acting Set* 中，第一个 OSD **osd.25** 是 *Primary*。如果该 OSD 失败，*Secondary*,**osd.32** 变为 *Primary*，并且 Ceph 将从 *Up Set* 中移除 **osd.25**。

2.13. CEPH 重新平衡和恢复

当管理员将 Ceph OSD 添加到 Ceph 存储集群时，Ceph 会更新 cluster map。对 cluster map 的这一更改也会更改对象放置，因为修改后的集群映射会更改 CRUSH 计算的输入。CRUSH 均匀地放置数据，但随机进行伪造。因此，当管理员添加新 OSD 时，只有少量的数据才会移动。数据量通常是新 OSD 的数量，它们通常根据集群中的总数据量来划分。例如，在添加 OSD 时，具有 50 个 OSD 的集群中，数据的 1/50th 或 2% 数据可能会在添加 OSD 时移动。

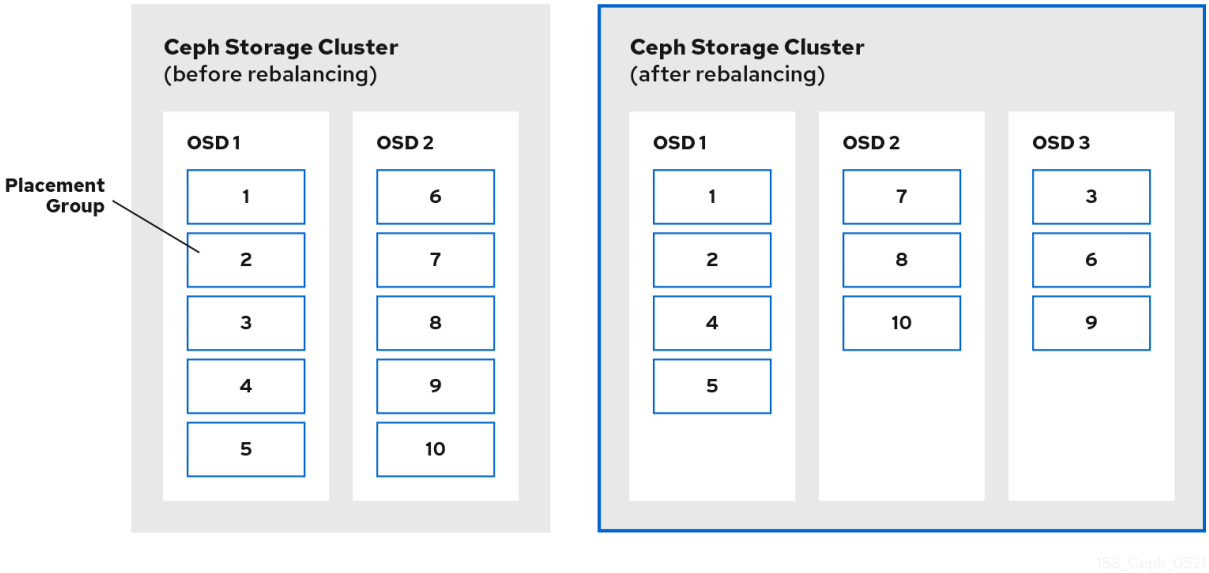
下图说明了重新平衡流程，其中部分，但并非所有 PG 都从现有 OSD、OSD 1 和 2 在图中迁移到新的 OSD、OSD 3。即使是重新平衡，CRUSH 仍保持稳定。许多放置组保留在其原始配置中，并且每个 OSD 获取一些额外的容量，因此在集群重新平衡后，新的 OSD 并没有负载激增。

Ceph 中有 2 个类型的负载均衡器：

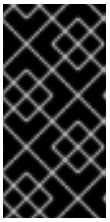
- **容量平衡** ：
  容量平衡是一项功能需求。当一个设备已满时，系统无法再处理写入请求。为了避免填满设备，务必要以公平的方式在设备间平衡容量。每个设备都必须获得与其大小成比例的容量，以便所有设备都有相同的完整级别。容量平衡会在 OSD 上创建公平共享工作负载，以便从性能角度写入请求。

  容量平衡需要数据移动，而且非常耗时，因为它需要时间来平衡系统。

  为了获得最佳性能，请确保所有设备都同构（相同大小和性能）。



- **read balancing**: [技术预览]



重要

红帽产品服务级别协议（SLA）不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。如需了解更多详细信息，请参阅红帽技术预览功能的支持范围。

读平衡是性能需求。它通过确保每个设备获得主 OSD 的公平共享，以便读取请求在集群中的 OSD 之间分布，从而提高系统性能。由于链中最弱的链接，未平衡的读取请求会导致负载不良，并降低集群读取带宽。读平衡成本较低，操作会快，因为没有涉及数据移动。这是一个元数据操作，其中 osdmap 已更新，以更改参与 pg 中的 OSD 是 primary。

读负载平衡仅支持复制池，不支持纠删代码池。读平衡不会考虑 OSD 的设备类别和 DR 解决方案的可用区。离线工具可用来使用读平衡功能，您需要在集群的每个池上运行流程。您需要在每次自动扩展更改后再次运行读取平衡程序。

为获得最佳读取性能，请确保所有设备都同构（相同大小和性能）并且您已平衡容量。

2.14. CEPH 数据完整性

作为维护数据完整性的一部分，Ceph 提供了大量机制来保护损坏的磁盘扇区和位访问权限。

- **Scrubbing:** Ceph OSD 守护进程可以在 PG 内清理对象。也就是说，Ceph OSD 守护进程可以将一个 PG 中的对象元数据与存储在其他 OSD 上的放置组中的副本进行比较。Scrubbin- 通常执行每日处理错误或存储错误。Ceph OSD 守护进程也通过比较对象位(for-bit)中的数据来执行深度清理。深度清理 - 通常会在不明显清理的驱动器上执行每周发现的扇区。

- **CRC 检查：** 在 Red Hat Ceph Storage 7 中使用 **BlueStore** 时，Ceph 可以在写操作上执行循环冗余检查(CRC)来确保数据完整性；然后在块数据库中存储 CRC 值。在读取操作中，Ceph 可以从块数据库检索 CRC 值，并将它与检索数据的 CRC 进行比较，以确保数据完整性。

2.15. CEPH 高可用性

除了 CRUSH 算法所启用的高可扩展性外，Ceph 还必须维护高可用性。这意味着，即使集群处于降级状态或 monitor 失败时，Ceph 客户端必须能够读取和写入数据。

2.16. 集群 CEPH 监控器

在 Ceph 客户端能够读取或写入数据之前，它们必须联系 Ceph 监控器来获取 cluster map 的最新副本。Red Hat Ceph Storage 集群可以使用单一监控器操作；但是，这引入了单点故障。也就是说，如果 monitor 停机，Ceph 客户端无法读取或写入数据。

为添加可靠性和容错功能，Ceph 支持 monitor 集群。在 Ceph Monitor 的集群中，延迟和其他故障可能会导致一个或多个 monitor 在集群的当前状态后面出现。因此，Ceph 在与存储集群状态相关的各种监控实例之间必须达成一致。Ceph 始终使用大多数 monitor 和 Paxos 算法来建立关于存储集群当前状态的监控器之间的共识。Ceph 监控节点需要 NTP 来防止时钟偏移。

存储管理员通常使用奇数个监控器来部署 Ceph，从而确定大部分的运行效率更高。例如，多数可以是 1, 2:3, 3:5, 4:6 等。

第 3 章 CEPH 客户端组件

Ceph 客户端在自己的材料上有所不同，即它们如何展示数据存储接口。Ceph 块设备提供块存储，它像物理存储驱动器一样挂载。Ceph 网关为对象存储服务提供符合 S3 兼容和 Swift 兼容的 RESTful 接口，具有其自己的用户管理。但是，所有 Ceph 客户端都使用可靠的自主分布式对象存储(RADOS)协议与红帽 Ceph 存储集群交互。

它们都有相同的基本需求：

- Ceph 配置文件和 Ceph 监视器地址。
- 池名称。
- 用户名和到 secret 密钥的路径。

Ceph 客户端往往遵循一些相似的特征，如 object-watch-notify 和 striping。以下小节介绍了一些关于 RADOS、libradrad 和 Ceph 客户端中使用的常见模式的信息。

先决条件

- 对分布式存储系统有基本了解。

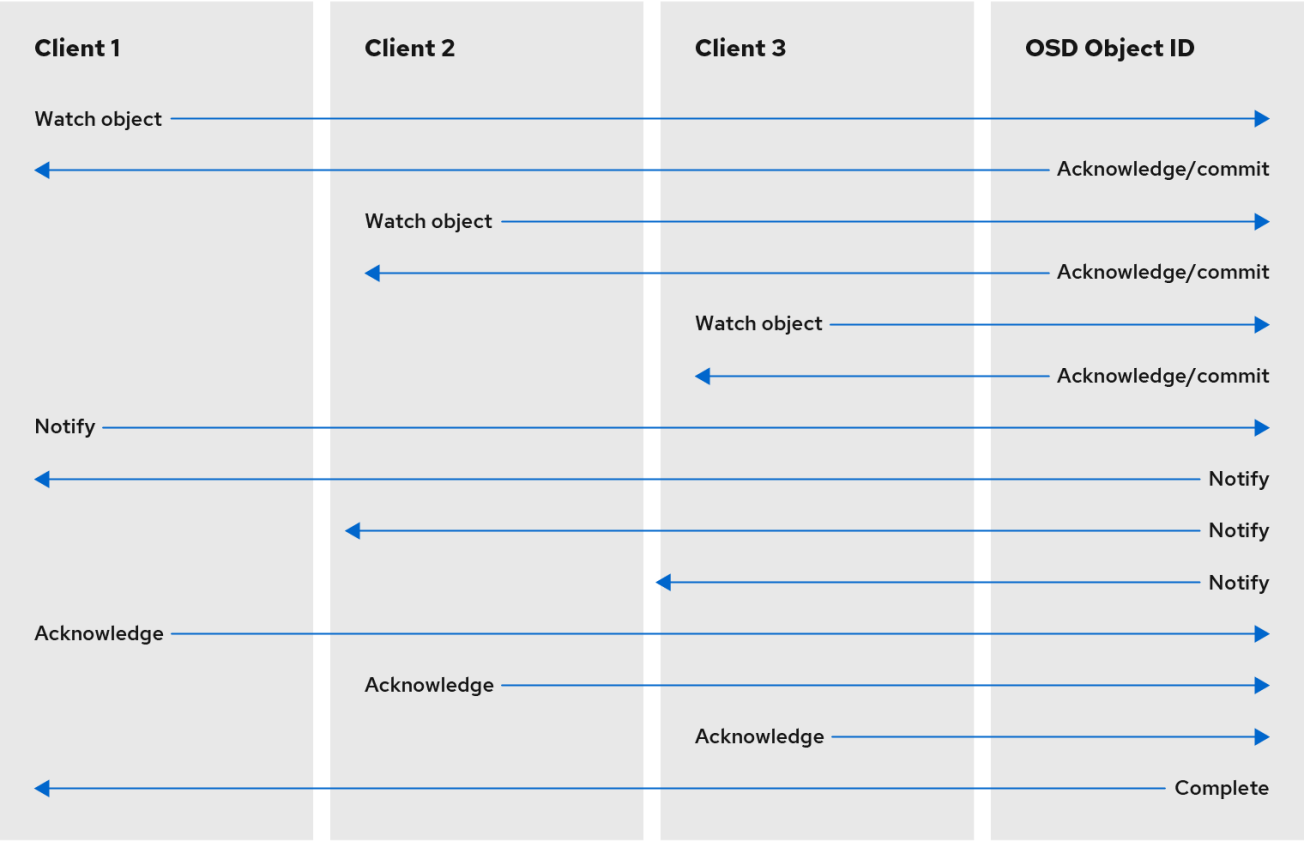
3.1. CEPH 客户端原生协议

现代应用需要简单的对象存储接口，同时具备异步通信功能。Ceph Storage 集群提供一个简单的对象存储接口，具有异步通信功能。界面提供对整个集群对象的直接并行访问。

- 池操作
- 快照
- 读/写对象
  - 创建和删除
  - 整个对象或指定范围
  - 附加或截断
- Create/Set/Get/Remove XATTRs
- create/Set/Get/Remove Key/Value Pairs
- compound 操作和双键语义

3.2. CEPH 客户端对象监视和通知

Ceph 客户端可以对对象注册永久关注，并使会话处于打开的 OSD。客户端可将通知消息和有效载荷发送到所有观察者，并在观察者收到通知时接收通知。这可让客户端使用任何对象作为同步/通信频道。



3.3. CEPH 客户端 MANDATORY EXCLUSIVE LOCKS

Mandatory Exclusive Locks 是一种将 RBD 锁定到单个客户端的功能（如果有多个挂载）。当多个挂载的客户端试图写入同一对象时，这有助于解决写入冲突的情况。此功能基于 **object-watch-notify**（在上一节中进行了介绍）。因此，在编写时，如果一个客户端首先在对象上建立一个专用锁定，则另一个挂载的客户端首先会检查一下对等点是否已锁定到对象后再写入。

启用此功能后，只有一个客户端一次可以修改 RBD 设备，特别是在操作过程中更改内部 RBD 结构（如**快照创建/删除**）。它还为失败的客户端提供一些保护。例如，如果虚拟机似乎没有响应，并且您在其他位置启动同一磁盘的副本，则第一个虚拟机将被放入 Ceph 中，且无法破坏新的磁盘。

默认情况下不启用强制锁定。在创建镜像时，您必须使用 **--image-feature** 参数显式启用它。

示例

```
[root@mon ~]# rbd create --size 102400 mypool/myimage --image-feature 5
```

此处，数字 **5** 是 **1** 和 **4** 的总和，**1** 个启用分层支持，**4** 个启用了 exclusive locking 支持。因此，上述命令将创建一个 100 GB rbd 镜像，启用分层和专用锁定。

Mandatory Exclusive Locks 也是对**象映射**的先决条件。如果没有启用专用锁定支持，无法启用对象映射支持。

Mandatory Exclusive Locks 也会为镜像进行一些操作。

3.4. CEPH 客户端对象映射

对象映射是在客户端写入到 rbd 镜像时跟踪支持的 RADOS 对象存在的功能。发生写入时，该写入将转换为后备 RADOS 对象内的偏移值。启用对象映射功能后，会跟踪这些 RADOS 对象是否存在。因此，我们可以知道对象是否确实存在。对象映射在 librbd 客户端上保留在内存中，这样可以避免查询它知道不存在的对象的 OSD。换句话说，对象映射是实际存在的对象的索引。

对象映射对某些操作很有用，viz：

- 调整大小
- Export
- Copy
- Flatten
- 删除
- 读

缩小大小操作就像是删除尾随对象所被删除的部分一样。

导出操作知道要从 RADOS 请求哪些对象。

复制操作知道哪些对象存在并需要复制。它不一定要迭代可能存在数百个和数千个可能的对象。

flatten 操作为所有父对象执行备份到克隆，以便克隆可以从父脱离，例如子克隆中到父快照的指代可以被删除。因此，只有存在的对象才能进行复制，而不是所有潜在的对象。

删除操作只删除镜像中存在的对象。

读取操作会跳过它所知的对象的读取操作。

因此，为了像调整大小、缩减、导出、复制、扁平化和删除等操作，这些操作需要针对所有可能影响的 RADOS 对象发出操作，无论它们是否存在。启用对象映射时，如果对象不存在，则不需要发出操作。

例如，如果我们有一个 1 TB 稀疏 RBD 镜像，它可以有数百个和数千个支持的 RADOS 对象。当删除操作没有启用对象映射时，则需要为镜像中的每个潜在对象发出一个 **remove object** 操作。但是，如果启用了对象映射，只需要为存在的对象发出**删除对象**操作。

对象映射对于没有实际对象但从父对象获取对象的克隆而言非常宝贵。当克隆的镜像存在时，克隆最初没有对象，所有读取都会被重定向到父镜像。因此，对象映射可以在没有对象映射的情况下提高读取操作，首先需要将读取操作向 OSD 发出克隆，当失败时，它会向父项发出另一个读取并启用了对象映射。它跳过它知道的对象的读取并不存在。

默认情况下不启用对象映射。在创建镜像时，您必须使用 **--image-features** 参数显式启用它。另外，**Mandatory Exclusive Locks** 是**对象映射**的先决条件。如果没有启用专用锁定支持，无法启用对象映射支持。要在创建镜像时启用对象映射支持，请执行：

```
[root@mon ~]# rbd -p mypool create myimage --size 102400 --image-features 13
```

此处，**13** 是 **1**、**4** 和 **8** 的总和，其中 **1** 个启用分层支持，**4** 个启用了 exclusive locking 支持， **8** 个启用对象映射支持。因此，上述命令将创建一个 100 GB rbd 镜像，启用分层、专用锁定和对象映射。

3.5. CEPH 客户端数据剥离

存储设备存在吞吐量限制，这会影响性能和可扩展性。因此，存储系统通常支持跨多个存储设备的连续部

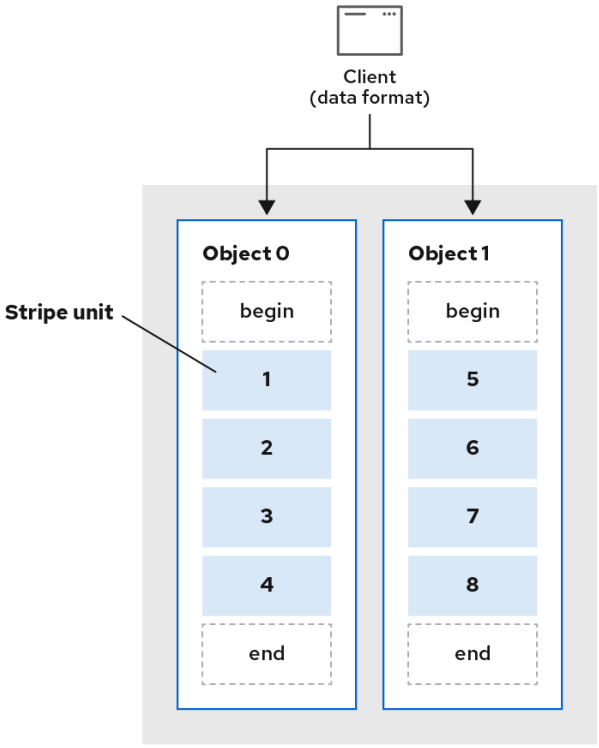
分信息 - 来提高吞吐量和性能。最常用的数据条带形式来自 RAID。RAID 类型与 Ceph 的条带最相似，是 RAID 0 或 'striped 卷'。Ceph 的条带化提供了 RAID 0 条带的吞吐量，即 n-way RAID 镜像的可靠性并更快地恢复。

Ceph 提供三种类型的客户端：Ceph 块设备、Ceph Filesystem 和 Ceph Object Storage。Ceph 客户端将其数据从它提供的格式转换为其用户，如块设备镜像、RESTful 对象和 CephFS 文件系统目录，并放入 Ceph 存储集群中存储的对象。

提示

Ceph 在 Ceph Storage 集群中存储的对象不是条带的。Ceph Object Storage、Ceph Block Device 和 Ceph Filesystem 在多个 Ceph Storage Cluster 对象上分条数据。使用 **librados** 直接写入到 Ceph 存储集群的 Ceph 客户端必须执行条带化，并且本身并行 I/O 以获得这些好处。

最简单的 Ceph 分条格式涉及 1 对象的条带数。Ceph 客户端将条带单元写入 Ceph Storage Cluster 对象到其最大容量，然后为数据的额外条带创建另一个对象。最简单的分条形式可能足以用于小型块设备镜像、S3 或 Swift 对象。但是，这种简单形式并不能最大利用 Ceph 在放置组之间分发数据的能力，因而不会提高性能。下图显示了最简单的条带形式：



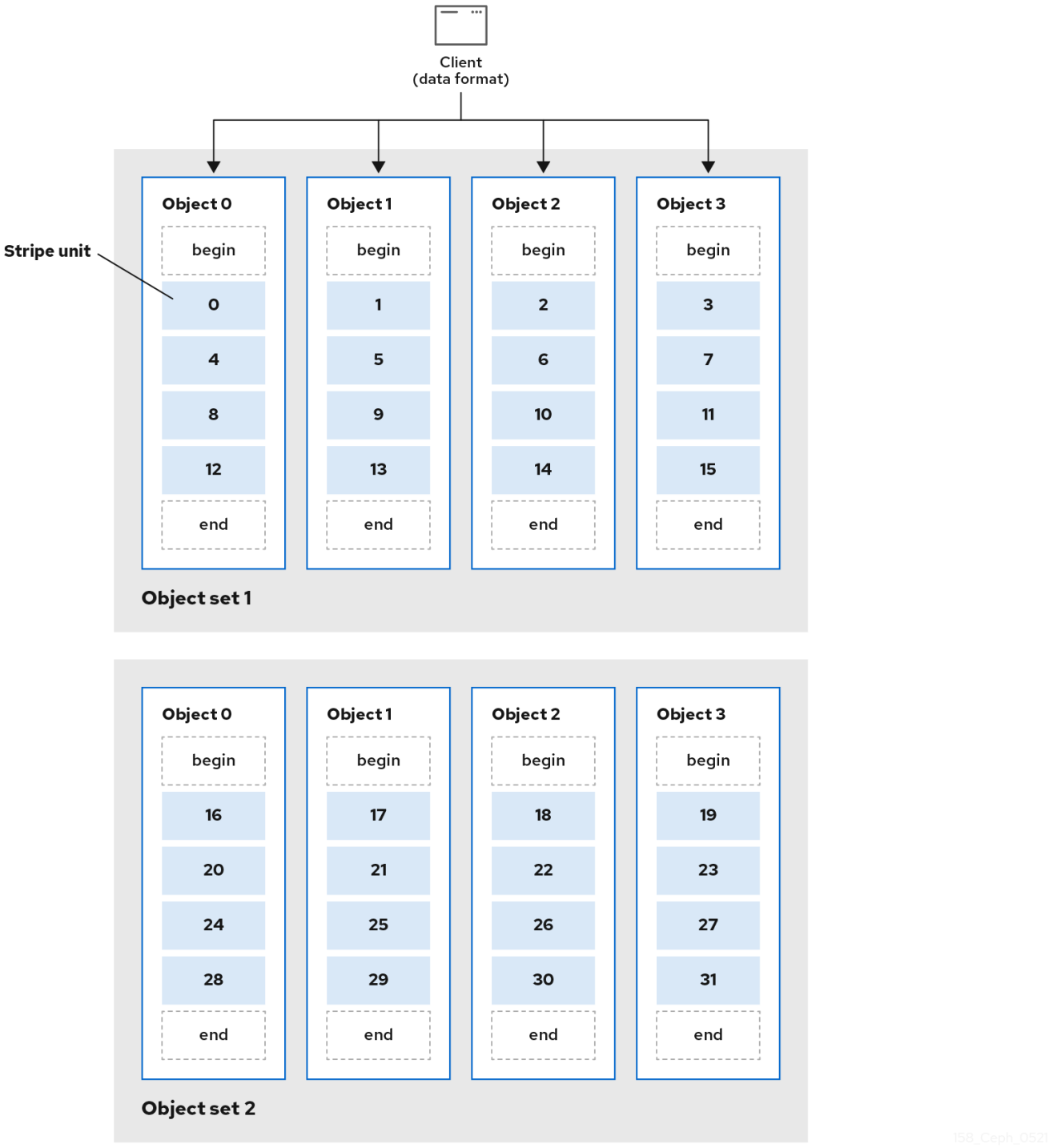
如果您预计大镜像大小，例如视频，则可能会看到大型 S3 或 Swift 对象，通过在对象集中的多个对象间分条客户端数据，从而提高了可读写的性能。当客户端并行将条状单元写入其对应对象时，会发生重大写入性能。由于对象被映射到不同的放置组，并且进一步映射到不同的 OSD，每个写入操作在最大写入速度上并行。对单磁盘的一个写入会限制磁头的移动（例如每个搜索 6ms）和设备的带宽（例如 100MB/s）。Ceph 通过分散在多个对象上写入多个对象（映射到不同的放置组和 OSD），Ceph 可以减少每个驱动器的查找数量，并组合多个驱动器的吞吐量，以实现更快的写入或读取速度。

注意

条带独立于对象副本。由于 CRUSH 在 OSD 之间复制对象，因此会自动复制条带。

在以下示意图中，客户端数据被分散到包括了 4 个对象的对象集（在下图中是 **object set 1**），其中第一

个条带单元是 **object 0** 中的 **stripe unit 0**，第四个条带单元是 **object 3** 中的 **stripe unit 3**。编写第四个条带后，客户端将确定对象集是否满。如果对象集没有满，客户端会开始重新写入第一个对象，在下图中为 **object 0**。如果对象集已满，客户端会创建一个新对象集，在下图中为 **object set 2**，并开始写入第一个条带，分条单元为 16，在新对象集中的第一个对象，在下图中为 **object 4**。



三个重要变量决定了 Ceph 条带数据的方式：

- **Object Size**：Ceph Storage 集群中的对象具有最大可配置的大小，如 2 MB 或 4 MB。对象大小应该足够大，以适应许多条分条单元，它应该是条带单元的倍数。

重要

红帽建议安全最大值为 16 MB。

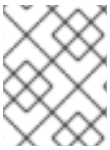
- **条带宽度：** 条带化有一个可配置的单位大小，例如 64 KB。Ceph 客户端将写入对象的数据分到平等的条状单元中，但最后条带单元除外。分条宽度应该是 Object Size 的比例，以便对象可以包含很多条带单元。

- **Stripe Count：** Ceph 客户端将一组由条带数决定的对象写入条带单元序列。一系列对象称为对象集。Ceph 客户端写入对象集合中的最后一个对象后，它会返回对象集中的第一个对象。

重要

在将集群放置到生产环境中前测试条带配置的性能。在分条数据并将其写入对象后，您的 CANNOT 会更改这些分条参数。

Ceph 客户端将数据分条数据到分条单元，并将分条单元映射到对象后，Ceph 的 CRUSH 算法会将对象映射到放置组，而放置组则 map 到 Ceph OSD 守护进程，然后对象存储在存储磁盘上的文件。



注意

由于客户端写入到单个池，因此所有数据分条到对象都会映射到同一池中的放置组。因此它们使用相同的 CRUSH map 和相同的访问控制。

3.6. CEPH ON-WIRE 加密

您可以使用 messenger 版本 2 协议通过网络为所有 Ceph 流量启用加密。**me** senger v2 的安全模式设置加密 Ceph 守护进程和 Ceph 客户端之间的通信，从而为您提供端到端加密。

Ceph on-wire 协议 **msgr2** 的第二个版本包括几个新功能：

- 安全模式通过网络加密所有数据。

- 身份验证有效负载的封装改进。

- 功能公告和协商的改进。

Ceph 守护进程绑定到多个端口，允许旧旧 v1- 兼容新的 v2 兼容 Ceph 客户端，以连接同一存储集群。Ceph 客户端或其他 Ceph 守护进程连接到 Ceph Monitor 守护进程将先尝试使用 **v2** 协议（如果可能），但若不可能，则使用旧的 **v1** 协议。默认情况下，启用 messenger 协议 **v1** 和 **v2**。新的 v2 端口为 3300，旧的 v1 端口默认为 6789。

messenger v2 协议有两个配置选项，用于控制是否使用 v1 还是 v2 协议：

- **ms_bind_msgr1** - 这个选项控制守护进程是否绑定到 v1 协议的端口，默认为 **true**。

- **ms_bind_msgr2** - 这个选项控制守护进程是否绑定到 v2 协议的端口，默认为 **true**。

同样，两个选项根据使用的 IPv4 和 IPv6 地址进行控制：

- **ms_bind_ipv4** - 此选项控制守护进程是否绑定到 IPv4 地址；默认为 **true**。

- **ms_bind_ipv6** - 这个选项控制守护进程是否绑定到 IPv6 地址，默认为 **true**。

msgr2 协议支持两种连接模式：

- **crc**
    - 当使用 **cephx** 建立连接时，提供强大的初始身份验证。

- 提供 **crc32c** 完整性检查，以防止比特反转（bit flipping）攻击。
- 不能提供对恶意的中间人攻击提供保护。
- 不能阻止对认证后的网络流量进行窃听。

- **secure**
  - 当使用 **cephx** 建立连接时，提供强大的初始身份验证。
  - 提供所有认证后的网络流量的完全加密。
  - 提供加密完整性检查。

默认模式是 **crc**。

请确定在规划 Red Hat Ceph Storage 集群时考虑集群 CPU 要求，使其包含加密开销。

重要

Ceph 内核客户端目前支持使用 **secure** 模式，比如 Red Hat Enterprise Linux 上的 CephFS 和 **krbd**。Ceph 客户端使用 **librbd** （如 OpenStack Nova、Glance 和 Cinder）支持使用 **secure** 模式。

地址更改

对于同一存储集群中共存的 messenger 协议的两个版本，地址格式已更改：

- 旧地址格式为：***IP_ADDR*:*PORT*/*CLIENT_ID***，例如 **1.2.3.4:5678/91011**。
- 新的格式为：***PROTOCOL_VERSION*:*IP_ADDR*:*PORT*/*CLIENT_ID***，例如 **v2:1.2.3.4:5678/91011**，其中 *PROTOCOL_VERSION* 可以是 **v1** 或 **v2**。

由于 Ceph 守护进程现在绑定到多个端口，因此守护进程会显示多个地址，而不是单个地址。以下是 monitor 映射转储的示例：

```
epoch 1
fsid 50fcf227-be32-4bcb-8b41-34ca8370bd17
last_changed 2021-12-12 11:10:46.700821
created 2021-12-12 11:10:46.700821
min_mon_release 14 (nautilus)
0: [v2:10.0.0.10:3300/0,v1:10.0.0.10:6789/0] mon.a
1: [v2:10.0.0.11:3300/0,v1:10.0.0.11:6789/0] mon.b
2: [v2:10.0.0.12:3300/0,v1:10.0.0.12:6789/0] mon.c
```

此外，**mon_host** 配置选项并使用 **-m** 在命令行中指定地址，支持新的地址格式。

连接阶段

进行加密的连接有四个阶段：

横幅

连接时，客户端和服务器会发送横幅。目前，Ceph 横幅是 **ceph 0 0n**。

身份验证交换

所有数据（发送或接收）包含在连接的持续时间内。服务器将决定身份验证是否已完成，以及连接模式是什么。框架格式已被修复，可以采用三种不同的形式，具体取决于所使用的身份验证标志。

Message Flow Handshake Exchange

对等点互相识别并建立会话。客户端会发送第一个消息，服务器将使用相同的消息进行回复。如果客户端与错误守护进程通信，服务器可以关闭连接。对于新会话，客户端和服务器会继续交换消息。客户端 Cookie 用于识别会话，并可重新连接现有会话。

消息交换

客户端和服务器开始交换消息，直到连接关闭为止。

其它资源

- 如需了解有关启用 **msgr2** 协议的详细信息，请参阅 *Red Hat Ceph Storage 数据安全性和强化指南*。