存储系统

Ceph 分布式存储

介绍

功能概览

部署规划

规划您的部署

部署架构

安全考虑

基础设施要求

架构

第1章：简介

第2章：架构概

第3章：Operat

装概

及概

安装

核心概念

操作指南

实用指南

TopoLVM 本地存储

介绍

安装

前提条件

操作指南

实用指南

Ceph 分布式存储

介绍

规划您的部署

架构

架构

安装

内部模式部署

外部模式部署

创建 Stretch

核心概念

核心概念

Ceph 模块功能

操作指南

存储池管理

创建存储池

删除存储池

查看对象存储池地址

节点特定组件部署

更新组件部署配置

重启存储组件

添加设备/设

添加设备类

添加设备

硬盘状态

监控与告警

监控

告警

实用指南

替换或移除设备

前提条件

约束与限制

操作步骤

参考资料

替换或移除存储节点

前提条件

约束与限制

操作步骤

References

配置用于分布

架构

基础设施要求

操作步骤

后续操作

清理分布式存储

注意事项

操作步骤

灾难恢复

更新优化参数

创建 Ceph 对象存储用户

前提条件

为 Ceph RGW 启用 D3N 缓存

背景

前提条件

操作概览

准备本地文件系统作为缓存

在 CephObjectStore 中启用 D3N 缓存

验证 D3N 配置

验证缓存行为

配置传输加密

概述

限制与前提条件

为新集群启用传输加密

部署后启用传输加密

禁用传输加密

验证

故障排查建议

池配

设置 Ceph C

前提条件

操作步骤

持久卷加密

概述

密钥管理系统（KMS）的访问配置

为持久卷加密创建 StorageClass

使用 tenant ConfigMap 覆盖 Vault 连接详

启用和禁用密钥轮转（可选）

配置 OSD WAL 和 DB 分区

简介

场景

前提条件

限制和约束

规划 WAL 和 DB 容量

操作步骤

配置 Multus

概述

限制和前提条件

在集群创建时配

验证

故障排查建议

MinIO 到 Rook Ceph RGW 数据迁移指南

1. 概述与架构
2. 前提条件与准备
3. 部署配置（Manifests）
4. 执行流程
5. 故障排查与调优建议

介绍

Alauda Build of Rook-Ceph 是平台在集群内提供的一种超融合存储解决方案。基于开源的 Rook + Ceph 存储方案，分布式存储实现自动管理、自动扩容和自动修复能力，满足中小型应用的块存储、文件存储和对象存储需求。

> **NOTE**
>
> 本文档中，分布式存储 指本集群内的 Ceph 存储，外部存储 指本集群外的 Ceph 存储。

目录

功能概览

- 简易部署：提供存储集群的图形化自动部署和管理服务；支持计算与存储的集成部署和解耦部署两种模式。
- 专业运维：具备持久卷快照备份和克隆新卷功能；容量、性能及组件级别的可视化监控；内置告警策略，满足大多数存储运维场景需求。
- 安全可靠：分布式多副本机制保障数据安全与可靠；简洁可靠的自动化管理支持存储资源的在线扩容。

- 卓越性能：提供弹性高性能存储服务；支持混合磁盘设备部署，提升存储系统性能和效率。

部署规划

参见 Planning Your Deployment

规划您的部署

本主题为在 Alauda Container Platform (ACP) 上部署 Ceph 分布式存储提供规划清单。它总结了架构选择、安全选项、基础设施容量、网络约束以及灾难恢复方面的考虑，帮助您在执行实际安装之前确定部署模型。

有关产品背景，请参阅 Introduction 和 Architecture。有关部署流程，请参阅 Install 和 How To 下的文档。

目录

部署架构

ACP 分布式存储基于 Ceph 和 Rook 构建。从整体上看，该平台结合了以下层：

- Ceph 守护进程，例如 MON、MGR、OSD、MDS 和 RGW，用于提供块、文件和对象存储能力
- Rook 和 CSI 组件，用于自动化部署、供给、扩展和生命周期管理
- ACP 平台集成，用于暴露存储池、可观测性和运维入口

在部署之前，请先决定您的环境应使用本地集群中的存储服务，还是使用外部 Ceph 环境中的存储。

内部和外部部署模型

您可以通过以下方式规划 ACP 分布式存储：

部署模式	存储服务运行位置	谁管理存储集群	最适合场景	主要权衡
内部，同驻	Ceph 组件运行在同时承载业务工作负载的同一	ACP 平台团队或集群管理员	早期环境、裸金属集群，或尚未	交付更简单，但应用与存储之间

部署模式	存储服务运行位置	谁管理存储集群	最适合场景	主要权衡
	组 ACP worker 节点上		完全明确存储需求的场景	发生资源竞争的可能性更高
内部，专用节点	Ceph 组件运行在同一 ACP 集群内专用的存储节点或基础设施节点上	ACP 平台团队或集群管理员	存储需求可预测且隔离要求更严格的生产环境	运维隔离性和容量控制更好，但需要预留更多节点并进行容量规划
外部	ACP 从外部 Ceph 环境消费存储类	独立的存储团队、SRE 团队，或现有的外部存储所有者	大规模环境、多个消费集群，或已经运行独立 Ceph 集群的组织	责任边界清晰，但需要更多跨集群网络、认证和依赖管理

内部部署更容易交付和管理，因为存储服务与消费工作负载是在同一 ACP 环境中一起规划的。在内部部署中，第一个设计选择是存储应与业务工作负载共享节点，还是使用专用节点。当需要在存储集群与应用集群之间建立更强的隔离，或者多个业务集群需要共享同一存储后端时，外部部署更合适。

主要的规划决策点如下：

- 当您希望更快交付，并且可以接受存储与应用工作负载共享同一个 worker 池时，选择同驻部署。
- 当存储需求已明确，并且希望获得更清晰的容量控制、故障隔离和维护边界时，选择专用节点部署。
- 当存储已由其他系统管理，或者单个外部集群需要为多个 ACP 集群提供服务时，选择外部部署。

节点角色

规划节点放置时，应区分控制平面节点、基础设施节点和 worker 节点的职责：

- 控制平面节点负责维护集群管理功能，除非部署模型明确支持，否则不应将其视为通用存储节点。
- 当您希望将存储平台组件与业务工作负载隔离时，基础设施节点是合适的选择。

- worker 节点可以在同驻部署中承载存储服务，但这会增加应用与存储守护进程之间的资源竞争。

用于生产环境时，请至少规划三个故障域，以便实现高可用存储服务。尽可能将存储节点分散到不同的机架、可用区或主机组中。

安全考虑

在部署之前，请确认存储设计是否需要传输中加密，并在启用前验证其对运维的影响。

存储类加密

您可以使用外部 Key Management System (KMS) 对 persistent volumes（仅块）进行加密，以存储设备加密密钥。Persistent volume 加密仅适用于 RADOS Block Device (RBD) persistent volumes。请参阅 how to create a storage class with persistent volume encryption。

Alauda Build of Rook-Ceph 支持使用 HashiCorp Vault KMS 和 Thales CipherTrust Manager KMS 进行存储类加密。

传输中加密

ACP 目前支持 Ceph 分布式存储的传输中加密。该功能可保护 Ceph 组件与客户端之间的流量，通常围绕 Ceph `msgr2` 和集群网络模型进行规划。

在启用传输中加密之前，请验证：

- 存储节点和客户端节点上的内核与操作系统支持情况
- 忙碌的存储节点上的预期 CPU 开销
- 目标硬件上的吞吐量和延迟影响

有关实现细节，请参阅 Configure in-transit encryption。

基础设施要求

最低和推荐配置

在创建集群之前，请规划节点数量、存储设备和可用资源。

项目	最低配置	推荐配置
存储节点	3 个节点	3 个或更多节点，分布在不同故障域中
存储设备	每个节点 1 个可用存储设备	每个节点多个专用设备，且类型和大小保持一致
节点分布	3 个可承载 Ceph 服务的节点	3 个故障域，例如机架或可用区
设备使用	系统盘与存储盘分离	Ceph 数据使用专用原始磁盘，并预留未来扩容空间

至少应保证集群拥有三个节点，并且每个节点上有一个可用的存储设备。用于生产环境时，请将集群部署在至少三个故障域中，并预留足够的空闲资源，以应对重平衡、修复和未来增长。

资源容量规划

Ceph 存储服务会持续消耗 CPU、内存和设备容量。请先为存储守护进程规划资源，然后再为恢复、重平衡、升级和后台任务预留额外空间。

作为基线：

- 至少从三个存储节点开始，以构建高可用集群
- 为 MON、MGR、OSD 以及任何已启用的 MDS 或 RGW 服务预留足够的 CPU 和内存
- 为新增池、额外设备和集群恢复事件保留增长空间
- 避免在第一天就规划一个已经接近饱和的集群

如果您的设计使用专用存储节点，资源规划会更可预测。如果存储与业务工作负载一起运行，请预留更多空间，以吸收高峰负载和节点故障期间的资源竞争。

集群总量规划预算

在早期容量评估中，请先基于集群总预算进行规划，而不是仅依赖单个组件数值。下表适合作为在进行工作负载特定调优之前，三节点高可用集群的规划参考：

部署模式	为存储预留的总 **CPU**	为存储预留的总内存	说明
内部，最低基线	24 个逻辑 CPU	72 GiB	当仅满足最低部署目标时，适用于入门级三节点规划基线
内部，标准基线	30 个逻辑 CPU	72 GiB	适合作为通用生产规划和未来扩展的更佳起点
内部，性能导向基线	45 个逻辑 CPU	96 GiB	当从一开始就需要更高吞吐量或更低延迟时适用
外部消费集群	仅按连接性和客户端访问进行容量规划	仅按连接性和客户端访问进行容量规划	存储守护进程运行在 ACP 集群之外，因此 ACP 集群主要需要网络可达性、凭据和客户端侧容量

这些数值应视为集群级规划目标，而不是精确的调度预留。若要估算三节点集群的单节点预算，请将总量平均分配到参与的存储节点上。

以下建议适合作为早期规划参考：

组件	推荐 **CPU**	推荐内存
MON	2 核	3 GiB
MGR	3 核	4 GiB
MDS	3 核	8 GiB
RGW	2 核	4 GiB
OSD	4 核	8 GiB

这些数值仅作为规划参考，而不是硬性的调度保证。实际需求取决于设备数量、启用的服务和工作负载强度。

如何估算集群规模

在进行集群容量规划时，请按以下顺序进行：

1. 选择部署模式：同驻、专用节点或外部。
2. 确定最少节点数量和故障域布局。
3. 决定是否需要块、文件、对象或混合存储服务。
4. 从集群总量规划预算开始。
5. 为额外设备组、恢复、监控和预期增长增加余量。

如果同时需要文件和对象服务，或者集群将同时承载繁重的业务工作负载，则应按高于最低基线的容量进行规划，而不要直接按最低值规划。

Pod 放置

Pod 放置规则会直接影响弹性。请按以下方式规划集群：

- 高可用组件可以分散到不同的故障域中
- 每个故障域都有可访问的存储设备和足够的可分配资源
- 新的设备组或未来扩展仍然可以遵循相同的放置模式

在实践中，这意味着仅仅拥有三个节点还不够。节点还需要以避免单个机架、主机组或可用区成为单点故障的方式进行分布。

存储设备规划

在选择存储设备时，请尽可能标准化设备大小和类型。混合设备会使性能调优和容量规划变得复杂。

请遵循以下原则：

- 为操作系统保留一块系统盘，并为 Ceph 数据使用独立的存储设备
- 优先使用原始磁盘或专用设备，而不是对共享磁盘进行分区
- 将每个节点的设备数量控制在可管理范围内，以便恢复和维护保持可操作性
- 关注可用容量而不是原始容量，因为副本机制会降低有效存储空间

容量规划还应包括告警阈值和扩容策略。请在集群接近满载之前就规划扩容。接近满容量运行会增加重平衡压力，并使恢复更加困难。

有关相关运维指导，请参阅 Managing Storage Pools 和 Adding Devices/Device Classes。

容量规划

在规划集群容量时，请计算可用容量，而不是原始磁盘容量。在采用副本机制的 Ceph 部署中，一部分原始存储始终会被数据保护所消耗。

请遵循以下规划原则：

- 保持可用容量领先于预期业务增长，而不是等到集群几乎满载后才扩容
- 为恢复、重平衡、快照以及数据使用量的临时突增预留额外空间
- 在节点和故障域之间以平衡的方式扩展存储，以避免新增容量造成使用率倾斜
- 在向集群添加新工作负载之前，同时评估当前利用率和预计增长

以下示例可作为三节点集群的早期规划参考：每个节点 1 个设备，并采用 3 副本数据保护策略：

每个节点的设备大小	集群原始容量	采用 **3** 副本时的近似可用容量
0.5 TiB	1.5 TiB	0.5 TiB
2 TiB	6 TiB	2 TiB
4 TiB	12 TiB	4 TiB

这些数值仅为示例。可用容量会随实际数据保护策略而变化，不应将其视为适用于所有集群设计的通用规则。

在 day-2 运维中，应在集群达到告警级别之前复查容量。如果增长可预测，应尽早扩容，而不是等到接近满载或已满载时再处理。

网络要求

Ceph 对网络质量非常敏感。在部署之前，请验证以下内容：

- 集群网络能够为复制和恢复流量提供稳定吞吐
- 各故障域之间的延迟处于所选部署模型支持的范围内
- 存储节点与消费集群之间所需端口已开放
- 任何专用网络设计，例如基于 Multus 的隔离，都已提前确定

如果您计划将存储流量与一般应用流量隔离，请在部署前确认网络接口、路由策略和运维归属。网络隔离可以提升安全性和性能，但也会增加设计复杂度。

IPv6 支持

ACP 分布式存储规划必须遵循平台所选择的集群网络协议栈。

- 在单栈 IPv6 环境中支持 IPv6。
- 双栈规划必须在存储部署之前与 ACP 集群网络设计进行验证。
- 存储节点和客户端节点应使用相同的地址族策略，以避免连通性和服务发现问题。

如果您的环境使用 IPv6，请在安装前确认以下内容：

- ACP 集群网络已经配置为 IPv6 运行
- 所有存储节点都可以通过所需的 IPv6 路由进行通信
- 访问存储端点的监控、告警和外部集成也支持 IPv6

应将 IPv6 视为安装时的架构决策。不要假设已有的 IPv4 导向存储设计可以在之后无需重新验证就直接转换。

灾难恢复规划

ACP 分布式存储可以根据不同的恢复目标进行规划。请选择符合您的恢复点目标 (RPO)、恢复时间目标 (RTO) 和站点拓扑的模型。

Regional-DR

ACP 支持 Regional-DR，用于跨地域或跨站点的灾难恢复场景，在这种场景中，可接受异步复制和少量潜在数据丢失。

在规划 Regional-DR 时，请提前确认以下事项：

- 源集群和目标集群具有兼容的存储和网络设计
- 复制延迟和故障切换预期与业务恢复目标一致
- 受保护的工作负载类型明确，例如块、文件系统或对象数据

有关实现细节，请参阅 Disaster Recovery。

拉伸集群

仅当站点之间的延迟受到严格控制，并且拓扑是专门为该模式设计时，拉伸集群才合适。通常应规划以下内容：

- 两个数据站点和一个仲裁站点
- 跨三个可用区至少五个节点
- 在创建集群之前设置手动且明确的故障域标签
- 每个数据站点拥有足够的节点，以保持存储服务可用性
- 区域间延迟保持在低延迟设计范围内，通常数据站点之间的 RTT 不超过 10 ms

> **WARNING**
>
> 不要将拉伸集群视为长距离、高延迟、多数据中心部署的通用方案。如果站点间延迟无法严格控制，请改用专用灾难恢复架构。

有关 ACP 特定的拉伸集群部署指导，请参阅 Create Stretch Type Cluster。

性能规划

性能应基于工作负载特征进行规划，而不是仅依据原始设备数量。在部署之前，请识别：

- 主要工作负载是块、文件还是对象类型

- 工作负载是延迟敏感、吞吐量敏感，还是容量密集型
- 热数据、备份流量还是分析作业将主导集群

同时还应确认是否需要特殊调优或特定功能设计。例如，对象工作负载可能需要单独规划网关容量，而某些环境可能需要缓存导向或专用集群设计。

下一步

完成规划后，请继续查阅与您所选部署模型相匹配的部署指南：

内部部署

- 对于同驻部署，请参阅 Deploying in Internal Mode。
- 对于拉伸集群部署，请参阅 Create Stretch Type Cluster。
- 对于专用节点部署，请参阅 Configure a Dedicated Cluster for Distributed Storage。

外部部署

- 要从其他集群或外部 Ceph 环境消费存储服务，请参阅 Deploying in External Mode。

相关后续配置

- 要为已部署的存储服务启用加密网络流量，请参阅 Configure in-transit encryption。
- 要在部署后配置灾难恢复，请参阅 Disaster Recovery。

架构

目录

第1章：简介

Alauda Build of Rook-Ceph 是 Alauda Container Platform (ACP) 的云原生存储层。它作为 Kubernetes 原生系统运行，通过 operator、自定义资源和 CSI 驱动提供持久存储服务。

从应用角度来看，平台提供三种主要存储类型：

- 文件存储：为需要多个 Pod 并发读写访问的工作负载提供共享文件系统语义（例如 RWX 场景）。
- 块存储：为延迟敏感或数据库类工作负载提供低级块设备，需专用卷。
- 对象存储：兼容 S3 的对象访问，用于非结构化数据、备份工件和数据平台集成。

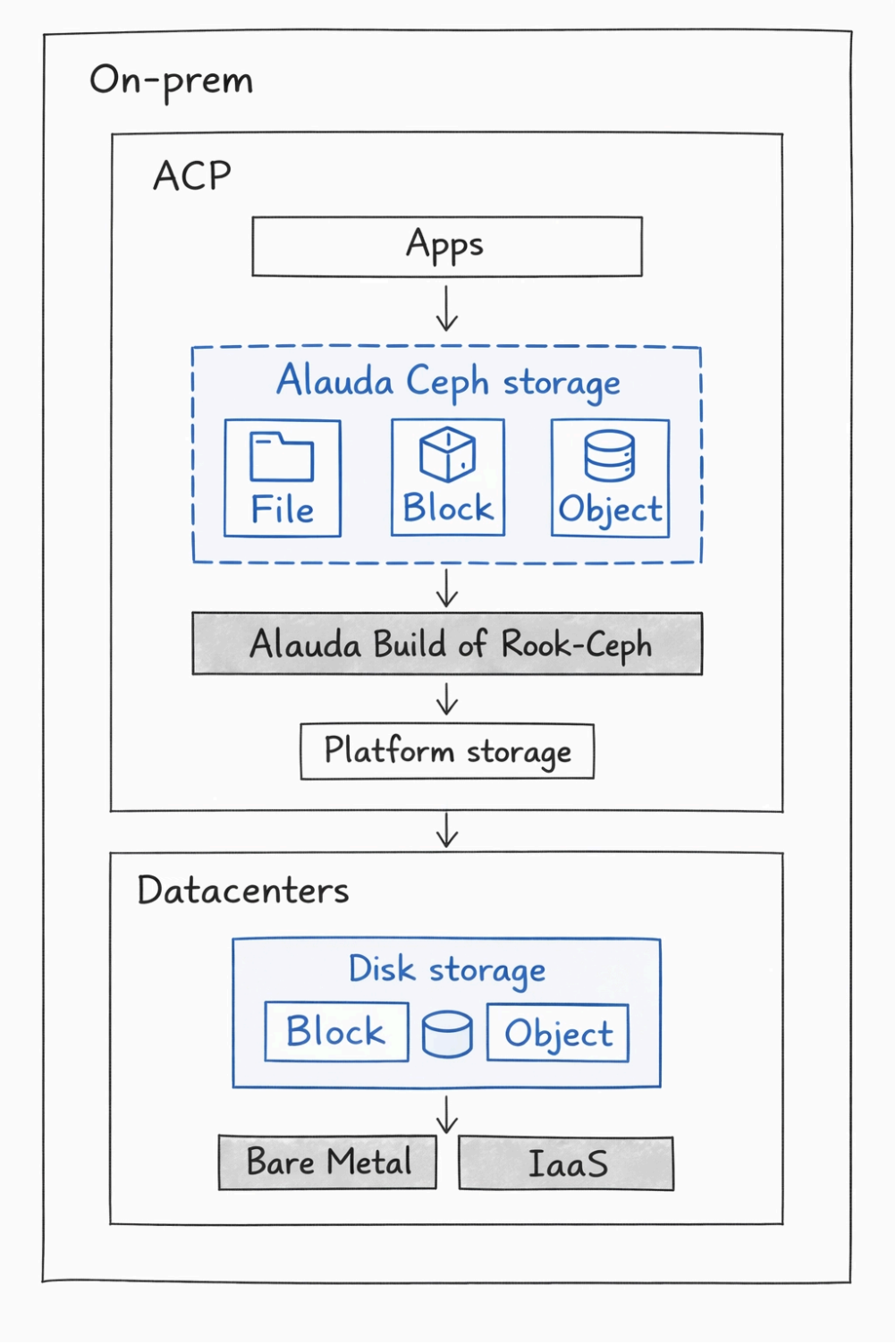
Alauda Build of Rook-Ceph 集成了以下核心开源项目：

- Ceph：分布式存储引擎，提供块、文件和对象能力。
- Rook：用于部署和管理 Ceph 集群的 Kubernetes operator 框架。
- Ceph CSI Operator：管理 Ceph CSI 驱动组件及其生命周期的 operator。
- Ceph CSI：Kubernetes 用于动态配置、挂载/卸载及 Ceph 支持卷生命周期操作的 CSI 实现。

第2章：架构概览

从整体来看，**Alauda Build of Rook-Ceph** 完全运行在 **Alauda Container Platform** 内部。

Alauda Build of Rook-Ceph 架构



Alauda Build of Rook-Ceph 使用故障域配置来确定数据副本如何分布在集群中。故障域代表物理边界，如主机、机架或可用区，数据副本在这些边界内分散以确保高可用性。**Alauda Build of Rook-Ceph** 根据分配给存储节点的标签自动识别这些故障域。

第3章：Operators

Alauda Build of Rook-Ceph 由以下两个 Operator Lifecycle Manager (OLM) operator 包组成，部署的 operators 将管理任务和自定义资源编码化，以便任务和资源特性可以轻松自动

化：

- acp-storage-operator
- rook-ceph-operator

管理员定义集群的期望最终状态，operators 确保集群处于该状态或接近该状态，且管理员干预最小。

Rook-Ceph operator

`rook-ceph-operator` 是 **Alauda Build of Rook-Ceph** 中 Ceph 的 operator。它使 Ceph 存储系统能够作为 Kubernetes 管理的服务在 **Alauda Container Platform** 上运行。

`rook-ceph-operator` 容器自动引导存储集群并监控存储守护进程以维护集群健康。

组件

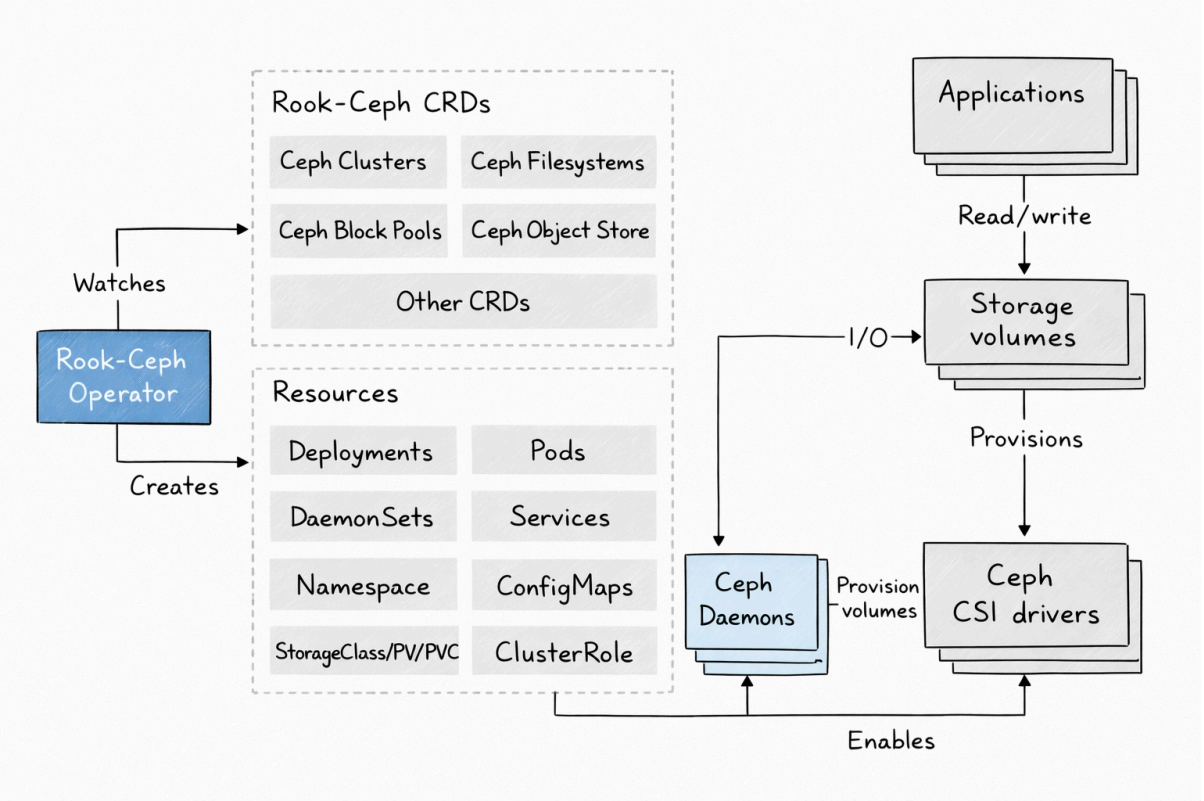
`rook-ceph-operator` 管理以下 Ceph 守护进程组件：

- Mons：Ceph 监视器（`mons`）维护集群元数据和仲裁。
- OSDs：对象存储守护进程（`OSDs`）存储和复制数据。
- Mgr：Ceph 管理器（`mgr`）提供指标和内部管理能力。
- RGW：RADOS 网关（`RGW`）通过兼容 S3 的 API 提供对象存储访问。
- MDS：元数据服务器（`MDS`）为 CephFS 共享文件系统提供元数据服务。

设计图

设计图展示了 `rook-ceph-operator` 如何将 Ceph 集成到 ACP 中。

应用可以通过挂载块设备和共享文件系统，或使用对象存储 API 来使用此存储层。



职责

`rook-ceph-operator` 负责引导和监控存储集群，具体职责包括：

- 自动配置存储组件。
- 启动、监控和管理 RADOS 集群的 Ceph 监视器 Pod 和 Ceph OSD 守护进程。
- 初始化用于管理以下内容的 Pod 和工件：
  - 池的自定义资源
  - 对象存储（S3）
  - 文件系统
- 监控 Ceph 监视器和 OSD 守护进程，保持存储可用和健康。
- 部署和管理监视器的位置，并随着集群规模变化更新监视器配置。
- 监听通过 API 服务请求的期望状态变更并应用这些变更。
- 初始化 Ceph-CSI 驱动。
- 自动配置 Ceph-CSI 将 Ceph 存储挂载到应用 Pod。

`rook-ceph-operator` 镜像包含生命周期管理所需的工具，不修改数据路径，也不暴露所有低级 Ceph 配置；高级 Ceph 构造如 placement groups 和 CRUSH maps 保持抽象，以简化运维模型。

资源

operator 会为其在 `rook-ceph` 命名空间创建的资源添加所有者引用。卸载时，这些引用确保相关资源也被删除，包括 `ConfigMaps`、`Secrets`、`Services`、`Deployments` 和 `DaemonSets`。

operator 监听并调和以下自定义资源：

- `CephCluster`
- `CephObjectStore`
- `CephFilesystem`
- `CephBlockPool`

生命周期

operator 管理以下 Ceph 和 CSI Pod 的生命周期：

- Rook operator：
  - 一个 operator Pod。
- RBD CSI 驱动：
  - 两个由一个 Deployment 管理的 provisioner Pod。
  - 每个节点一个由 DaemonSet 管理的插件 Pod。
- CephFS CSI 驱动：
  - 两个由一个 Deployment 管理的 provisioner Pod。
  - 每个节点一个由 DaemonSet 管理的插件 Pod。
- 监视器（`mons`）：
  - 三个监视器 Pod，每个有独立的 Deployment。
  - 在跨区域集群中，五个监视器 Pod：一个在仲裁区，两个在每个数据区。
- 管理器（`mgr`）：
  - 标准部署中两个管理器 Pod。

- 对象存储守护进程（`OSDs`）：
  - 初始创建至少三个 OSD。
  - 随集群容量扩展增加更多 OSD。
- 元数据服务器（`MDS`）：
  - 两个元数据服务器 Pod。
- RADOS 网关（`RGW`）：
  - 至少两个网关 Pod。

ACP storage operator

`acp-storage-operator` 是 ACP 特定的 operator，负责调和 `CephCluster` 资源并管理围绕操作、监控和故障排除的内置 ACP 存储集成。

> **NOTE**
>
> 仅适用于内部模式部署。

组件

`acp-storage-operator` 管理以下内置集成组件：

- 用于 `CephCluster` 的 Ceph 调和控制器。
- 两个 `MutatingWebhookConfiguration` 条目：
  - 一个用于 `CephCluster`
  - 一个用于 `CephFilesystem`

设计图

`acp-storage-operator` 作为 ACP 中的控制平面 operator 运行，扩展 Ceph 集群生命周期，提供 ACP 原生的监控和运维资源。

职责

`acp-storage-operator` 执行以下职责：

- 监听并调和 `CephCluster` 资源。
- 通过变更 webhook 在创建时变更 `CephCluster` 和 `CephFilesystem` 对象。
- 在变更过程中初始化 ACP 特定默认值，包括 `resources`、`placement` 和相关运行时设置。
- 创建内置的 `.mgr` 池以支持 ACP 所需的 Ceph 管理器功能。
- 创建并维护 ACP 内置的 `AlertRule` 资源。
- 创建并维护 ACP 内置的 `MonitorDashboard` 资源。
- 自动启用并维护 `rook-ceph-tools` Pod。

资源

`acp-storage-operator` 为每个调和的 `CephCluster` 创建并管理以下资源：

- 内置 Ceph `.mgr` 池。
- ACP 内置的 `AlertRule` 对象。
- ACP 内置的 `MonitorDashboard` 对象。
- `rook-ceph-tools` Pod 及其相关运行时资源。

生命周期

`acp-storage-operator` 在内部模式下遵循以下生命周期：

1. 引导阶段：operator 部署启动，注册 `CephCluster` 和 `CephFilesystem` 的控制器及变更准入 webhook。
2. 准入阶段：在创建请求时，webhook 注入 ACP 默认值（如 `resources` 和 `placement`），然后对象被持久化。
3. 调和阶段：`CephCluster` 可用后，operator 持续调和 ACP 内置资源，包括 `.mgr` 池、`AlertRule`、`MonitorDashboard` 和 `rook-ceph-tools` Pod。
4. 清理阶段：当 `CephCluster` 被删除时，operator 移除其为该集群创建的所有 ACP 管理资源。

第4章：安装概览

本章描述内部模式和外部模式的安装流程。

Operator 安装

- OLM 安装 `acp-storage-operator` 以支持 ACP 内置存储集成。
- OLM 从其包中安装 rook-ceph-operator 并创建 operator 部署。
- 部署包含调和 `CephCluster` 和 Ceph 相关资源所需的控制组件。
- `acp-storage-operator` 注册 `CephCluster` 和 `CephFilesystem` 对象的变更准入 webhook。
- 启动后，operator 开始监听 CRD 并准备存储消费所需的 CSI 组件。

Ceph 集群创建概览

- 管理员创建 `CephCluster` 自定义资源。
- operator 使用该声明协调 Ceph 相关资源和 CSI 服务。
- 在内部模式下，`acp-storage-operator` 还调和围绕同一 `CephCluster` 的 ACP 内置存储资源。
- 具体调和路径根据部署模式不同而异：内部模式或外部模式。

CephCluster 创建

内部模式

在内部模式下，存储守护进程运行在创建 `CephCluster` 的 ACP 集群内。

1. operator 接收存储命名空间中的 `CephCluster`。
2. `acp-storage-operator` 变更 webhook 初始化 `CephCluster` 和 `CephFilesystem` 资源的 ACP 默认值（如 `resources` 和 `placement`）。
3. `rook-ceph-operator` 引导监视器仲裁并创建管理守护进程。
4. OSD 准备作业发现配置的设备，然后创建 OSD 部署。

5. 部署 Ceph CSI provisioner 和节点插件 Pod，用于 RBD 和 CephFS。
6. `acp-storage-operator` 调和 ACP 内置资源，包括 `.mgr` 池、内置 `AlertRule`、内置 `MonitorDashboard` 和 `rook-ceph-tools` Pod。
7. `StorageClass` 对象可用于通过 PVC 工作流动态配置。
8. 通过 `CephCluster` 状态持续报告集群健康和就绪状态。

外部模式

在外部模式下，Ceph 数据平面运行在外部提供者集群，而 ACP 托管消费者侧控制集成。

1. operator 在消费者 ACP 集群中安装并调和 CSI 组件。
2. 导入外部 Ceph 连接数据（如监视器端点、集群标识符和客户端密钥）。
3. 以外部模式（`external: true`）调和 `CephCluster`，表示远程 Ceph 后端。
4. 消费者集群中不创建本地监视器或 OSD 守护进程。
5. RBD 和 CephFS CSI 驱动使用导入的连接信息从外部 Ceph 集群配置和挂载卷。
6. 从消费者侧组件和外部连通性检查调和状态和健康。

第5章：升级概览

Alauda Build of Rook-Ceph 通过 `Subscription`、`InstallPlan` 和 `ClusterServiceVersion` (CSV) 资源遵循 OLM 管理的升级工作流。

当当前频道出现新版本包时，OLM 创建 `InstallPlan`，执行取决于自动或手动批准策略。

`InstallPlan` 批准后，OLM 执行 CSV 调和：

- 安装新 CSV。
- 更新 operator Deployment 到新镜像并重启。
- 成功过渡检查后替换旧 CSV。

随后 operator 级调和完成升级：

- operators 验证并重新应用用户面 CR 的期望状态。
- 检查 Ceph 和 CSI 资源的版本一致配置。

- 纠正漂移，直到集群状态报告健康且收敛。

> **WARNING**
>
> 不要依赖手动编辑生成的 CSV 内容作为持久自定义机制。后续升级过程中，OLM 调和可能覆盖这些临时更改。

安装

内部模式部署

内部模式（UI 中的标准集群类型）是 Ceph 存储最常见的部署方式。它在当前业务集群内创建本地 Ceph 集群，并将数据副本分布在不同主机的硬盘上，以保证单主机故障时的可用性。

目录

前提条件

准备软件包

- 下载与您的平台架构对应的 **Alauda Container Platform Storage Essentials** 安装包。
- 通过上传软件包机制上传 **Alauda Container Platform Storage Essentials** 安装包。
- 下载与您的平台架构对应的 **Alauda Build of Rook-Ceph** 安装包。
- 通过上传软件包机制上传 **Alauda Build of Rook-Ceph** 安装包。

准备基础设施

- 存储集群至少需要 3 个节点。
- 每个节点必须至少有 1 块空白硬盘或 1 个未格式化的硬盘分区可用。
- 建议可用硬盘容量大于 50 G。
- 如果您使用的是以 Containerd 作为运行时组件的附加 Kubernetes 集群，请确保集群所有节点的 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值配置为 `1048576`，以保证分布式存储部署成功。配置说明请参考 修改 Containerd 配置信息。注意：从早于 v3.10.2 版本升级到当前版本时，如果需要在自定义 Kubernetes 集群（以 Containerd 作为运行时组件）上部署 Ceph 分布式存储，也必须将集群所有节点的 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值设置为 `1048576`。

注意事项

内部模式部署与外部模式部署互斥，集群只能选择一种部署方式。

操作步骤

1 部署 Alauda Container Platform Storage Essentials

1. 登录，进入 **Administrator** 页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。

3. 找到 **Alauda Container Platform Storage Essentials**，点击 **Install**，进入 **Install Alauda Container Platform Storage Essentials** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `stable`。
Installation Mode	`Cluster`：集群内所有命名空间共用一个 Operator 实例进行创建和管理，资源占用较低。
Installation Place	选择 `Recommended`，命名空间仅支持 **acp-storage**。
Upgrade Strategy	`Manual`：Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

2 （可选）部署 Alauda Build of LocalStorage

当使用“选择设备”方式向 Ceph 集群添加存储设备时，需部署 `Alauda Build of LocalStorage Operator`。该 Operator 负责自动发现 Kubernetes 集群内所有节点的硬盘设备，收集完整的设备信息，简化存储集成流程。

1. 登录，进入 **Administrator** 页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Build of LocalStorage**，点击 **Install**，进入 **Install Alauda Build of LocalStorage** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `stable`。
Installation Mode	`Cluster`：集群内所有命名空间共用一个 Operator 实例进行创建和管理，资源占用较低。

参数	推荐配置
Installation Place	选择 `Recommended`，命名空间仅支持 **acp-storage**。
Upgrade Strategy	`Manual`：Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

3 部署 Operator

1. 进入 **Administrator**。
2. 在左侧边栏点击 **Storage Management** > **Distributed Storage**。
3. 点击 **Configure Now**。
4. 在 **Deploy Operator** 向导页面，点击右下角 **Deploy Operator** 按钮。
   - 页面自动跳转到下一步时，表示 Operator 部署成功。
   - 若部署失败，请参考界面提示的 **Clean Up Deployed Information and Retry**，清理后重试部署；若需返回分布式存储选择页面，点击 **Application Store**，先卸载已部署的 **rook-operator** 资源，再卸载 **rook-operator**。

4 创建集群

1. 在 **Create Cluster** 向导页面，配置相关参数，点击右下角 **Create Cluster** 按钮。

参数	说明
Cluster Type	选择 **Standard**（内部模式）。
Device Class Type	设备类是硬盘的分组，您可以根据存储需求自定义设备类，将不同性能的硬盘分配存储不同内容。 • 默认设备类：平台会自动分类集群节点中的硬盘类型，例如创建名为 `hdd`、`ssd`、`nvme` 的设备类。 • 自定义设备类：自定义节点中特定组合硬盘的设备类名称，支持添加多个设备类。同一块硬盘只能属于一个设备类。

参数	说明
Device Class - Name	设备类名称。选择 自定义设备类 时，设备类名称不能使用以下名称：`hdd`、`ssd`、`nvme`。

Device Class - Storage Devices	向设备类添加存储设备时，可选择“选择设备”或“输入设备”方式：
	• 选择设备：
	从可用存储设备中选择。设备可用条件包括：
	  • 设备类型为 disk 或 mpath
	  • 未检测到文件系统（fsType 为空）
	  • 容量大于 10 GiB
	rbd、nbd、dm-* 等设备不会显示在可选设备列表中。
	注意：需先部署 Alauda Build of LocalStorage Operator。
	• 输入设备：
	手动输入节点下空白设备名称，如 `sda`。
	注意：建议使用裸盘作为存储设备，避免使用磁盘上的单个分区，以获得更佳性能和管理效果。

Snapshot	启用后支持创建 PVC 快照，并使用快照配置新 PVC，实现业务数据快速备份与恢复。 创建存储时未启用快照，也可在存储集群详情页的 **Operations** 中按需启用。 注意：使用前请确保已为当前集群部署卷快照插件。
Monitoring Alarm	启用后提供开箱即用的监控指标采集和告警能力，详见监控与告警。 注意：若此时未启用，需自行寻找存储监控和告警方案，例如在运维中心手动配置监控面板和告警策略。

2. 点击 **Advanced Configuration** 进行高级组件配置。

参数	说明
Network Configuration	• **Host Network**：存储集群使用主机网络，需在优化参数栏填写相关网络优化参数，如配置 `public` 和 `cluster` 子网。若留空，则使用默认主机子网。 注意：使用主机网络可能存在安全风险，因数据通过主机端口明文传输。请联系平台支持团队获取加密传输方案。 • **Container Network**：存储集群使用容器网络；可在网络管理中创建子网并分配给 `rook-ceph` 命名空间。若留空，则使用默认子网。 注意： 不支持 IPv6。 使用容器网络时，存储仅在集群内可访问。 Ceph CSI Pod 故障或重启可能导致服务中断。
Optimization Parameters	支持填写 Ceph 配置文件格式的参数，系统会根据填写内容覆盖默认参数。 注意：首次填写或修改初始化参数后，请点击初始化参数，需初始化成功后方可创建集群。
Component Fixed-point Deployment	可将组件部署到指定节点，至少需三个节点以保证最小可用性。支持固定点部署的组件包括 MON、MGR、MDS、RGW。

- 页面自动跳转到下一步时，表示 Ceph 集群部署成功。
- 若创建失败，可点击清理 **Created Information or Retry**，自动清理资源后重新创建集群，或根据文档 分布式存储服务资源清理 手动清理资源。

5 创建存储池

1. 在 **Create Storage Pool** 向导页面，配置相关参数，点击右下角 **Create Storage Pool** 按钮。

参数	说明
Storage Type	• 文件存储：提供安全、可靠、可扩展的共享文件存储服务，适用于文件共享、数据备份等。 • 块存储：提供高 IOPS 和低延迟存储服务，适用于数据库、虚拟化等。 • 对象存储：提供标准 S3 接口存储服务，适用于大数据、备份归档、云存储等。
Replica Count	副本数越多，冗余度和数据安全性越高，但存储利用率降低。通常设置为 3，满足大多数需求。
Device Class	对同类型设备或同一业务逻辑的磁盘进行统一分类，从前一步添加的设备类中选择。 • 选择设备类后，数据将存储在所选设备类中。 • 未选择设备类时，数据将在存储池所有设备中随机存储。

对象存储还需配置以下参数：

参数	说明
Region	指定存储池所在的地域。
Gateway Type	默认为 S3，且不可修改。
Internal Port	指定集群内访问端口。
External Access	启用/禁用外部访问，将创建/销毁 Nodeport 类型 Service。
Instance Count	对象存储的资源实例数量。

- 页面自动跳转到下一步时，表示存储池部署成功。
- 若部署失败，请根据界面提示检查核心组件，然后点击 **Clean Up Created Information and Retry** 重新创建存储池。

2. 点击 **Create Storage Pool**，在 **Details** 标签页查看已创建存储池信息。

相关操作

创建 **Stretch** 类型集群

详情请参考 创建 Stretch 类型集群。

清理分布式存储

详情请参考 清理分布式存储。

外部模式部署

外部模式用于消费已有的 Ceph 存储服务，而不是在当前业务集群中创建新的本地 Ceph 集群。您可以连接平台上其他业务集群中的分布式存储，或集成外部 Ceph 环境。

目录

注意事项

内部模式部署和外部模式部署互斥。一个集群只能选择一种部署方式。

前提条件

准备软件包

- 下载对应平台架构的 **Alauda Container Platform Storage Essentials** 安装包。
- 通过上传软件包机制上传 **Alauda Container Platform Storage Essentials** 安装包。
- 下载对应平台架构的 **Alauda Build of Rook-Ceph** 安装包。
- 通过上传软件包机制上传 **Alauda Build of Rook-Ceph** 安装包。

准备存储

选择以下之一：

- 其他业务集群已部署分布式存储，并创建了存储池。请记录存储池名称以备后续集成使用。
- 平台外部已创建外部 Ceph 存储（版本 ≥ 14.2.3）并创建了存储池。请记录存储池名称以备后续集成使用。

开放端口

目标 **IP**	目标端口	源 **IP**	源端口
Ceph 节点 IP	3300, 6789, 6800-7300, 7480	业务集群所有节点 IP	任意

获取认证信息（外部 **Ceph**）

如果准备的存储是外部 Ceph 存储，必须使用以下命令获取认证信息。

参数	获取方式
FSID	`ceph fsid`
MON 组件信息	`ceph mon dump` 必须为 {name= IP} 格式，例如 *a=192.168.100.100:6789* 。

参数	获取方式
Admin Key	`ceph auth get-key client.admin`
存储池	• 文件存储：使用 `ceph fs ls` 命令获取 `name` 值。 • 块存储：`ceph osd dump` | `grep "application rbd"` | `awk '{print $3}'`
数据存储池	（仅文件存储需要）使用 `ceph fs ls` 命令获取 `data pools` 值。

操作步骤

注意：以下步骤以外部 **Ceph** 存储为例，消费其他业务集群分布式存储的流程类似。

1 部署 Alauda Container Platform Storage Essentials

1. 登录，进入 **Administrator** 页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Storage Essentials**，点击 **Install**，进入 **Install Alauda Container Platform Storage Essentials** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `stable`。
Installation Mode	`Cluster`：集群内所有命名空间共用一个 Operator 实例进行创建和管理，资源占用较低。
Installation Place	选择 `Recommended`，命名空间仅支持 **acp-storage**。

参数	推荐配置
Upgrade Strategy	`Manual`：Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

2 配置外部模式

1. 在左侧导航栏点击 **Storage Management** > **Distributed Storage**。
2. 点击 **Access Storage**。
3. 在 **Access Configuration** 向导页面，选择 **External Mode**，然后选择 **External Ceph**。

参数	描述
Snapshot	启用后支持创建 PVC 快照，并使用快照配置新的 PVC，实现业务数据的快速备份与恢复。 如果访问存储时未启用快照，后续仍可在存储集群详情页的 **Operations** 部分按需启用。 注意：请确保当前集群已部署卷快照插件后再使用。
Network Configuration	• **Host Network**：本集群计算组件将使用 主机网络 访问 存储集群。 • **Container Network**：本集群计算组件将使用 容器网络 访问 存储集群。可在网络管理中创建子网，并将子网分配给 `rook-ceph` 命名空间。若留空，则使用默认子网。
Other Parameters	填写前提条件中获取的外部 Ceph 认证参数。

4. 在 **Create Storage Class** 向导页面，完成配置后点击 **Access**。

参数	描述
Type	根据上述创建的存储池类型，默认对应的存储类为：

参数	描述
	• 文件存储：CephFS 文件存储 • 块存储：CephRBD 块存储
Reclaim Policy	持久卷的回收策略。 • Delete：删除持久卷声明时，绑定的持久卷也会被删除。 • Retain：即使删除持久卷声明，绑定的持久卷仍会保留。
Project Allocation	可使用该类型存储的项目。 如果当前没有需要该类型存储的项目，可以暂不分配，后续再更新。

5. 等待约 1-5 分钟，完成集成成功。

后续操作

- 创建存储类：CephFS 文件存储、CephRBD 块存储
- 使用上述存储类创建持久卷声明的开发者，可扩展使用卷快照和弹性伸缩功能。

注意：如果需要维护外部存储的存储池、存储设备配置等，必须在存储集群的管理平台进行操作。

创建 Stretch 类型集群

Stretch 集群可以跨越两个地理位置不同的站点，提供存储基础设施的灾难恢复能力。当两个可用区中的一个完全不可用时，Ceph 仍能保持可用性。

> ⚠️ 重要提示：**Alpha** 功能免责声明
> 本文档描述的功能目前处于**Alpha** 阶段，仅供早期技术验证和评估。作为功能提供方，我们不对其稳定性、可靠性或数据完整性提供任何保证。配置和使用该功能即表示您已知晓并接受以下技术限制：
>
> - 非生产环境使用： 该功能缺乏全面的系统级验证，在生产环境中部署存在较高的流程失败或数据损坏风险。
> - 无 **SLA** 保证： 该功能不在标准服务等级协议（SLA）范围内，不保证技术支持响应时间，也不提供紧急热修复。
> - 破坏性变更及废弃风险： API 端点、配置模式及核心处理逻辑未来版本可能发生不兼容变更，且该功能可能被完全废弃且不另行通知。
> - 无平滑升级路径： 不提供版本间升级脚本或数据迁移工具，升级通常需要彻底清理现有资源并重新部署，任何状态或数据丢失由用户自行承担。

目录

术语

术语	说明
Quorum 可用区	通常位于不承载主业务负载的独立可用区，专注于维护集群一致性，主要用于在主数据中心故障或网络分区时进行仲裁决策。
数据可用区	Ceph 集群中实际存储和处理数据的主要区域，承担业务负载和数据存储任务，与仲裁区共同构成完整的高可用存储系统。

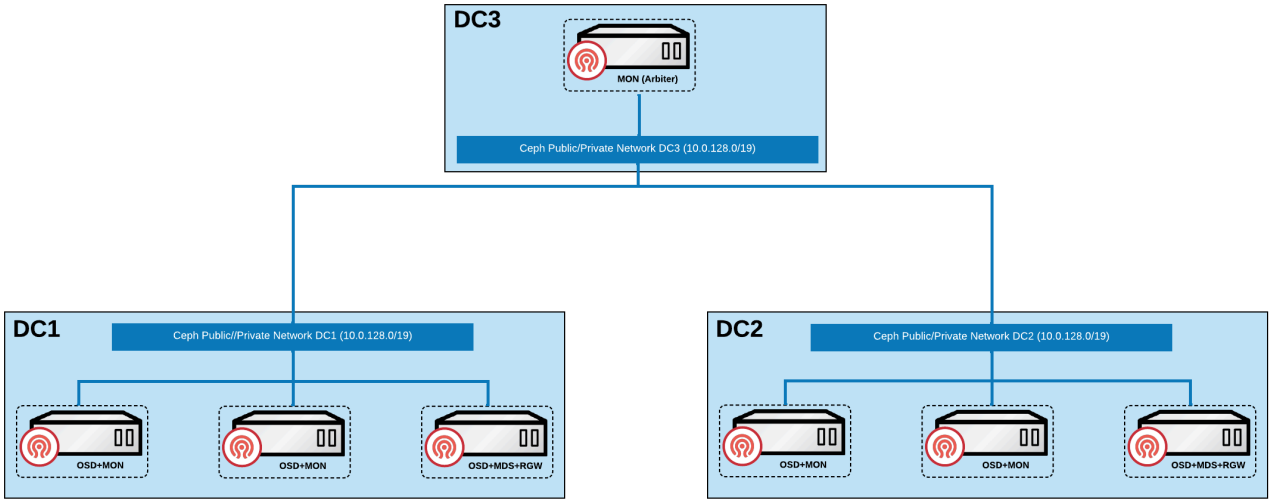
典型部署方案

以下内容提供了 Stretch 集群的典型部署方案，并附带组件说明及灾难恢复原理。

组件说明

节点需分布在三个可用区内，包括两个数据可用区和一个仲裁可用区。

- 两个数据可用区均需完整部署所有核心 Ceph 组件（MON、OSD、MGR、MDS、RGW），且每个数据可用区必须配置两个 MON 实例以实现高可用。当同一数据可用区内的两个 MON 实例均不可用时，系统判定该可用区处于故障状态。
- 仲裁可用区仅需部署一个 MON 实例，作为仲裁决策节点。



灾难恢复说明

- 当某个数据可用区完全故障时，Ceph 集群会自动进入降级状态并触发告警通知。系统会将存储池的最小副本数（min_size）从默认的 2 调整为 1。由于另一个数据可用区仍保持双副本，集群可继续提供服务。当故障数据可用区恢复后，系统会自动执行数据同步并恢复至健康状态；若故障无法修复，建议更换为新的数据可用区。
- 当两个数据可用区之间的网络连接中断，但它们仍能正常连接仲裁可用区时，仲裁可用区会根据预设策略对两个数据可用区进行仲裁，选择状态较好的数据可用区继续作为主数据区提供服务。

约束与限制

- 存储池限制：不支持纠删码存储池，仅支持副本机制进行数据保护。
- 设备分类限制：不支持设备分类功能，无法基于设备特性进行存储分层。
- 区域部署限制：仅支持两个数据可用区，不能超过两个数据可用区。
- 数据均衡要求：两个数据可用区的 OSD 权重必须严格保持一致，确保数据分布均衡。
- 存储介质要求：仅允许全闪存（All-Flash）OSD 配置，最大限度缩短连接恢复后的恢复时间，减少潜在数据丢失风险。
- 网络延迟要求：两个数据可用区之间的 RTT（往返时延）不得超过 10ms，仲裁可用区必须满足 ETCD 规范的延迟要求，以保证仲裁机制的可靠性。

前提条件

请提前将集群中全部或部分节点划分为三个可用区，具体如下：

- 确保至少有 5 个节点分布在一个仲裁可用区和两个数据可用区中。其中，仲裁可用区至少包含一个节点，该节点可以是虚拟机或云主机。
- 确保三个可用区中至少有一个可用区包含 Master 节点（控制节点）。
- 确保至少有 4 个计算节点均匀分布在两个数据可用区中，每个数据可用区至少配置 2 个计算节点。
- 尽量保证两个数据可用区的节点数量和磁盘配置保持一致。

操作步骤

1 标记节点

1. 进入 管理员。
2. 在左侧导航栏点击 集群管理 > 集群。
3. 点击对应集群名称，进入集群概览页面。
4. 切换至 节点 标签页。
5. 根据前提条件规划，为这些节点添加 `topology.kubernetes.io/zone=<zone>` 标签，将其划分到指定的可用区。此处将 <zone> 替换为可用区名称。

2 创建存储服务

本文档仅描述与内部模式部署不同的参数，其他参数请参考内部模式部署。

创建集群

参数	说明
集群类型	选择 **Stretch**。
仲裁可用区	选择仲裁可用区名称。
数据可用区	选择可用区名称并选择节点。

创建存储池

参数	说明
副本数	默认值为 4。
实例数	当存储类型为 对象存储 时，为保证可用性，实例最小数为 2，最大数为 5。

相关操作

内部模式部署

详情请参见内部模式部署。

清理分布式存储

详情请参见清理分布式存储。

核心概念

核心概念

目录

Rook Operator

Rook operator 是一个简单的容器，包含启动和监控存储集群所需的所有内容。operator 会启动并监控 Ceph monitor pods、提供 RADOS 存储的 Ceph OSD 守护进程，以及启动和管理其他 Ceph 守护进程。operator 通过初始化运行服务所需的 pods 和其他资源来管理 pools、object stores（S3/Swift）和 filesystems 的 CRDs。

operator 会监控存储守护进程以确保集群健康。当需要时会启动或故障转移 Ceph mons，并根据集群的扩展或缩减进行其他调整。operator 还会监视 Ceph 自定义资源（CRs）中指定的期望状态变化并应用这些变化。

Rook 会自动配置 Ceph-CSI 驱动，将存储挂载到您的 pods。rook/ceph 镜像包含管理集群所需的所有工具。

Ceph CSI

Ceph CSI 插件实现了 CSI 支持的容器编排器（CO）与 Ceph 集群之间的接口。它们支持动态创建 Ceph 卷并将其挂载到工作负载。

Ceph 模块功能

模块	功能
MON	监视器（MON）是 Ceph 集群中最重要的组件。它管理 Ceph 集群并维护整个集群的状态。MON 确保集群的相关组件能够同时同步。它作为集群的领导者，负责收集、更新和发布集群信息。
MGR	管理器（MGR）是一个监控系统，提供数据收集、存储、分析（包括告警）和可视化功能。它使某些集群参数可供外部系统使用。
OSD	对象存储守护进程（OSD）存储实际的用户数据。每个 OSD 通常绑定到一个物理驱动器。OSD 处理来自客户端的读写请求。
MDS	Ceph 元数据服务器（MDS）跟踪文件层次结构并存储仅用于 CephFS 的元数据。RBD 和 RGW 不需要元数据。MDS 不直接为客户端提供数据服务。
RGW	RADOS 网关（RGW）是一个 Ceph 对象网关，提供兼容 S3 和 Swift 的 RESTful API。RGW 还支持多租户和 OpenStack 身份服务（Keystone）。
RADOS	可靠自治分布式对象存储（RADOS）是 Ceph 存储集群的核心。Ceph 中的所有数据都以对象形式存储在 RADOS 中，无论其数据类型如何。RADOS 层通过数据复制、故障检测与恢复以及跨集群节点的数据恢复来确保数据一致性和可靠性。
LIBRADOS	Librados 是简化访问 RADOS 的方法。目前支持 PHP、Ruby、Java、Python、C 和 C++ 等编程语言。它提供了 Ceph 存储集群的本地接口 RADOS，是其他服务（如 RADOS 块设备（RBD）和 RADOS 网关（RGW））的基础组件。此外，它为 Ceph 文件系统（CephFS）提供了可移植操作系统接口（POSIX）。Librados API 可用于直接访问 RADOS，使开发者能够创建自己的接口以访问 Ceph 集群存储。
RBD	RADOS 块设备（RBD）是 Ceph 块设备，为外部系统提供块存储。它可以像驱动器一样映射、格式化并挂载到服务器。
CephFS	CephFS 提供了一个 POSIX 兼容的分布式文件系统，支持任意大小。它依赖 Ceph MDS 来跟踪文件层次结构，即元数据。

操作指南

存储池管理

- 创建存储池
- 删除存储池
- 查看对象存储池地址

节点特定组件部署

- 更新组件部署配置
- 重启存储组件

添加设备/设

- 添加设备类
- 添加设备
- 硬盘状态

监控与告警

- 监控
- 告警

存储池管理

存储池是指用于存储数据的逻辑分区。单个存储集群支持同时使用不同类型的存储池，如文件存储和块存储，以满足各种业务需求。

目录

创建存储池

除了在分布式存储配置过程中创建的存储池外，还可以创建其他类型的存储池。

提示：同一存储集群内，仅允许存在一个文件存储池和一个对象存储池，而块存储池最多可创建八个。

操作步骤

1. 进入 管理员。

2. 在左侧导航栏中，点击 存储管理 > 分布式存储。

3. 在 集群信息 标签页，向下滚动至 存储池 区域，点击 创建存储池。

4. 按照以下说明配置相关参数。

参数	说明
存储类型	选择当前未部署的存储类型。 - 文件存储：提供安全、可靠且可扩展的共享文件存储服务，适用于文件共享、数据备份等场景。 - 块存储：提供高IOPS和低延迟的存储服务，适用于数据库、虚拟化等场景。 - 对象存储：提供标准的S3接口存储服务，适用于大数据、备份归档、云存储服务等。
副本数	• 当集群类型为标准版时：副本数越高，冗余和数据安全性越强，但存储利用率会降低。通常设置为3即可满足大多数需求。 • 当集群类型为增强版时：副本数默认为4，且不可修改。

设备类型	• 当集群类型为标准版时：选择已添加到创建的存储池中的设备类型。
	  • 选择设备类型后，数据将存储在所选设备类型中。
	  • 若未选择设备类型，数据将随机存储在存储池内的所有设备中。
	• 当集群类型为增强版时：不支持添加设备类型。

如果是对象存储类型，还可以配置以下参数：

参数	说明
地域	指定存储池所在的地域。
网关类型	默认为 S3，且不可修改。

参数	说明
内网端口	指定集群内网访问端口。
外网访问	启用/禁用外网访问将创建/销毁 NodePort 类型的 Service。
实例数	对象存储的资源实例数量。

5. 点击 创建。

删除存储池

当某种类型的存储不再需要时，可在解除与存储类的关联后删除该存储池。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。
3. 在 集群信息 标签页，向下滚动至 存储池 区域，点击要删除的存储池旁的⋮> 删除。
4. 阅读提示信息并输入存储池名称。
5. 点击 删除。

查看对象存储池地址

创建对象存储池后，可以查看该存储池的内外网访问地址。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。
3. 在 集群信息 标签页，向下滚动至 存储池 区域，点击对象存储池旁的⋮并选择 查看地址。

节点特定组件部署

创建分布式存储后，您仍然可以查看和修改组件的部署位置，方便存储扩容和维护。

目录

更新组件部署配置

注意事项

- 更新配置将触发系统自动重建组件实例，可能会影响服务对存储系统的访问，建议在非高峰时段进行更新。
- 当集群类型为 **Extend** 时，不支持组件的固定部署功能。

操作步骤

1. 进入 管理员。

2. 在左侧导航栏点击 存储管理 > 分布式存储。
3. 在 存储组件 标签页下，点击 组件部署配置。
4. 根据业务需求启用/禁用 固定部署 开关，将组件部署到指定节点。节点数量不得少于三个，以保证最低可用性。支持固定部署配置的组件包括 MON、MGR、MDS、RGW。
5. 点击 更新，组件将开始调度到指定节点。

重启存储组件

当您删除已部署的存储组件时，系统将根据当前组件部署策略自动重新调度并重新部署组件到节点。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 分布式存储。
3. 在 存储组件 标签页下，点击组件名称旁的⋮> 删除。

添加设备/设备类

目录

添加设备类

统一集群节点中同类型设备或具有相同业务逻辑的硬盘分类，根据存储需求自定义设备类，并将不同存储内容分配到不同类型的存储盘。

注意事项

集群类型为 **Extend** 时，不支持添加设备类。

操作步骤

1. 进入 管理员。

2. 在左侧导航栏中，点击 存储管理 > 分布式存储。
3. 点击 设备类 标签页。
4. 点击 添加设备类，并根据以下说明配置相关参数。

参数	说明
名称	设备类的名称。设备类名称不能使用以下名称：`hdd`、`ssd`、`nvme`。

存储设备	向设备类添加存储设备时，可选择“选择设备”或“输入设备”两种方式：
	• 选择设备：
	从可用存储设备中选择。设备被视为可用需满足以下条件：
	  • 设备类型为 disk 或 mpath
	  • 未检测到文件系统（fsType 为空）
	  • 容量大于 10 GiB
	rbd、nbd 和 dm-* 等设备不会显示在可选设备列表中。
	注意：需事先部署 Alauda Build 版本的 LocalStorage Operator。
	• 输入设备：
	手动输入节点下的空白设备名称，如 `sda`。
	注意：为获得最佳性能和管理效果，强烈建议使用裸盘作为存储设备，而非使用磁盘上的单个分区。

添加设备

将可用硬盘映射为存储设备以供使用和管理。

注意：硬盘添加为存储设备后，不支持通过界面进行更新或删除操作。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。
3. 点击 设备类 标签页。
4. 在设备类右侧，点击 添加设备，并根据以下说明配置相关参数。

参数	说明
指定磁盘	向设备类添加存储设备时，可选择“选择设备”或“输入设备”两种方式： • 选择设备： 从可用存储设备中选择。设备被视为可用需满足以下条件： • 设备类型为 disk 或 mpath • 未检测到文件系统（fsType 为空） • 容量大于 10 GiB rbd、nbd 和 dm-* 等设备不会显示在可选设备列表中。 注意：需事先部署 Alauda Build 版本的 LocalStorage Operator。 • 输入设备： 手动输入节点下的空白设备名称，如 `sda`。 注意：为获得最佳性能和管理效果，强烈建议使用裸盘作为存储设备，而非使用磁盘上的单个分区。

5. 点击 添加。

硬盘状态

- 正常：存储设备对应状态为 IN+UP。
- 异常：存储设备对应状态为 IN+DOWN。

- 离线：存储设备对应状态为 OUT+UP。
- 故障：存储设备对应状态为 OUT+DOWN。

监控与告警

分布式存储提供开箱即用的监控指标采集和告警通知功能。启用监控与告警功能后，您可以对存储集群、存储性能和存储组件等方面进行监控和告警，并支持配置通知策略。

直观呈现的监控数据可为运维巡检或性能调优提供决策支持，完善的告警与通知机制有助于确保存储系统的稳定运行。

提示：如果创建分布式存储时未启用监控与告警功能，则需要寻找其他方案实现存储监控与告警。例如，在运维中心手动配置监控面板和告警策略。

目录

监控

平台自动采集分布式存储的常用监控指标，如读写性能、CPU 和内存使用率。在 存储管理 > 分布式存储 的 监控 标签页中，您可以查看这些指标的实时监控数据。

存储概览

监控存储的健康状态、物理容量使用情况以及活跃的 OSD/MON 组件数量。存储状态异常时，可查看告警原因。

性能监控

从集群、存储池和 OSD 三个维度监控读写带宽和读写 IOPS，此外还可专门监控 OSD 的读写延迟。

组件监控

监控 MON、OSD 等组件的 CPU 使用率和内存使用情况。

告警

平台已启用一套默认的告警策略。当资源异常或监控数据达到告警状态时，会自动触发告警。预设策略满足组件及集群状态告警、设备容量告警、用户数据告警等常见运维需求。

配置通知

为及时接收告警，建议您在运维中心配置通知策略：通过邮件、短信等方式将告警信息发送给相关人员，提醒其采取必要措施解决问题或防范故障。点击 告警配置 可跳转至运维中心完成操作，详见 Create Alert Strategies。

处理告警

- 当监控到存储集群处于 `Warning` 状态时，表示已触发告警，相关异常可能导致故障。请及时查看 实时告警 中的详情，依据原因定位并排查故障。
- 当监控到存储集群处于 `Failure` 状态时，表示存储集群无法正常运行。请立即定位问题并进行故障排查。

下表说明了预设策略使用的告警级别含义，可作为您建立告警处理原则的参考。

告警级别	含义
灾难	告警规则对应的资源发生故障，导致平台服务中断、数据丢失，影响严重。
严重	告警规则对应的资源存在已知问题，可能导致平台功能故障，影响服务的正常运行。
警告	告警规则对应的资源存在运行风险，若不及时处理，可能影响服务的正常运行。

故障复盘

告警历史 记录所有已触发且无需再处理的告警。在利用告警历史进行故障复盘时，为有效达到总结经验的目的，您可能需要回答以下问题。

- 事件发生时的具体异常情况是什么。
- 是否存在某个告警在告警列表中反复出现的规律，是否可以在下次发生前进行预防。
- 时间线上是否显示某段时间内告警激增；是否由不可抗力或操作失误引起，是否需要调整运维方案。

实用指南

替换或移除设备

替换或移除存储节点

配置用于分布式存储的专用集群

配置用于分布式存储的专用集群

架构

基础设施要求

操作步骤

后续操作

清理分布式存储

清理分布式存储

注意事项

操作步骤

灾难恢复

文件存储灾难恢复

术语

备份配置

故障切换

块存储灾难恢复

术语

备份配置

计划迁移

灾难恢复

对象存储灾

术语

前提条件

架构

操作步骤

故障切换

故障恢复

更新优化参数

更新优化参数

操作步骤

创建 **Ceph** 对象存储用户

创建 **Ceph** 对象存储用户

前提条件

操作步骤

设置存储池配额

设置存储池配额

前提条件

为文件存储池设置池配额

为块存储池设置池配额

为对象存储池设置池配额

通过 Ceph 终端验证池配额

为 **Ceph RGW** 启用 **D3N** 缓存

为 Ceph RGW 启用 D3N 缓存

配置传输加密

配置传输加密

设置 Ceph OSD 满容量阈值

设置 Ceph OSD 满容量阈值

持久卷加密

持久卷加密

配置 **OSD WAL** 和 **DB** 分区

配置 **OSD WAL** 和 **DB** 分区

配置 Multus 多 NIC 网络

MinIO 到 Rook Ceph RGW 数据迁移指南

替换或移除设备

本文档介绍如何从由 Container Platform 管理的 Rook-Ceph 集群中移除存储设备（磁盘）。根据剩余 OSD 是否有足够容量承载被移除磁盘上的数据，您可能需要先添加替换磁盘。

目录

前提条件

- 所有集群组件均正常运行。

- 存储集群未使用“添加所有空磁盘”选项创建。通过运行以下命令验证，输出必须显示 `useAllDevices: false`。

```
kubectl get cephcluster -n rook-ceph ceph-cluster -o yaml | grep useAll
Devices
```

- 适用于平台版本 3.8 及以上。

约束与限制

- 在数据重新平衡期间，集群性能可能会暂时下降。除非必要，避免同时操作多个磁盘。
- 如果集群处于 `HEALTH_ERR` 状态，且原因非被移除磁盘导致，请勿继续操作。此时继续操作可能进一步影响数据可靠性。
- 如果被移除的磁盘是某个设备类的最后一块磁盘，则该设备类将不复存在。任何依赖该设备类的存储池或策略将受到影响。操作前请确保没有存储池仅绑定该设备类。

操作步骤

1 检查集群状态和容量

1. 验证集群整体健康状态。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

2. 确认待移除磁盘对应的 OSD ID 及使用情况。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd d
f
```

记录目标 OSD 的 **USE** 值。然后确认剩余所有 OSD（不含目标）**AVAIL** 总和大于目标 OSD 的 **USE** 值，确保剩余 OSD 有足够空闲空间承载移除后的数据。
若剩余容量不足，请继续下一步先添加替换磁盘；否则跳至缩减 Rook Operator。

2 添加替换磁盘（如有需要）

若剩余 OSD 空间不足，需先添加替换磁盘再移除旧磁盘。此步骤需确保 Rook operator 正在运行。

1. 进入 **Container Platform**。
2. 在左侧导航栏点击 存储管理 > 分布式存储 > 设备类。
3. 点击 添加设备，选择替换磁盘所在节点，选择新磁盘，并分配到与被移除磁盘相同的设备类。
4. 等待新 OSD 创建完成并完成数据重新平衡。可通过以下命令监控进度：

   ```
   kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
   ```

   等待输出显示 `HEALTH_OK`，且无 misplaced 或 recovering 的 PG。

3 缩减 Rook Operator

缩减 Rook operator，防止其在移除过程中干扰（例如中途重新创建已删除的 OSD 部署）。

```
kubectl -n rook-ceph scale deploy rook-ceph-operator --replicas=0
```

4 标记 OSD 为 Out 并等待数据迁移

1. 若 rook-ceph-tools pod 未运行，启用它。

   ```
   kubectl -n rook-ceph scale deploy rook-ceph-tools --replicas=1
   ```

2. 进入 tools pod。

   ```
   kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
   ```

3. 将 OSD 标记为 `out`，指示 Ceph 将该 OSD 上所有数据迁移到剩余 OSD。

```
ceph osd out osd.<id>
```

4. 监控重新平衡进度，直到集群恢复 `HEALTH_OK`，且无 misplaced 或 recovering 的 PG。

```
ceph -s
```

数据迁移未完成前请勿继续操作。 过早移除 OSD 会导致数据丢失。

5 移除 OSD

1. 编辑 CephCluster 资源，删除对应磁盘条目。

```
kubectl edit cephcluster -n rook-ceph ceph-cluster
```

找到 `spec.storage.nodes` 下的磁盘条目并删除，保存退出。

2. 删除 OSD 部署。

```
kubectl -n rook-ceph delete deploy rook-ceph-osd-<osd-id>
```

3. 进入 tools pod，永久从集群中移除该 OSD。将 `<id>` 替换为 OSD ID。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

在 tools pod 内执行：

```
ceph osd purge osd.<id> --yes-i-really-mean-it
```

6 清理磁盘

若磁盘仍物理连接在节点上，需清除其元数据，防止 Rook 误识别。请在磁盘所在节点上执行以下命令，将 `/dev/vdb` 替换为实际设备路径。

```
# 移除 Ceph 留下的任何 device-mapper 条目
dmsetup remove /dev/mapper/ceph--<vg-name>    # 如存在，替换为实际映射名称

# 清除分区表
sgdisk --zap-all /dev/vdb

# 清零前 100 MB，清除残留 Ceph 元数据
dd if=/dev/zero of=/dev/vdb bs=1M count=100 oflag=direct,dsync
```

7 恢复 Rook Operator

集群健康后，恢复 Rook operator。

```
kubectl -n rook-ceph scale deploy rook-ceph-operator --replicas=1
```

8 验证集群健康

1. 确认已移除的 OSD 不再出现在集群中。

   ```
   kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd tree
   ```

2. 验证集群已恢复健康状态。

   ```
   kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
   ```

   输出应显示 `HEALTH_OK`，且所有 PG 处于 `active+clean` 状态。

参考资料

- Rook Ceph OSD Management ↗

替换或移除存储节点

本文档介绍如何从由 Container Platform 管理的 Rook-Ceph 集群中移除存储节点。根据剩余 OSD 是否有足够容量承载被移除节点上的数据，您可能需要先添加一个替换节点。

目录

前提条件

- 除了故障节点（如适用）外，所有集群组件均正常运行。
- 开始操作前，记录旧节点拥有的磁盘数量及每个磁盘所属的设备类别。

约束与限制

- 在三节点 Ceph 集群中，丢失一个节点已降低冗余度。请尽快完成操作步骤，以最小化风险窗口。
- 数据重新平衡期间，集群 I/O 性能可能会暂时下降。
- 如果集群处于 `HEALTH_ERR` 状态，且原因不是被移除节点，切勿继续操作。此时继续操作可能进一步影响数据的可靠性。

操作步骤

1 检查集群状态和容量

1. 确认待移除节点上所有运行的 OSD ID 及其磁盘使用情况。

```
kubectl -n rook-ceph get pod -l app=rook-ceph-osd -o wide | grep <
old-node-name>
```

2. 验证集群整体健康状态。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

3. 查看所有 OSD 的容量信息。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd d
f
```

将待移除节点上所有 OSD 的 **USE** 值相加，再确认剩余节点上所有 OSD 的 **AVAIL** 总和是否大于该值。这样可以确保剩余 OSD 有足够的空闲空间承载移除节点上的数据。如果剩余容量不足，请继续下一步先添加替换节点。否则跳至调整组件部署配置。

2 添加替换节点（如有需要）

如果剩余 OSD 空闲容量不足，需先添加替换节点再移除旧节点。

1. 进入 **Container Platform**。

2. 使用平台的节点管理功能，将替换机器添加为新的集群节点。
3. 节点加入集群后，将其添加为存储节点。导航至 **Storage Management** > **Distributed Storage** > **Device Classes**。
4. 点击 **Add Device**，选择新节点并选择对应磁盘。如果旧节点有多个磁盘且分属不同设备类别，请对每个磁盘/设备类别组合重复此步骤，直至所有磁盘添加完成。
5. 等待新 OSD 激活并完成数据重新平衡。可通过以下命令监控进度：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

等待集群恢复到 `HEALTH_OK` 状态，且无 **misplaced** 或 **recovering PG** 后再继续操作。

3 调整组件部署配置

Rook 管理的 Ceph 守护进程（MON、MGR、MDS）可能调度在旧节点上。需将旧节点从组件调度中排除，使 operator 将它们重新调度到其他节点。

1. 在 **Container Platform** 中，导航至 **Storage Management** > **Distributed Storage** > **Storage Components** > **Component Deployment Configuration**。
2. 启用节点绑定，仅选择应保留在集群中的节点（排除待移除节点）。
3. 等待所有 MON、MGR 和 MDS Pod 在剩余节点上运行后再继续。

```
kubectl -n rook-ceph get pod -o wide
```

4 标记所有 OSD 为 Out 并等待数据迁移

1. 如果 rook-ceph-tools Pod 未运行，启用它。

```
kubectl -n rook-ceph scale deploy rook-ceph-tools --replicas=1
```

2. 进入 tools Pod。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

3. 将旧节点上的每个 OSD 标记为 `out`，指示 Ceph 将数据从这些 OSD 迁移到剩余 OSD。

```
ceph osd out osd.<id>
```

对节点上每个 OSD ID 重复此操作。

4. 监控重新平衡进度，直到集群恢复到 `HEALTH_OK`，且无 misplaced 或 recovering PG。

```
ceph -s
```

数据迁移完成前请勿继续操作。提前移除 **OSD** 会导致数据丢失。

5 移除旧节点的 OSD

1. 编辑 CephCluster 资源，删除旧节点条目。

```
kubectl edit cephcluster -n rook-ceph ceph-cluster
```

找到 `spec.storage.nodes` 下的旧节点，删除整个节点条目，保存并退出。

2. 删除旧节点上每个 OSD 的部署。

```
kubectl -n rook-ceph delete deploy rook-ceph-osd-<osd-id>
```

对每个 OSD ID 重复此操作。

3. 进入 tools Pod，永久从集群中移除每个 OSD。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

在 tools Pod 内，对每个 OSD 执行：

```
ceph osd purge osd.<id> --yes-i-really-mean-it
```

6 验证集群健康状态

1. 确认已移除的 OSD 不再出现在集群中。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd tree
```

2. 验证集群已恢复健康状态。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

输出应显示 `HEALTH_OK`，且所有 PG 处于 `active+clean` 状态。

References

- Rook Ceph OSD Management ↗

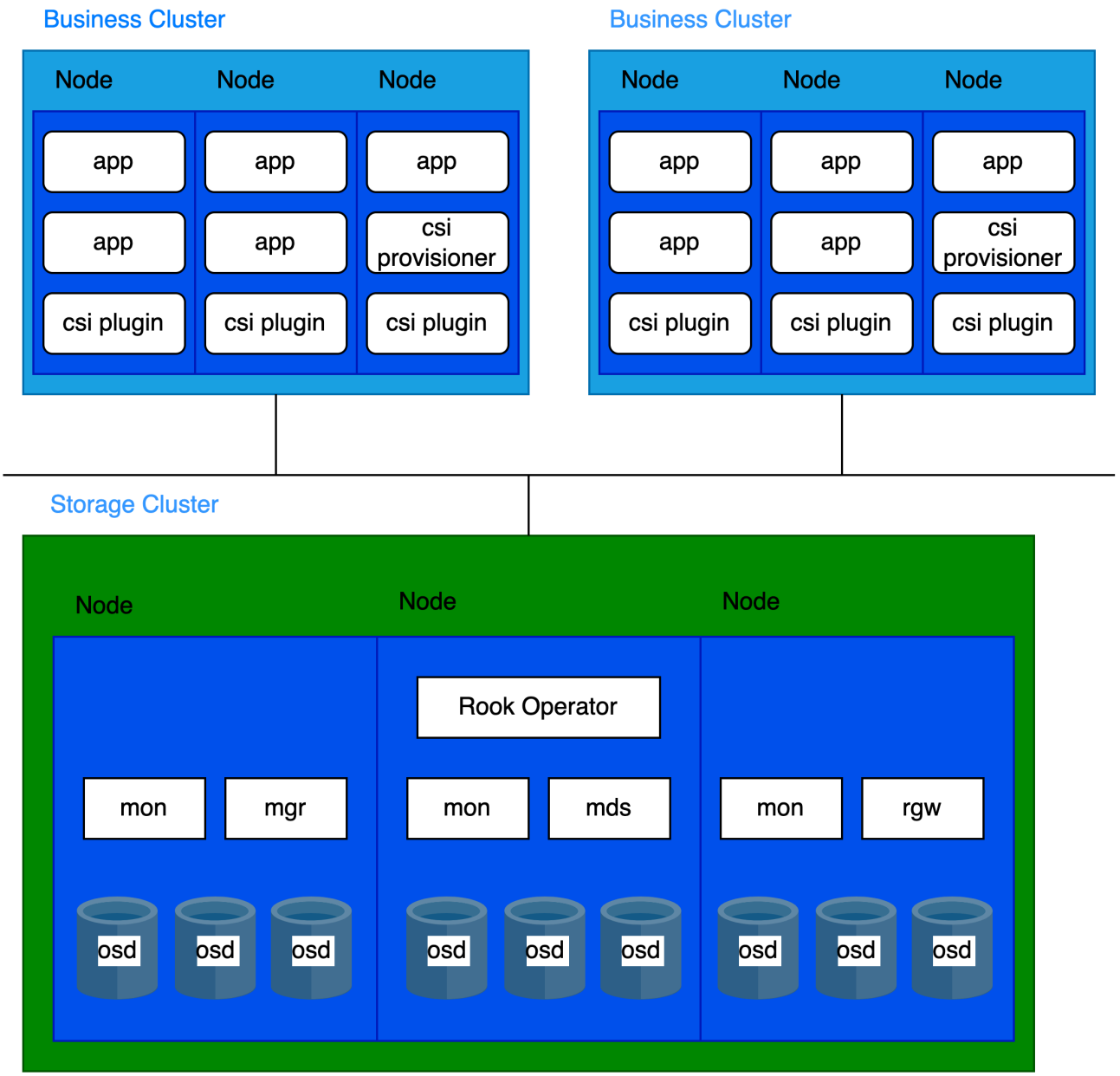
配置用于分布式存储的专用集群

专用集群部署是指使用独立集群部署平台的分布式存储，平台内的其他业务集群通过集成接入并使用其提供的存储服务。
为确保平台分布式存储的性能和稳定性，专用存储集群中仅部署平台核心组件和分布式存储组件，避免与其他业务工作负载共址。这种隔离式部署方式是平台分布式存储的推荐最佳实践。

目录

架构

存储-计算分离架构



基础设施要求

平台要求

支持 3.18 及更高版本。

集群要求

建议使用裸金属集群作为专用存储集群。

资源要求

有关分布式存储部署的组件，请参阅 核心概念。

每个组件对 CPU 和内存的要求不同，推荐配置如下：

进程	CPU	内存
MON	2c	3Gi
MGR	3c	4Gi
MDS	3c	8Gi
RGW	2c	4Gi
OSD	4c	8Gi

一个集群通常运行：

- 3 个 MON
- 2 个 MGR
- 多个 OSD
- 2 个 MDS（如果使用 CephFS）
- 2 个 RGW（如果使用 CephObjectStorage）

根据组件分布，建议的单节点资源如下：

CPU	内存
16c + (4c * 每节点 OSD 数)	20Gi + (8Gi * 每节点 OSD 数)

存储设备要求

建议每个节点部署 12 个或更少的存储设备。这有助于限制节点故障后的恢复时间。

存储设备类型要求

建议使用企业级 SSD，单盘容量不超过 10TiB，并确保所有磁盘的容量和类型完全一致。

容量规划

部署前，必须根据具体业务需求规划存储容量。默认情况下，分布式存储系统采用 3 副本冗余策略，因此可用容量按所有存储设备的总原始容量除以 3 计算。

以 30(N) 个节点为例（副本数 = 3），可用容量场景如下：

存储设备大小**(D)**	每节点存储设备数**(M)**	总容量**(D*M*N)**	可用容量**(D*M*N/3)**
0.5 TiB	3	45 TiB	15 TiB
2 TiB	6	360 TiB	120 TiB
4 TiB	9	1080 TiB	360 TiB

容量监控与扩容

1. 主动进行容量规划
   始终确保可用存储容量大于消耗量。如果存储完全耗尽，恢复需要手动干预，无法仅通过删除或迁移数据来解决。
2. 容量告警
   集群会在两个阈值触发告警：

   - **80%** 使用率（“接近满”）：主动释放空间或横向扩展集群。

- **95%** 使用率（“已满”）：存储已完全耗尽，标准命令无法释放空间。请立即联系平台支持。

请始终及时处理告警并定期监控存储使用情况，以避免服务中断。

3. 扩容建议

- 避免：向现有节点添加存储设备。
- 推荐：通过新增存储节点进行横向扩展。
- 要求：新节点必须使用与现有节点在容量、类型和数量上完全一致的存储设备。

网络要求

分布式存储必须使用 **HostNetwork**。

网络隔离

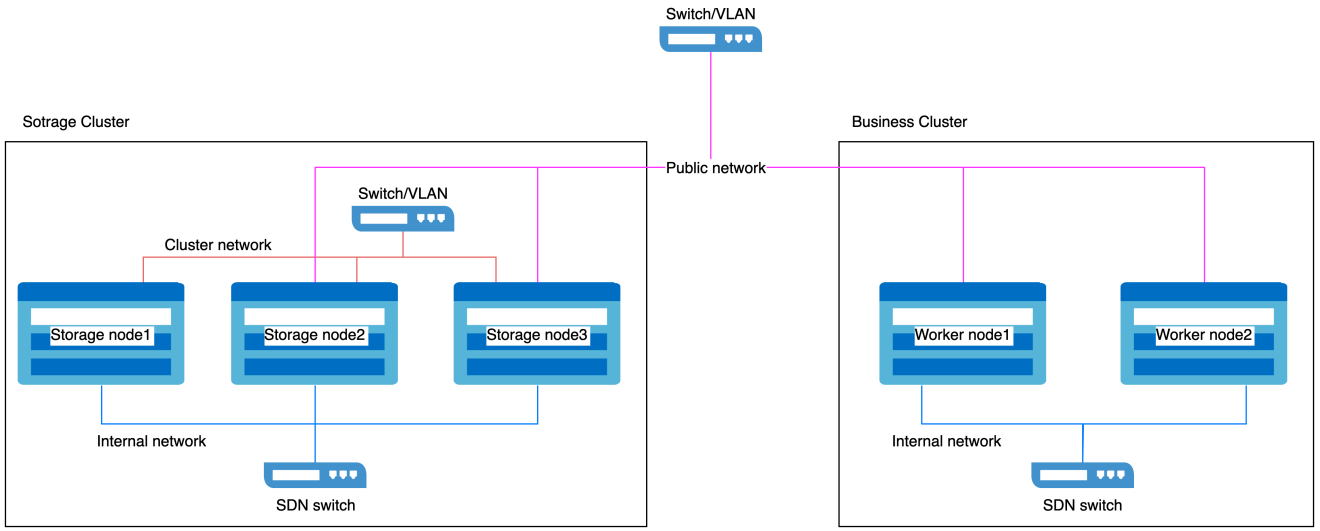
网络分为两类：

- 公网：用于客户端与存储组件之间的交互（例如 I/O 请求）。
- 集群网络：专用于副本之间的数据复制和数据重新平衡（例如恢复）。

为确保服务质量和性能稳定性：

1. 对于专用存储集群：
   每台主机保留两个网络接口：
   - 公网：用于客户端与组件通信。
   - 集群网络：用于内部复制和重平衡流量。
2. 对于业务集群：
   每台主机保留一个网络接口，用于访问存储公网。

网络隔离配置示例



网卡速率要求

1. 存储节点

   - 公网和集群网络都需要 10GbE 或更高的网卡。

2. 业务集群节点

   - 用于访问存储公网的网卡必须为 10GbE 或更高。

操作步骤

1 部署 **Operator**

1. 进入管理员。
2. 在左侧边栏中，单击 存储管理 > 分布式存储。
3. 单击 立即创建。
4. 在 部署 **Operator** 向导页面中，单击右下角的 部署 **Operator** 按钮。

   - 当页面自动进入下一步时，表示 Operator 已成功部署。
   - 如果部署失败，请按照界面提示 清理已部署信息并重试，重新部署 Operator；如果需要返回分布式存储选择页面，请单击 原生应用商店，先卸载已部署的 **rook-operator** 中的资源，然后再卸载 **rook-operator**。

2 创建 **ceph** 集群

在存储集群的控制节点上执行命令。

▶ 点击查看

对于直接创建的内部 CephCluster，如果省略该字段，平台默认将 `spec.disruptionManagement.managePodBudgets` 设为 `true`。 仅在选择退出 Rook 管理的 PDB 保护时，才显式将其设置为 `false`。 平台不会设置 `osdMaintenanceTimeout`；将使用已安装 Rook 版本的 默认值。

参数：

- **public network cidr**：存储公网的 CIDR（例如 `- 10.0.1.0/24`）。
- **cluster network cidr**：存储集群网络的 CIDR（例如 `- 10.0.2.0/24`）。
- **storage devices**：指定分布式存储将使用的存储设备。
  示例格式：

```
nodes:
- name: storage-node-01
  devices:
  - name: /dev/disk/by-id/wwn-0x5000cca01dd27d60
  useAllDevices: false
- name: storage-node-02
  devices:
  - name: sdb
  - name: sdc
  useAllDevices: false
- name: storage-node-03
  devices:
  - name: sdb
  - name: sdc
  useAllDevices: false
```

> **提示**
>
> 使用磁盘的 World Wide Name（WWN）进行稳定命名，避免依赖类似 `sdb` 这类可能在重启后发生变化的易变设备路径。

3 创建存储池

共有三种存储池类型。请根据业务需求选择并创建相应的类型。

创建文件池

在存储集群的控制节点上执行命令。

▶ 点击查看

创建块池

在存储集群的控制节点上执行命令。

▶ 点击查看

创建对象池

在存储集群的控制节点上执行命令。

▶ 点击查看

后续操作

当其他集群需要使用分布式存储服务时，请参考以下指南。
在外部模式下部署

清理分布式存储

如果需要删除 rook-ceph 集群并重新部署新的集群，应按照本文档顺序清理分布式存储服务相关资源。

目录

注意事项

在清理 rook-ceph 之前，确保所有使用 Ceph 存储的 PVC 和 PV 资源已被删除。

操作步骤

1 删除 VolumeSnapshotClasses

1. 删除 VolumeSnapshotClasses。

```
kubectl delete VolumeSnapshotClass csi-cephfs-snapshotclass csi-rbd-snapshotclass
```

2. 验证 VolumeSnapshotClasses 是否已清理。

```
kubectl get VolumeSnapshotClass | grep csi-cephfs-snapshotclass
kubectl get VolumeSnapshotClass | grep csi-rbd-snapshotclass
```

当这些命令无输出时，表示清理完成。

2 删除 StorageClasses

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 存储类。
3. 点击⋮> 删除，删除所有使用 Ceph 存储方案的 StorageClasses。

3 删除存储池

此步骤应在上一步完成后执行。

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 分布式存储。
3. 在 存储池区域，点击⋮> 删除，逐个删除所有存储池。当存储池区域显示 无存储池 时，表示存储池已成功删除。
4. （可选）如果集群模式为 **Extended**，还需在集群的主节点执行以下命令删除创建的内置存储池。

```
kubectl -n rook-ceph delete cephblockpool -l cpaas.io/builtin=true
```

响应：

```
cephblockpool.ceph.rook.io "builtin-mgr" deleted
```

4 删除 ceph-cluster

此步骤应在上一步完成后执行。

1. 更新 ceph-cluster 并启用清理策略。

```
kubectl -n rook-ceph patch cephcluster ceph-cluster --type merge -
p '{"spec":{"cleanupPolicy":{"confirmation":"yes-really-destroy-da
ta"}}}'
```

2. 删除 ceph-cluster。

```
kubectl delete cephcluster ceph-cluster -n rook-ceph
```

3. 删除执行清理的 jobs。

```
kubectl delete jobs --all -n rook-ceph
```

4. 验证 ceph-cluster 清理是否完成。

```
kubectl get cephcluster  -n rook-ceph | grep ceph-cluster
```

当该命令无输出时，表示清理完成。

5 删除 rook-operator

此步骤应在上一步完成后执行。

1. 删除 rook-operator。

```
kubectl -n rook-ceph delete subscriptions.operators.coreos.com roo
k-operator
```

2. 验证 rook-operator 清理是否完成。

```
kubectl get subscriptions.operators.coreos.com -n rook-ceph | grep rook-operator
```

当该命令无输出时，表示清理完成。

3. 验证所有 ConfigMaps 是否已清理。

```
kubectl get configmap -n rook-ceph
```

当该命令无输出时，表示清理完成。如有输出结果，执行以下命令清理，替换 `<configmap>` 为实际输出。

```
kubectl -n rook-ceph patch configmap <configmap> --type merge -p '{"metadata":{"finalizers": []}}'
```

4. 验证所有 Secrets 是否已清理。

```
kubectl get secret -n rook-ceph
```

当该命令无输出时，表示清理完成。如有输出结果，执行以下命令清理，替换 `<secret>` 为实际输出。

```
kubectl -n rook-ceph patch secrets <secret> --type merge -p '{"metadata":{"finalizers": []}}'
```

5. 验证 rook-ceph 清理是否完成。

```
kubectl get all -n rook-ceph
```

当该命令无输出时，表示清理完成。

6 执行清理脚本

完成上述步骤后，表示 Kubernetes 和 Ceph 相关资源已清理。接下来需要清理宿主机上 rook-ceph 的残留。

清理脚本

清理脚本 clean-rook.sh 内容如下：

▶ 点击查看

注意事项

清理脚本依赖 sgdisk 命令，请确保执行清理脚本前已安装。

- Ubuntu 安装命令：`sudo apt install gdisk`
- RedHat 或 CentOS 安装命令：`sudo yum install gdisk`

操作步骤

1. 在业务集群中部署分布式存储的每台机器上执行清理脚本 clean-rook.sh。

   ```
   sh clean-rook.sh /dev/[device_name]
   ```

   示例：`sh clean-rook.sh /dev/vdb`
   执行时会提示确认是否真的清理该设备，确认后输入 yes 开始清理。
2. 使用 `lsblk -f` 命令查看分区信息。当该命令输出中 `FSTYPE` 列为空时，表示清理完成。

灾难恢复

文件存储灾难恢复

CephFS Mirror 是 Ceph 文件系统的一个功能，用于在不同的 Ceph 集群之间实现异步数据复制，从而提供跨集群灾难恢复。其核心功能是以主备模式同步数据，确保当主集群发生故障时，备集群可以快速接管服务。

WARNING

- CephFS Mirror 基于快照执行增量同步，默认快照间隔为每小时一次（可配置）。主备集群之间的差异数据通常为一个快照周期内写入的数据量。
- CephFS Mirror 仅提供底层存储数据的备份，无法处理 Kubernetes 资源的备份。请结合平台的 **Backup and Restore** 功能，一并备份 PVC 和 PV 资源。

目录

术语

术语	说明
Primary Cluster	当前提供存储服务的集群。
Secondary Cluster	用于备份的集群。

备份配置

前提条件

- 准备两个适合部署 Alauda Build of Rook-Ceph 的集群，即 Primary 集群和 Secondary 集群，并确保两个集群之间的网络互通。
- 两个集群使用的平台版本（v3.12 及以上）必须一致。
- 在 Primary 集群和 Secondary 集群中均创建分布式存储服务
- 在 Primary 集群和 Secondary 集群中创建同名的文件存储池。

操作步骤

1 在 **Secondary** 集群中为文件存储池启用 **Mirror**

在 Secondary 集群的 Control 节点上执行以下命令：

命令行

```
kubectl -n rook-ceph patch cephfilesystem <fs-name> \
--type merge -p '{"spec":{"mirroring":{"enabled": true}}}'
```

输出

```
cephfilesystem.ceph.rook.io/<fs-name> patched
```

参数：

- `<fs-name>`：文件存储池名称。

2 获取 Peer Token

该令牌是建立两个集群之间镜像连接的关键凭证。

在 Secondary 集群的 Control 节点上执行以下命令：

命令

```
kubectl get secret -n rook-ceph \
$(kubectl -n rook-ceph get cephfilesystem <fs-name> -o jsonpath
='{.status.info.fsMirrorBootstrapPeerSecretName}') \
-o jsonpath='{.data.token}' | base64 -d
```

输出

```
# Due to the involvement of sensitive information, the output ha
s been truncated.
eyJmc2lkIjogImMyYjAyNmMzLTA3ZGQtNDA3Z...
```

参数：

- `<fs-name>`：文件存储池名称。

3 在 Primary 集群中创建 Peer Secret

从 Secondary 集群获取 Peer Token 后，需要在 Primary 集群中创建 Peer Secret。

在 Primary 集群的 Control 节点上执行以下命令：

命令

```
kubectl -n rook-ceph create secret generic fs-secondary-site-sec
ret \
--from-literal=token=<token> \
--from-literal=pool=<fs-name>
```

输出

```
secret/fs-secondary-site-secret created
```

参数：

- `<token>`：在第 2 步中获取的令牌。
- `<fs-name>`：文件存储池名称。

4 在 Primary 集群中为文件存储池启用 Mirror

在 Primary 集群的 Control 节点上执行以下命令：

命令

```
kubectl -n rook-ceph patch cephfilesystem <fs-name> --type merge
-p \
'{
  "spec": {
    "mirroring": {
      "enabled": true,
      "peers": {
        "secretNames": [
          "fs-secondary-site-secret"
        ]
      },
      "snapshotSchedules": [
        {
          "path": "/",
          "interval": "<schedule-interval>"
        }
      ],
      "snapshotRetention": [
        {
          "path": "/",
          "duration": "<retention-policy>"
        }
      ]
    }
  }
}'
```

示例

```
kubectl -n rook-ceph patch cephfilesystem cephfs --type merge -p
\
'{
  "spec": {
    "mirroring": {
      "enabled": true,
      "peers": {
        "secretNames": [
          "fs-secondary-site-secret"
        ]
      },
      "snapshotSchedules": [
        {
          "path": "/",
          "interval": "1h"
        }
      ],
      "snapshotRetention": [
        {
          "path": "/",
          "duration": "h 1"
        }
      ]
    }
  }
}'
```

输出

```
cephfilesystem.ceph.rook.io/<fs-name> patched
```

参数：

- `<fs-name>`：文件存储池名称。
- `<schedule-interval>`：快照执行周期。详情请参阅官方文档↗。
- `<retention-policy>`：快照保留策略。详情请参阅官方文档↗。

5 在 Primary 集群中部署 Mirror Daemon

Mirror Daemon 会持续监控文件存储池中启用 Mirror 后的数据变化。它会定期创建快照，并通过网络将快照差异推送到 Secondary 集群。

在 Primary 集群的 Control 节点上执行以下命令：

命令

```
cat << EOF | kubectl apply -f -
apiVersion: ceph.rook.io/v1
kind: CephFilesystemMirror
metadata:
  name: cephfs-mirror
  namespace: rook-ceph
spec:
  placement:
    tolerations:
    - key: node-role.kubernetes.io/master
      operator: Exists
      effect: NoSchedule
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
    - key: node-role.kubernetes.io/cpaas-system
      operator: Exists
      effect: NoSchedule
  resources:
    limits:
      cpu: "500m"
      memory: "1Gi"
    requests:
      cpu: "500m"
      memory: "1Gi"
  priorityClassName: system-node-critical
EOF
```

输出

```
cephfilesystemmirror.ceph.rook.io/cephfs-mirror created
```

故障切换

当 Primary 集群发生故障时，可以直接继续使用 Secondary 集群中的 CephFS。

前提条件

Primary 集群的 Kubernetes 资源已备份并恢复到 Secondary 集群，包括 PVC、PV 和应用的工作负载。

块存储灾难恢复

RBD Mirror 是 Ceph 块存储（RBD）的一项功能，支持不同 Ceph 集群之间的异步数据复制，实现跨集群的灾难恢复（DR）。其核心功能是以主备模式同步数据，确保主集群故障时备份集群能够快速接管服务。

WARNING

- RBD Mirror 基于快照执行增量同步，默认快照间隔为每小时一次（可配置）。主备集群之间的差异数据通常对应于一个快照周期内的写入。
- RBD Mirror 仅提供底层存储数据备份，不包含 Kubernetes 资源备份。请使用平台的 备份与恢复 功能备份 PVC 和 PV 资源。

目录

术语

术语	说明
Primary Cluster	当前提供存储服务的集群。
Secondary Cluster	用作备份的备用集群。

备份配置

前提条件

- 准备两个能够部署 Alauda Build 版本 Rook-Ceph 的集群：Primary 集群和 Secondary 集群。
- 两个集群必须运行相同的平台版本（v3.12 或更高）。
- 在 Primary 和 Secondary 集群中分别创建分布式存储服务。
- 在 Primary 和 Secondary 集群中创建同名的块存储池。
- 请确保以下两个镜像已上传至平台私有镜像仓库：

- `quay.io/csiaddons/k8s-controller:v0.12.0` -> `<registry>/csiaddons/k8s-controller:v0.12.0`
- `quay.io/csiaddons/k8s-sidecar:v0.12.0` -> `<registry>/csiaddons/k8s-sidecar:v0.12.0`

网络前提条件

Primary 和 Secondary 集群之间的同步通过 Ceph 的 Public 网络进行，因此跨站点 Public 网络必须满足以下要求：

- 每个集群必须拥有独立的 Public 网络段。Primary 和 Secondary 集群的 Public 网络必须完全可路由且相互可达。
- Public 网络的持续带宽利用率不应超过 60%，以保证复制突发和故障切换场景下的充足余量。
- 两站点间的往返时延（RTT）应小于 30 毫秒。
- 两个 Public 网络间的丢包率应低于 0.05%，以保证复制性能的稳定和可预期。

操作步骤

引导对等节点（`Primary <-> Secondary`）

1. 启用 **Primary** 集群块存储池的镜像功能
   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

   ```
   kubectl -n rook-ceph patch cephblockpool <block-pool-name> \
     --type merge -p '{"spec":{"mirroring":{"enabled":true,"mode":"image"}}}'
   ```

   输出

   ```
   cephblockpool.ceph.rook.io/<block-pool-name> patched
   ```

参数说明：

- `<block-pool-name>`：块存储池名称。

2. 获取对等令牌

   该令牌是建立集群镜像连接的关键凭证。

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

   ```
   kubectl get secret -n rook-ceph \
     $(kubectl get cephblockpool.ceph.rook.io <block-pool-name> -
   n rook-ceph -o jsonpath='{.status.info.rbdMirrorBootstrapPeerS
   ecretName}') \
     -o jsonpath='{.data.token}' | base64 -d
   ```

   输出

   ```
   # 输出因敏感信息被截断
   eyJmc2lkIjoiMjc2N2I3ZmEtY2YwYi00N...
   ```

3. 在对等集群中创建对等令牌 **Secret**

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

   ```
   kubectl -n rook-ceph create secret generic rbd-peer-site-secre
   t \
     --from-literal=token=<token> \
     --from-literal=pool=<block-pool-name>
   ```

   输出

```
secret/rbd-peer-site-secret created
```

参数说明：

- `<token>`：从步骤 2获取的令牌。
  在 Primary 集群中，使用从 Secondary 集群获取的令牌配置此字段。
  在 Secondary 集群中，使用从 Primary 集群获取的令牌配置此字段。
- `<block-pool-name>`：块存储池名称。

4. 为块存储池打补丁添加对等 **Secret**
   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

命令

```
kubectl -n rook-ceph patch cephblockpool <block-pool-name> --type merge -p \
'{
  "spec": {
    "mirroring": {
      "peers": {
        "secretNames": [
          "rbd-peer-site-secret"
        ]
      }
    }
  }
}'
```

输出

```
cephblockpool.ceph.rook.io/<block-pool-name> patched
```

参数说明：

- `<block-pool-name>`：块存储池名称。

5. 部署 **Mirror** 守护进程

   该守护进程负责监控和管理 RBD 镜像同步进程，包括数据同步和错误处理。

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

```
cat << EOF | kubectl apply -f -
apiVersion: ceph.rook.io/v1
kind: CephRBDMirror
metadata:
  name: rbd-mirror
  namespace: rook-ceph
spec:
  count: 1
EOF
```

   输出

```
cephrbdmirror.ceph.rook.io/rbd-mirror created
```

6. 验证镜像状态

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

```
kubectl get cephblockpools.ceph.rook.io <block-pool-name> -n r
ook-ceph -o jsonpath='{.status.mirroringStatus.summary}'
```

   输出

```
# 所有 "OK" 状态表示正常运行
{"daemon_health":"OK","health":"OK","image_health":"OK","state
s":{}}
```

参数说明：

- `<block-pool-name>`：块存储池名称。

配置卷复制环境

该功能支持高效的数据复制与同步，且不会中断主应用操作，提升系统可靠性和可用性。

1. 部署 **CsiAddons Controller**

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   ```
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.12.0/deploy/controller/crds.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.12.0/deploy/controller/setup-controller.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.12.0/deploy/controller/rbac.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.12.0/deploy/controller/csi-addons-config.yaml

   kubectl -n csi-addons-system set image deployment/csi-addons-controller-manager manager=<registry>/csiaddons/k8s-controller:v0.12.0
   ```

   参数说明：

   - `<registry>`：平台镜像仓库地址。

2. 启用 **CsiAddons sidecar**

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

```
kubectl patch cm rook-ceph-operator-config -n rook-ceph --type jso
n --patch \
'[
  {
    "op": "add",
    "path": "/data/CSI_ENABLE_OMAP_GENERATOR",
    "value": "true"
  },
  {
    "op": "add",
    "path": "/data/CSI_ENABLE_CSIADDONS",
    "value": "true"
  },
  {
    "op": "add",
    "path": "/data/ROOK_CSIADDONS_IMAGE",
    "value": "<registry>/csiaddons/k8s-sidecar:v0.12.0"
  }
]'
```

等待所有 csi pod 成功重启

```
kubectl get po -n rook-ceph -w | grep csi
```

3. 创建 **VolumeReplicationClass**

   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

   命令

```
cat << EOF | kubectl apply -f -
apiVersion: replication.storage.openshift.io/v1alpha1
kind: VolumeReplicationClass
metadata:
  name: rbd-volumereplicationclass
spec:
  provisioner: rook-ceph.rbd.csi.ceph.com
  parameters:
    mirroringMode: snapshot
    schedulingInterval: "<scheduling-interval>" 1
    replication.storage.openshift.io/replication-secret-name:
rook-csi-rbd-provisioner
    replication.storage.openshift.io/replication-secret-namesp
ace: rook-ceph
EOF
```

输出

```
volumereplicationclass.replication.storage.openshift.io/rbd-vo
lumereplicationclass created
```

1 `<scheduling-interval>`：调度间隔，例如 schedulingInterval: "1h" 表示每小时执行一次。

为 **PVC** 启用镜像功能

在 Primary 集群的 Control 节点上执行以下命令：

命令

```
cat << EOF | kubectl apply -f -
apiVersion: replication.storage.openshift.io/v1alpha1
kind: VolumeReplication
metadata:
  name: <vr-name> 1
  namespace: <namespace> 2
spec:
  autoResync: false
  volumeReplicationClass: rbd-volumereplicationclass
  replicationState: primary
  dataSource:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: <pvc-name> 3
EOF
```

输出

```
volumereplication.replication.storage.openshift.io/<mirror-pvc-name> created
```

1. `<vr-name>`：VolumeReplication 对象名称，建议与 PVC 名称相同。
2. `<namespace>`：VolumeReplication 所属命名空间，必须与 PVC 命名空间一致。
3. `<pvc-name>`：需要启用镜像的 PVC 名称。

注意

启用后，Secondary 集群中的 RBD 镜像将变为只读。

计划迁移

使用场景：数据中心维护、技术更新、灾难规避等。

迁移切换

迁移切换是指将生产环境切换到备份设施（通常是恢复站点）或反向切换的过程。

迁移切换时，应停止对主站点镜像的访问，并将镜像在 Secondary 集群上设置为主状态，以恢复访问。

前提条件

- Primary 集群的 Kubernetes 资源已备份并恢复到 Secondary 集群，包括 PVC、PV、应用工作负载等。

操作步骤

按以下步骤将工作负载从 Primary 集群计划迁移到 Secondary 集群：

1. 缩减所有使用镜像 PVC 的应用 Pod（在 Primary 集群）。
2. 更新 Primary 集群中所有启用镜像的 PVC 的 VolumeReplication。
   将 `spec.replicationState` 设置为 `secondary`。
3. 在 Secondary 集群为所有启用镜像的 PVC 创建 VolumeReplication。

示例

```
cat << EOF | kubectl apply -f -
apiVersion: replication.storage.openshift.io/v1alpha1
kind: VolumeReplication
metadata:
  name: <vr-name> 1
  namespace: <namespace> 2
spec:
  autoResync: false
  volumeReplicationClass: rbd-volumereplicationclass
  replicationState: primary
  dataSource:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: <pvc-name> 3
EOF
```

1. `<vr-name>`：VolumeReplication 对象名称，建议与 PVC 名称相同。
2. `<namespace>`：VolumeReplication 所属命名空间，必须与 PVC 命名空间一致。

3 `<pvc-name>`：需要启用镜像的 PVC 名称。

4. 检查 VolumeReplication CR 状态，确认镜像在 Secondary 站点被标记为 `primary`。
5. 镜像标记为 `primary` 后，PVC 即可使用。此时可扩展应用以使用该 PVC。

灾难恢复

使用场景：自然灾害、电力故障、系统故障及崩溃等。

故障切换（突发停机）

发生灾难恢复时，在 Secondary 站点创建 VolumeReplication CR。

由于与 Primary 站点的连接丢失，operator 会自动向驱动发送 GRPC 请求，强制将数据源标记为 Secondary 站点的 `primary`。

前提条件

- Primary 集群的 Kubernetes 资源已备份并恢复到 Secondary 集群，包括 PVC、PV、应用工作负载等。

操作步骤

1. 在 Secondary 集群为所有启用镜像的 PVC 创建 VolumeReplication。

示例

```
cat << EOF | kubectl apply -f -
apiVersion: replication.storage.openshift.io/v1alpha1
kind: VolumeReplication
metadata:
  name: <vr-name> 1
  namespace: <namespace> 2
spec:
  autoResync: false
  volumeReplicationClass: rbd-volumereplicationclass
  replicationState: primary
  dataSource:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: <pvc-name> 3
EOF
```

1. `<vr-name>`：VolumeReplication 对象名称，建议与 PVC 名称相同。
2. `<namespace>`：VolumeReplication 所属命名空间，必须与 PVC 命名空间一致。
3. `<pvc-name>`：需要启用镜像的 PVC 名称。

2. 检查 VolumeReplication CR 状态，确认镜像在 Secondary 站点被标记为 `primary`。
3. 镜像标记为 `primary` 后，PVC 即可使用。此时可扩展应用以使用该 PVC。

故障恢复后的回切

当 Primary 站点的故障集群恢复后，若需从 Secondary 站点回切，按以下步骤操作：

前提条件

- Primary 集群的 Kubernetes 资源已备份并恢复到 Secondary 集群，包括 PVC、PV、应用工作负载等。

操作步骤

1. 缩减 Primary 站点正在运行的应用（如有），确保所有工作负载使用的持久卷不再被 Primary 集群使用。

2. 在 Primary 站点将 VolumeReplication CR 的 replicationState 从 `primary` 更新为 `secondary`。
3. 缩减 Secondary 站点的应用。
4. 在 Secondary 站点将 VolumeReplication CR 的 replicationState 从 `primary` 更新为 `secondary`。
5. 在 Primary 站点验证 VolumeReplication 状态，确认卷已标记为可用。
6. 卷标记为可用后，在 Primary 站点将 replicationState 从 `secondary` 更新为 `primary`。
7. 在 Primary 站点重新扩展应用。

对象存储灾备

Ceph RGW Multi-Site 功能是一种跨集群异步数据复制机制，旨在同步地理分布的 Ceph 集群之间的对象存储数据，提供高可用（HA）和灾难恢复（DR）能力。

目录

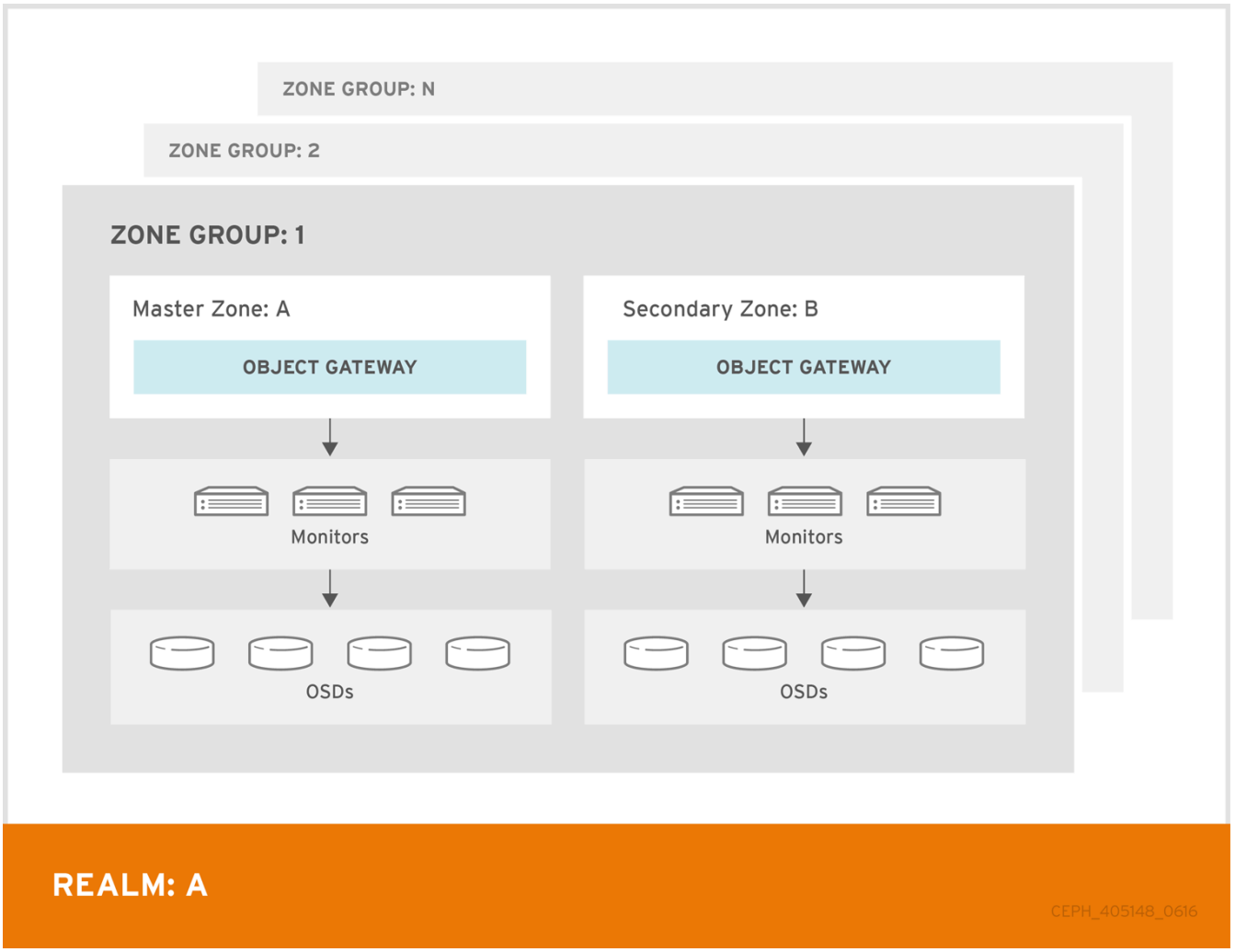
术语

术语	说明
Primary Cluster	当前提供存储服务的集群。
Secondary Cluster	用作备份的备用集群。
Realm, ZoneGroup, Zone	• Realm：Ceph 对象存储中最高级别的逻辑分组，表示完整的对象存储命名空间，通常用于多站点复制和同步。一个 Realm 可以跨越不同的地理位置或数据中心。 • ZoneGroup：Realm 内的逻辑分组，包含多个 Zone。ZoneGroup 实现跨 Zone 的数据同步和复制，通常位于同一地理区域内。 • Zone：ZoneGroup 内的逻辑分组，物理存储数据。每个 Zone 独立管理和存储对象，可以拥有自己的数据/元数据池配置。

前提条件

- 准备两个可部署 Rook-Ceph 的集群（Primary 和 Secondary 集群），并确保它们之间网络连通。
- 两个集群必须使用相同的平台版本（v3.12 或更高）。
- 确保 Primary 和 Secondary 集群上均未部署 Ceph 对象存储。
- 参考创建存储服务文档部署 Operator 并创建集群。集群创建完成后，不要通过向导创建对象存储池，而是使用以下描述的 CLI 工具进行配置。

架构



操作步骤

本指南提供同一 ZoneGroup 中两个 Zone 之间的同步方案。

1 在 **Primary** 集群创建对象存储

本步骤创建 Realm、ZoneGroup、Primary Zone 及 Primary Zone 的 gateway 资源。

在 Primary 集群的 Control 节点执行以下命令：

1. 设置参数

```
export REALM_NAME=<realm-name>
export ZONE_GROUP_NAME=<zonegroup-name>
export PRIMARY_ZONE_NAME=<primary-zone-name>
export PRIMARY_OBJECT_STORE_NAME=<primary-object-store-name>
```

参数说明：

- `<realm-name>`：Realm 名称。
- `<zonegroup-name>`：ZoneGroup 名称。
- `<primary-zone-name>`：Primary Zone 名称。
- `<primary-object-store-name>`：Gateway 名称。

2. 创建对象存储

命令

```
cat << EOF | kubectl apply -f -
---
apiVersion: ceph.rook.io/v1
kind: CephObjectRealm
metadata:
  name: $REALM_NAME
  namespace: rook-ceph

---
apiVersion: ceph.rook.io/v1
kind: CephObjectZoneGroup
metadata:
  name: $ZONE_GROUP_NAME
  namespace: rook-ceph
spec:
  realm: $REALM_NAME

---
apiVersion: ceph.rook.io/v1
kind: CephObjectZone
metadata:
  name: $PRIMARY_ZONE_NAME
  namespace: rook-ceph
spec:
  zoneGroup: $ZONE_GROUP_NAME
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
      requireSafeReplicaSize: true
  dataPool:
    failureDomain: host
    replicated:
      size: 3
      requireSafeReplicaSize: true
    parameters:
      compression_mode: none
  preservePoolsOnDelete: false

---
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
```

```
  labels:
    cpaas.io/builtin: "true"
  name: builtin-rgw-root
  namespace: rook-ceph
spec:
  name: .rgw.root
  application: rgw
  enableCrushUpdates: true
  failureDomain: host
  replicated:
    size: 3
  parameters:
    pg_num: "8"

---
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: $PRIMARY_OBJECT_STORE_NAME
  namespace: rook-ceph
spec:
  gateway:
    port: 7480
    instances: 2
  zone:
    name: $PRIMARY_ZONE_NAME
EOF
```

输出

```
cephobjectrealm.ceph.rook.io/<realm-name> created
cephobjectzonegroup.ceph.rook.io/<zonegroup-name> created
cephobjectzone.ceph.rook.io/<primary-zone-name> created
cephobjectstore.ceph.rook.io/<primary-object-store-name> creat
ed
```

2 配置 **Primary Zone** 的外部访问

1. 获取 ObjectStore 的 UID

```
export PRIMARY_OBJECT_STORE_UID=$(kubectl -n rook-ceph get cephobj
ectstore $PRIMARY_OBJECT_STORE_NAME -o jsonpath='{.metadata.uid}')
```

2. 创建外部访问 Service

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: rook-ceph-rgw-$PRIMARY_OBJECT_STORE_NAME-external
  namespace: rook-ceph
  labels:
    app: rook-ceph-rgw
    rook_cluster: rook-ceph
    rook_object_store: $PRIMARY_OBJECT_STORE_NAME
  ownerReferences:
    - apiVersion: ceph.rook.io/v1
      kind: CephObjectStore
      name: $PRIMARY_OBJECT_STORE_NAME
      uid: $PRIMARY_OBJECT_STORE_UID
spec:
  ports:
    - name: rgw
      port: 7480
      targetPort: 7480
      protocol: TCP
  selector:
    app: rook-ceph-rgw
    rook_cluster: rook-ceph
    rook_object_store: $PRIMARY_OBJECT_STORE_NAME
  sessionAffinity: None
  type: NodePort
EOF
```

3. 向 CephObjectZone 添加外部端点

```
IP=$(kubectl get nodes -l 'node-role.kubernetes.io/control-plane'
-o jsonpath='{.items[0].status.addresses[?(@.type=="InternalIP")].
address}' | cut -d ' ' -f1 | tr -d '\n')
PORT=$(kubectl -n rook-ceph get svc rook-ceph-rgw-$PRIMARY_OBJECT_
STORE_NAME-external -o jsonpath='{.spec.ports[0].nodePort}')
ENDPOINT=http://$IP:$PORT
kubectl -n rook-ceph patch cephobjectzone $PRIMARY_ZONE_NAME --typ
e merge -p "{\"spec\":{\"customEndpoints\":[\"$ENDPOINT\"]}}"
```

3 获取 `access-key` 和 `secret-key`

```
kubectl -n rook-ceph get secrets $REALM_NAME-keys -o jsonpath='{.dat
a.access-key}'
kubectl -n rook-ceph get secrets $REALM_NAME-keys -o jsonpath='{.dat
a.secret-key}'
```

4 创建 Secondary Zone 并配置 Realm 同步

本节说明如何创建 Secondary Zone 并通过拉取 Primary 集群的 Realm 信息配置同步。

在 Secondary 集群的 Control 节点执行以下命令：

1. 设置参数

```
export REALM_NAME=<realm-name>
export ZONE_GROUP_NAME=<zonegroup-name>

export REALM_ENDPOINT=<realm-endpoint>
export ACCESS_KEY=<access-key>
export SECRET_KEY=<secret-key>

export SECONDARY_ZONE_NAME=<secondary-zone-name>
export SECONDARY_OBJECT_STORE_NAME=<secondary-object-store-name>
```

参数说明：

- `<realm-name>`：Realm 名称。
- `<zone-group-name>`：ZoneGroup 名称。

- `<realm-endpoint>`：从 Primary 集群获取的外部地址。
- `<access-key>`：从此处获取的 AK。
- `<secret-key>`：从此处获取的 SK。
- `<secondary-zone-name>`：Secondary Zone 名称。
- `<secondary-object-store-name>`：Secondary Gateway 名称。

2. 创建 **Secondary Zone** 并配置 **Realm** 同步

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: $REALM_NAME-keys
  namespace: rook-ceph
data:
  access-key: $ACCESS_KEY
  secret-key: $SECRET_KEY

---
apiVersion: ceph.rook.io/v1
kind: CephObjectRealm
metadata:
  name: $REALM_NAME
  namespace: rook-ceph
spec:
  pull:
    endpoint: $REALM_ENDPOINT

---
apiVersion: ceph.rook.io/v1
kind: CephObjectZoneGroup
metadata:
  name: $ZONE_GROUP_NAME
  namespace: rook-ceph
spec:
  realm: $REALM_NAME

---
apiVersion: ceph.rook.io/v1
kind: CephObjectZone
metadata:
  name: $SECONDARY_ZONE_NAME
  namespace: rook-ceph
spec:
  zoneGroup: $ZONE_GROUP_NAME
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
      requireSafeReplicaSize: true
  dataPool:
```

```
    failureDomain: host
    replicated:
      size: 3
      requireSafeReplicaSize: true
  preservePoolsOnDelete: false

---
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  labels:
    cpaas.io/builtin: "true"
  name: builtin-rgw-root
  namespace: rook-ceph
spec:
  name: .rgw.root
  application: rgw
  enableCrushUpdates: true
  failureDomain: host
  replicated:
    size: 3
  parameters:
    pg_num: "8"

---
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: $SECONDARY_OBJECT_STORE_NAME
  namespace: rook-ceph
spec:
  gateway:
    port: 7480
    instances: 2
  zone:
    name: $SECONDARY_ZONE_NAME
EOF
```

5 配置 **Secondary Zone** 的外部访问

1. 获取 Secondary Gateway 的 UID

```
export SECONDARY_OBJECT_STORE_UID=$(kubectl -n rook-ceph get cepho
bjectstore $SECONDARY_OBJECT_STORE_NAME -o jsonpath='{.metadata.ui
d}')
```

2. 创建外部访问 Service

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: rook-ceph-rgw-$SECONDARY_OBJECT_STORE_NAME-external
  namespace: rook-ceph
  labels:
    app: rook-ceph-rgw
    rook_cluster: rook-ceph
    rook_object_store: $SECONDARY_OBJECT_STORE_NAME
  ownerReferences:
    - apiVersion: ceph.rook.io/v1
      kind: CephObjectStore
      name: $SECONDARY_OBJECT_STORE_NAME
      uid: $SECONDARY_OBJECT_STORE_UID
spec:
  ports:
    - name: rgw
      port: 7480
      targetPort: 7480
      protocol: TCP
  selector:
    app: rook-ceph-rgw
    rook_cluster: rook-ceph
    rook_object_store: $SECONDARY_OBJECT_STORE_NAME
  sessionAffinity: None
  type: NodePort
EOF
```

3. 向 Secondary CephObjectZone 添加外部端点

```
IP=$(kubectl get nodes -l 'node-role.kubernetes.io/control-plane'
-o jsonpath='{.items[0].status.addresses[?(@.type=="InternalIP")].
address}' | cut -d ' ' -f1 | tr -d '\n')
PORT=$(kubectl -n rook-ceph get svc rook-ceph-rgw-$SECONDARY_OBJEC
T_STORE_NAME-external -o jsonpath='{.spec.ports[0].nodePort}')
ENDPOINT=http://$IP:$PORT
kubectl -n rook-ceph patch cephobjectzone $SECONDARY_ZONE_NAME --t
ype merge -p "{\"spec\":{\"customEndpoints\":[\"$ENDPOINT\"]}}"
```

6 检查 Ceph 对象存储同步状态

在 Secondary 集群的 `rook-ceph-tools` pod 中执行以下命令

```
# 进入 rook-ceph-tools pod
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get po -l app=ro
ok-ceph-tools -o jsonpath='{range .items[*]}{@.metadata.name}') -- ba
sh

radosgw-admin sync status
```

输出示例

```
          realm 962d6b80-b218-4fc8-8198-e498fabc4e9d (relam-primary)
      zonegroup 9de3acd7-0b01-4a04-ac84-1421c6103a16 (zonegroup-prima
ry)
           zone 7b3ce7f5-7090-46f6-afb1-d1bb156053da (zone-secondary)
   current time 2025-12-04T06:27:15Z
zonegroup features enabled: resharding
                    disabled: compress-encrypted
  metadata sync syncing
                full sync: 0/64 shards
                incremental sync: 64/64 shards
                metadata is caught up with master
      data sync source: 6319ca70-964e-47be-8b96-7c5bf5a128b1 (zone-pr
imary)
                        syncing
                        full sync: 0/128 shards
                        incremental sync: 128/128 shards
                        data is caught up with source
```

> `metadata is caught up with master` 和 `data is caught up with source` 表示同步状态正常。

故障切换

当 Primary 集群发生故障时，需要将 Secondary Zone 提升为 Primary Zone。切换后，Secondary Zone 的 gateway 可以继续提供对象存储服务。

操作步骤

在 Secondary 集群的 `rook-ceph-tools` pod 中执行以下命令

```
# 进入 rook-ceph-tools pod
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get po -l app=rook-ceph-tools -o jsonpath='{range .items[*]}{@.metadata.name}') -- bash

# 将恢复的 zone 设置为主 zone 和默认 zone
radosgw-admin zone modify --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name> --rgw-zone=<secondary-zone-name> --master

# 更新 period 使更改生效
radosgw-admin period update --commit --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name>
```

参数

- `<realm-name>`：Realm 名称。
- `<zone-group-name>`：Secondary Zone Group 名称。
- `<secondary-zone-name>`：Secondary Zone 名称。

故障恢复

当主站点的故障集群恢复后，若需要从 Secondary 站点切换回主站点，请按以下步骤操作。

操作步骤

在 Primary 集群的 `rook-ceph-tools` pod 中执行以下命令

```
# 进入 rook-ceph-tools pod
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get po -l app=rook-ceph-tools -o jsonpath='{range .items[*]}{@.metadata.name}') -- bash

# 检查同步状态，等待 Secondary 站点同步完成
radosgw-admin sync status

#           realm 962d6b80-b218-4fc8-8198-e498fabc4e9d (relam-primary)
#       zonegroup 9de3acd7-0b01-4a04-ac84-1421c6103a16 (zonegroup-primary)
#            zone 6319ca70-964e-47be-8b96-7c5bf5a128b1 (zone-primary)
#    current time 2025-12-04T07:18:26Z
# zonegroup features enabled: resharding
#                   disabled: compress-encrypted
#   metadata sync syncing
#                 full sync: 0/64 shards
#                 incremental sync: 64/64 shards
#                 metadata is caught up with master
#       data sync source: 7b3ce7f5-7090-46f6-afb1-d1bb156053da (zone-secondary)
#                         syncing
#                         full sync: 0/128 shards
#                         incremental sync: 128/128 shards
#                         data is caught up with source

# 将恢复的 zone 设置为主 zone 和默认 zone
radosgw-admin zone modify --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name> --rgw-zone=<primary-zone-name> --master

# 更新 period 使更改生效
radosgw-admin period update --commit --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name>
```

参数

- `<realm-name>`：Realm 名称。
- `<zone-group-name>`：Zone Group 名称。

- `<primary-zone-name>`：Primary Zone 名称。

更新优化参数

平台支持在创建存储集群时填写 Ceph 配置文件格式的优化参数，但创建后不提供通过界面修改的方式。您需要按照以下操作步骤手动更新。

目录

操作步骤

1. 首先，将存储优化参数更新到名为 `rook-config-override-user` 的 Configmap 中，替换 `.data.config` 字段，并将 `.metadata.annotations[rook.cpaas.io/need-sync]` 字段的值设置为 `true`。例如：

```
apiVersion: v1
data:
  config: |
    [global]
    mon_memory_target=1073741824
    mds_cache_memory_limit=2147483648
    osd_memory_target=4147483648
kind: ConfigMap
metadata:
  annotations:
    cpaas.io/creator: admin
    cpaas.io/updated-at: "2022-03-01T12:24:04Z"
    rook.cpaas.io/need-sync: "true"
    rook.cpaas.io/sync-status: synced
  creationTimestamp: "2022-03-01T12:24:04Z"
  finalizers:
  - rook.cpaas.io/config-merge
  name: rook-config-override-user
  namespace: default
  resourceVersion: "38816864"
  uid: ce3a8f3e-6453-4bdd-bff0-e16cf7d5d5fa
```

2. 在 rook-ceph-tools 的 Pod 中执行 `ceph tell [mon|osd|mgr|mds|rgw].* config set [key] [value]`，以实时应用配置。
3. 若要启动 tools Pod，编辑 rook-ceph 命名空间下的 ClusterServiceVersion (CSV)，将 Deployments 部分中 rook-ceph-tools 的 replicas 值设置为 1。

创建 Ceph 对象存储用户

我们允许通过自定义资源定义（CRDs）创建和自定义对象存储用户。

目录

前提条件

- 对象存储池已创建

操作步骤

1 创建用户

在集群的控制节点上执行命令。

```
cat << EOF | kubectl apply -f -
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: <name>
  namespace: <namespace>
spec:
  store: <ObjectStore>
  displayName: <displayName>
  clusterNamespace: <clusterNamespace>
  quotas:
    maxBuckets: 100
    maxSize: -1
    maxObjects: -1
  capabilities:
    user: "*"
    bucket: "*"
EOF
```

参数说明

参数	描述
name	要创建的对象存储用户的名称。
namespace	创建对象存储用户的命名空间。
displayName	显示名称。
clusterNamespace	父级 `CephCluster` 和 `CephObjectStore` 所在的命名空间。如果未指定，用户必须与集群和对象存储处于同一命名空间。要启用此功能，CephObjectStore 的 `allowUsersInNamespaces` 必须包含该用户的命名空间。
ObjectStore	用户将被创建的对象存储。这与对象存储池的名称匹配。
quotas	可选 表示可以为用户设置的配额限制。 • maxBuckets：用户的最大桶数量限制。默认值为 `100`。

参数	描述
	• maxSize：所有用户桶中对象的最大总大小限制。设置为 `-1` 表示无限制。 • maxObjects：所有用户桶中对象的最大数量限制。设置为 `-1` 表示无限制。
capabilities	可选 Ceph 允许为用户赋予额外权限。此设置目前仅能在创建对象存储用户时使用。如果需要修改用户权限，必须删除并重新创建用户。详情请参见 Ceph 文档↗。支持为以下资源添加 `read`、`write`、`read,write` 或 `*` 权限： • user • buckets • usage • metadata • zone • roles • info • amz-cache • bilog • mdlog • datalog • user-policy • odic-provider • ratelimit

2 允许在其他命名空间创建用户

如果在除 Rook 集群命名空间之外的命名空间中创建 CephObjectStoreUser，则必须将该命名空间添加到允许的命名空间列表中，或者指定 "*" 以允许所有命名空间。这对于需要

在其自身命名空间中创建对象存储凭据的应用程序非常有用。

在集群的控制节点上执行命令。

```
kubectl -n rook-ceph patch cephobjectstore <ObjectStore> --type merge
-p '{"spec":{"allowUsersInNamespaces":["*"]}}'
```

3 获取用户信息

在集群的控制节点上执行命令。

```
user_secret=$(kubectl -n <namespace> get cephobjectstoreuser <user-na
me> -o jsonpath='{.status.info.secretName}')

# ACCESS_KEY
kubectl -n <namespace> get secret $user_secret -o jsonpath='{.data.Ac
cessKey}' | base64 --decode

# SECRET_KEY
kubectl -n <namespace> get secret $user_secret -o jsonpath='{.data.Se
cretKey}' | base64 --decode
```

设置存储池配额

池配额是应用于 Ceph 存储池级别的逻辑数据大小限制。
它控制池可以写入的逻辑数据量，实际物理空间为 配额 * 池副本数。

目录

前提条件

- 已安装 Ceph 集群
- 已创建 Ceph 存储池

为文件存储池设置池配额

在集群的控制节点上执行命令。

Command

```
SIZE="<size>"
POOL_NAME="<fs-pool-name>"

kubectl patch cephfilesystem $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/dataPools/0/quotas/maxSize
\",\"value\":\"$SIZE\"}]"
```

Example

```
SIZE="100Gi"
POOL_NAME="cephfs"

kubectl patch cephfilesystem $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/dataPools/0/quotas/maxSize
\",\"value\":\"$SIZE\"}]"
```

参数说明

参数	描述
size	以带单位后缀的字符串形式表示的配额大小（例如 "10Gi"）
fs-pool-name	文件存储池的名称

为块存储池设置池配额

在集群的控制节点上执行命令。

Command

```
SIZE="<size>"
POOL_NAME="<block-pool-name>"

kubectl patch cephblockpool $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/quotas/maxSize\",\"value
\":\"$SIZE\"}]"
```

Example

```
SIZE="100Gi"
POOL_NAME="rbd"

kubectl patch cephblockpool $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/quotas/maxSize\",\"value
\":\"$SIZE\"}]"
```

参数说明

参数	描述
size	以带单位后缀的字符串形式表示的配额大小（例如 "10Gi"）
block-pool-name	块存储池的名称

为对象存储池设置池配额

在集群的控制节点上执行命令。

Command

```
SIZE="<size>"
POOL_NAME="<object-pool-name>"

kubectl patch cephobjectstore $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/dataPool/quotas/maxSize
\",\"value\":\"$SIZE\"}]"
```

Example

```
SIZE="100Gi"
POOL_NAME="object"

kubectl patch cephobjectstore $POOL_NAME -n rook-ceph --type=json \
    -p "[{\"op\":\"add\",\"path\":\"/spec/dataPool/quotas/maxSize
\",\"value\":\"$SIZE\"}]"
```

参数说明

参数	描述
size	以带单位后缀的字符串形式表示的配额大小（例如 "10Gi"）
object-pool-name	对象存储池的名称

通过 **Ceph** 终端验证池配额

```
ceph osd pool ls detail | grep max_bytes

pool 3 'cephfs-data0' replicated size 3 min_size 2 crush_rule 4 object_ha
sh rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 100 lfor
0/0/56 flags hashpspool max_bytes 107374182400 stripe_width 0 application
cephfs read_balance_score 1.31
pool 4 'rbd' replicated size 3 min_size 2 crush_rule 5 object_hash rjenki
ns pg_num 32 pgp_num 32 autoscale_mode on last_change 117 lfor 0/0/111 fl
ags hashpspool,selfmanaged_snaps max_bytes 107374182400 stripe_width 0 ap
plication rbd read_balance_score 1.31
pool 12 'object.rgw.buckets.data' replicated size 3 min_size 2 crush_rule
13 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_chang
e 304 lfor 0/0/168 flags hashpspool max_bytes 107374182400 stripe_width 0
compression_mode none application rgw read_balance_score 1.59
```

为 Ceph RGW 启用 D3N 缓存

D3N（Data Delivery Network）缓存通过使用 **RGW** 节点上的本地 **NVMe/SSD** 磁盘作为缓存层，加速对热点对象的访问。
它减少了对后端 OSD 的频繁读取操作，显著提升了热点数据的读取性能。

目录

背景

D3N 的核心思想简单而有效：
RGW 不再反复从 Ceph 后端存储获取对象数据，而是直接从位于 RGW 节点上快速磁盘

（NVMe/SSD）上的 本地持久缓存 提供热点数据。

该设计特别适用于：

- 大对象下载
- 热点对象的重复读取
- 对带宽敏感的工作负载

前提条件

- 已安装 Ceph 集群
- 已部署 Rook-Ceph
- 已创建对象存储（RGW / CephObjectStore）
- RGW 节点上有高性能本地磁盘（NVMe / SSD）

操作概览

1. 部署 Ceph 并创建对象存储
2. 在 RGW 节点上准备高速本地磁盘并挂载到目录
3. 配置 `CephObjectStore` ：
   - 通过 `hostPath` 挂载该目录
   - 启用与 D3N 相关的 RGW 参数

准备本地文件系统作为缓存

在每个运行 RGW 服务的节点上挂载一个目录。建议使用高性能的专用磁盘（而非分区），格式化为 **XFS** 文件系统，并配置 `/etc/fstab` 以确保重启后挂载持续生效。后续文档中出现的 `</path/to/cache/dir>` 请替换为你实际配置的目录路径。

在 **CephObjectStore** 中启用 **D3N** 缓存

编辑 `CephObjectStore` 资源：

```
kubectl edit cephobjectstore object-store -n rook-ceph
```

在 `gateway` 下添加如下配置：

```
gateway:
  additionalVolumeMounts:
  - subPath: cache
    volumeSource:
      hostPath:
        path: </path/to/cache/dir>
  rgwConfig:
    rgw_d3n_l1_datacache_persistent_path: /var/rgw/cache/
    rgw_d3n_l1_datacache_size: "10737418240"
    rgw_d3n_l1_local_datacache_enabled: "true"
```

参数说明

参数	说明
`rgw_d3n_l1_local_datacache_enabled`	启用 D3N 本地缓存
`rgw_d3n_l1_datacache_persistent_path`	RGW 容器内的缓存目录
`rgw_d3n_l1_datacache_size`	最大缓存大小（字节）

验证 **D3N** 配置

通过 **Ceph** 工具检查 **RGW** 配置

在 `rook-ceph-tools` pod 中运行：

```
ceph config dump | grep d3n
```

示例输出：

```
client.rgw.obj.a  advanced  rgw_cache_enabled                     true
client.rgw.obj.a  advanced  rgw_d3n_l1_datacache_persistent_path /var/rg
w/cache/
client.rgw.obj.a  advanced  rgw_d3n_l1_datacache_size            10737418
240
client.rgw.obj.a  advanced  rgw_d3n_l1_local_datacache_enabled   true
```

验证 **RGW** 日志

RGW pod 重启后，应出现 D3N 初始化日志：

```
kubectl logs -n rook-ceph rook-ceph-rgw-object-store-a-xxxx | grep -i d3n
```

示例输出：

```
rgw_d3n: rgw_d3n_l1_local_datacache_enabled=1
rgw_d3n: rgw_d3n_l1_datacache_persistent_path='/var/rgw/cache/'
rgw_d3n: rgw_d3n_l1_datacache_size=10737418240
rgw_d3n: rgw_d3n_l1_evict_cache_on_start=1
rgw_d3n: rgw_d3n_l1_eviction_policy=lru
```

验证缓存行为

通过 RGW 下载对象时，缓存文件会出现在宿主机的缓存目录中：

```
ls </path/to/cache/dir>
```

示例输出：

```
40b934ab-6c7e-45e7-9fed-a35bc143ce95...multipart_v2v-demo.mkv.1
40b934ab-6c7e-45e7-9fed-a35bc143ce95...multipart_v2v-demo.mkv.2
40b934ab-6c7e-45e7-9fed-a35bc143ce95...shadow_v2v-demo.mkv.3_1
...
```

这些文件的存在证明 RGW 正在从本地 D3N 缓存提供数据。

配置传输加密

本主题介绍如何基于 `msgr2` 为 ACP 分布式存储启用或禁用传输加密。

目录

概述

Ceph `msgr2` 是 Ceph messenger 协议的第二代。它支持两种连接模式：

- `crc`：对等端身份验证并校验数据完整性，但不加密负载数据。
- `secure`：对传输流量进行加密，并提供加密完整性保护。

在 ACP 分布式存储中，传输加密由 `CephCluster.spec.network.connections.encryption.enabled` 控制。

限制与前提条件

启用此功能前，请注意以下限制：

- **ACP** 版本
  - v4.3.0 及以后版本。
- 操作系统和内核（**ceph** 守护进程和客户端节点）
  - 内核版本 `5.11` 及以后。
  - `Ubuntu 22.04` 及以后。

> **WARNING**
>
> 传输加密会增加 CPU 开销，可能降低吞吐量或增加延迟，尤其是在繁忙的存储节点或低频 CPU 上。请先在预发布环境评估影响。

为新集群启用传输加密

如果存储集群尚未创建，请在创建前在 `CephCluster` 清单中添加以下字段：

```
spec:
  network:
    connections:
      encryption:
        enabled: true
```

集群创建完成后，验证：

- CephFS PVC 仍能成功挂载
- 所有节点上的 RBD 和 CephFS 工作负载使用支持的内核版本

部署后启用传输加密

如果集群已运行，仅修改加密开关是风险最低的方式。

步骤 1. 确认节点内核版本

在所有挂载 Ceph 卷的 Kubernetes 节点上运行以下命令，确认内核版本满足前提条件：

```
uname -r
```

如果部分工作节点不满足要求，升级这些节点之前请勿在生产集群启用传输加密。

步骤 2. 在 CephCluster 中启用加密

```
kubectl patch cephcluster ceph-cluster -n rook-ceph --type merge -p '
spec:
  network:
    connections:
      encryption:
        enabled: true
'
```

步骤 3. 等待配置生效

配置更新后：

- 检查相关 Pod 是否正常重启
- 重新创建一个挂载 CephFS 或 RBD PVC 的测试 Pod
- 确认 I/O 正常工作

禁用传输加密

在现有集群中禁用加密

仅禁用传输加密，同时保持 `msgr2` 可用：

```
kubectl patch cephcluster ceph-cluster -n rook-ceph --type merge -p '
spec:
  network:
    connections:
      encryption:
        enabled: false
'
```

验证

启用功能后，从 Kubernetes 和 Ceph 两方面验证集群状态。

检查 **CephCluster** 设置

```
kubectl get cephcluster ceph-cluster -n rook-ceph -o yaml
```

确认输出包含：

```yaml
spec:
  network:
    connections:
      encryption:
        enabled: true
```

检查客户端兼容性

启用传输加密后：

- 使用 `msgr2 secure` 的客户端应能正常连接
- 配置为非加密模式（如 `legacy` 或 `crc`）的客户端将无法连接

检查工作负载挂载

创建或重启一个挂载以下 PVC 的测试工作负载：

- CephFS PVC
- RBD PVC

然后验证：

- Pod 成功启动
- 文件系统可读写
- CSI 或工作负载日志中无挂载相关错误

故障排查建议

启用加密导致挂载失败或服务中断时，优先检查：

1. 节点内核版本不满足要求。
2. 部分节点或外部客户端不支持 `msgr2 secure`，或仍配置为 `ms_mode=legacy` 或 `ms_mode=crc`。
3. 网络策略、防火墙或安全组未放通端口 `3300`。

4. 启用加密后 CPU 资源不足。

若变更影响生产工作负载，先禁用加密，再排查兼容性和性能瓶颈。

性能影响

ACP 无法给出 `msgr2 secure` 开销的固定百分比，实际影响取决于 CPU 型号、是否支持 AES 加速、网络带宽、I/O 大小，以及工作负载是 CPU 绑定还是网络绑定。

实际情况：

- 延迟通常略有增加，且在小 I/O 或延迟敏感工作负载上更明显
- 客户端和 Ceph 守护进程的 CPU 使用率通常都会增加，因为流量必须加密并保证完整性
- 对高吞吐量工作负载、较慢 CPU 或无强 AES 加速环境影响更明显

作为运维估算，使用带 AES-NI 的现代 x86 CPU 时，合理的初始预期为：

- 平均延迟增加约 `5%` 到 `15%`
- 处理重 I/O 的存储和客户端节点 CPU 使用率增加约 `10%` 到 `30%`

以上数值为工程估算，非产品保证。生产环境启用加密前，请在预发布环境对代表性工作负载进行基准测试，至少对比以下指标：

- 平均和 P99 读写延迟
- 客户端节点 CPU 使用率
- OSD 节点 CPU 使用率
- 吞吐量和 IOPS

设置 Ceph OSD 满容量阈值

本主题介绍如何调整 ACP 分布式存储中 Ceph OSD 的容量阈值。您可以直接在 Ceph 中更改阈值，也可以在 `CephCluster` 自定义资源中声明式管理它们。

Ceph 使用三个阈值来控制随着 OSD 使用率增加时集群的行为：

阈值	作用
`nearfull`	提前发出集群接近满容量的预警。
`backfillfull`	防止 Ceph 向已经过满的 OSD 回填更多数据。
`full`	当 OSD 达到配置的限制时，停止写入以保护集群。

WARNING

始终保持阈值按升序排列：`nearfull` < `backfillfull` < `full`。将值设置得过于接近 `1.0` 可能导致集群没有恢复空间。

目录

前提条件

- 您拥有 ACP 集群的 cluster-admin 权限。
- `rook-ceph` 命名空间中已部署 `rook-ceph-tools`，或者您被允许临时启动该部署。
- 您了解集群为何接近满容量，并且在紧急调整后有添加存储或清理数据的计划。

操作步骤

1 检查当前集群状态

如果 `rook-ceph-tools` Pod 未运行，先启动它：

```
kubectl -n rook-ceph scale deploy rook-ceph-tools --replicas=1
```

检查集群整体健康状态：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

查看当前阈值：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- \
  ceph osd dump | egrep 'nearfull_ratio|backfillfull_ratio|full_rati
o'
```

输出示例：

```
full_ratio 0.95
backfillfull_ratio 0.9
nearfull_ratio 0.85
```

2 通过 ceph CLI 设置阈值

使用 Ceph 命令直接更改集群生效的阈值。

例如，稍微提高阈值：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- \
  ceph osd set-nearfull-ratio 0.88

kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- \
  ceph osd set-backfillfull-ratio 0.92

kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- \
  ceph osd set-full-ratio 0.96
```

使用最小的提升幅度以恢复集群进度。更改值后，应尽快进行容量扩展或数据清理。

如果写入被阻塞且 OSD 处于卡住、挂起或无法恢复状态，先停止应用 I/O，然后仅小幅提高 `full` 阈值，等待重平衡完成，集群恢复稳定后再恢复阈值。

3 通过更新 `CephCluster` CR 设置阈值

如果想覆盖默认设置，可以通过更新 `CephCluster` CR 来设置 Ceph OSD 满容量阈值。

```
kubectl patch cephcluster ceph-cluster -n rook-ceph --type merge -p \
'{"spec":{"storage":{"nearFullRatio":0.88,"backfillFullRatio":0.92,"f
ullRatio":0.96}}}'
```

如果只需更改一个阈值，只补丁该字段。例如：

```
kubectl patch cephcluster ceph-cluster -n rook-ceph --type merge -p \
'{"spec":{"storage":{"fullRatio":0.96}}}'
```

4 验证应用的值

验证 Kubernetes 中持久化的设置：

```
kubectl get cephcluster ceph-cluster -n rook-ceph -o yaml
```

验证 Ceph 中生效的运行时值：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- \
  ceph osd dump | egrep 'nearfull_ratio|backfillfull_ratio|full_rati
o'
```

重新检查集群健康和重平衡状态：

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

当这些字段在 `CephCluster` 中设置时，该资源即成为阈值的声明式来源。如果 ACP 或 Rook 后续对集群配置进行调和，`CephCluster` 中的值应被视为预期的基线。

5 恢复或重新基线阈值

在添加容量、删除数据或完成重平衡后，决定是否保留新阈值。如果较高的值仅作为紧急应对措施，补丁 `CephCluster` 资源回到标准基线，并确认运行时值也恢复到预期状态。

建议

- 将阈值更改视为临时缓解措施，而非容量规划的替代方案。
- 如果只有少数 OSD 明显比其他更满，检查 OSD 利用率分布。
- 在更改前记录原始阈值，以便集群稳定后恢复。

持久卷加密

目录

概述

持久卷（PV）加密有助于保护有状态工作负载的数据机密性和租户隔离。 对于 ACP 分布式存储，CSI 预配的 PV 的静态加密通过外部密钥管理系统（KMS）和启用加密的 StorageClass 实现。

本章介绍如何：

- 配置 KMS 访问凭证和连接详情
- 创建启用加密的 StorageClass
- 启用和禁用密钥轮转策略

> **WARNING**
>
> 本章中的持久卷加密适用于基于 RBD 的 PV。不包含对象存储存储桶。

密钥管理系统（KMS）的访问配置

请先配置 KMS 访问。基于 StorageClass 的 PV 加密依赖于有效的 KMS 凭证和连接元数据。

不同的提供商模式使用不同的认证和元数据模型：

模式	`encryptionKMSType`	认证材料
Vault token 模式	`vaulttokens`	存储在 Secret 中的静态 Vault token
Vault tenant ServiceAccount 模式	`vaulttenantsa`	映射到 Vault 角色的 Kubernetes ServiceAccount 身份
Thales CipherTrust Manager 模式	`kmip`	KMIP 端点 + 客户端证书链 + key UID

> ⚠️ 重要：**Alpha** 功能免责声明
> 使用 `vaulttenantsa` 访问 KMS 处于 **Alpha** 阶段，仅用于早期技术验证和评估。作为功能提供方，我们不对其稳定性、可靠性或数据完整性提供任何保证。通过配置和使用此功能，您即表示已知悉并接受以下技术限制：

> - 不适用于生产环境： 此功能缺少全面的系统级验证。将其部署到生产环境会带来较高的进程失败或数据损坏风险。
> - 不保证 **SLA**： 此功能不在标准服务级别协议（SLA）范围内。我们不保证技术支持响应时间，也不提供紧急热修复。
> - **Breaking Changes** 与弃用： API 端点、配置 schema 和核心处理逻辑在未来版本中可能发生不向后兼容的更改。该功能也可能在未提前通知的情况下被完全弃用。
> - 无法平滑升级： 我们不提供版本之间的升级脚本或数据迁移工具。升级到后续版本通常需要完全清除现有资源并重新部署。由此导致的任何状态或数据丢失均由用户自行承担责任。

使用 `vaulttokens` 配置访问

前提条件

- ACP 分布式存储集群处于 `Ready` 状态。
- 您具有 `cluster-admin` 权限。
- 集群节点可以访问 Vault。
- 您拥有一个可访问目标后端路径的有效 Vault token。

操作步骤

1. 创建一个用于存储 Vault token 的 Secret。

```
kubectl -n rook-ceph create secret generic ceph-csi-kms-token \
  --from-literal=token='<vault_token>'
```

2. 创建或更新 KMS 连接 `ConfigMap` 条目。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-kms-connection-details
  namespace: rook-ceph
data:
  vault-token-kms: |
    {
       "encryptionKMSType": "vaulttokens",
       "vaultAddress": "<https://hostname_or_ip_of_vault_server:port>",
       "vaultBackend": "<vault backend>",
       "vaultBackendPath": "<vault backend path name>",
       "vaultTLSServerName": "<vault TLS server name>",
       "vaultCAVerify": "true",
       "vaultCAFromSecret": "<secret containing CA cert>",
       "vaultClientCertFromSecret": "<secret containing client cert>",
       "vaultClientCertKeyFromSecret": "<secret containing client priva
te key>"
    }
```

参数

参数	说明	必需
`encryptionKMSType`	KMS 提供商模式。使用 `vaulttokens` 进行静态 Vault token 认证。	是
`vaultAddress`	Vault 服务端点，包括协议和端口。	是
`vaultBackend`	Vault secret engine，默认值：kv-v2	否
`vaultBackendPath`	用于 data-encryption keys（DEKs）的 Vault secret engine 路径。默认值：secret/	否
`vaultTLSServerName`	用于 SNI 和证书主机名验证的 TLS server name。	否

参数	说明	必需
`vaultCAVerify`	连接 Vault server 时是否需要 CA 认证，默认值：true	否
`vaultCAFromSecret`	存储 Vault CA 证书的 Secret 名称。	否
`vaultClientCertFromSecret`	存储 Vault 客户端证书的 Secret 名称。	否
`vaultClientCertKeyFromSecret`	存储 Vault 客户端私钥的 Secret 名称。	否

验证步骤

1. 验证 token Secret：

```
kubectl -n rook-ceph get secret ceph-csi-kms-token
```

2. 验证 `ConfigMap` 条目，并确认 `encryptionKMSType: vaulttokens`：

```
kubectl -n rook-ceph get configmap csi-kms-connection-details -o yaml
```

使用 `vaulttenantsa` 配置访问

前提条件

- ACP 分布式存储集群处于 `Ready` 状态。
- 在外部密钥管理系统（KMS）中，
  - 确保已存在 policy，并且 Vault 中的 key value backend path 已启用。
  - 确保 Vault server 使用的是签名证书。

操作步骤

在 ACP 分布式存储可以使用 `Vault` 进行认证并开始使用之前，需要先配置 Kubernetes 认证方式。以下说明将创建并配置 ACP 分布式存储通过 `Vault` 进行认证所需的 `serviceAccount`、`ClusterRole` 和 `ClusterRoleBinding`。

1. 在工作负载 namespace 中创建 tenant ServiceAccount。

```
kubectl -n <tenant_namespace> create serviceaccount ceph-csi-vault-sa
```

2. 在存储 namespace 中应用用于 Vault token review 的 RBAC 资源。

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rbd-csi-vault-token-review
  namespace: rook-ceph
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-vault-token-review
rules:
  - apiGroups: ["authentication.k8s.io"]
    resources: ["tokenreviews"]
    verbs: ["create", "get", "list"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-vault-token-review
subjects:
  - kind: ServiceAccount
    name: rbd-csi-vault-token-review
    namespace: rook-ceph
roleRef:
  kind: ClusterRole
  name: rbd-csi-vault-token-review
  apiGroup: rbac.authorization.k8s.io
EOF
```

3. 为 token reviewer ServiceAccount 创建 service-account-token Secret。

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Secret
metadata:
  name: rbd-csi-vault-token-review-token
  namespace: rook-ceph
  annotations:
    kubernetes.io/service-account.name: rbd-csi-vault-token-review
type: kubernetes.io/service-account-token
EOF
```

4. 读取 reviewer JWT、cluster CA 和 API endpoint。

```
SA_JWT_TOKEN=$(kubectl -n rook-ceph get secret rbd-csi-vault-token-revi
ew-token \
  -o jsonpath="{.data['token']}" | base64 --decode; echo)
SA_CA_CRT=$(kubectl -n rook-ceph get secret rbd-csi-vault-token-review-
token \
  -o jsonpath="{.data['ca\.crt']}" | base64 --decode; echo)
ACP_HOST=$(kubectl config view --minify --flatten -o jsonpath="{.cluste
rs[0].cluster.server}")
```

5. 在 Vault 中配置 Kubernetes 认证。

```
vault auth enable kubernetes
vault write auth/kubernetes/config \
  token_reviewer_jwt="$SA_JWT_TOKEN" \
  kubernetes_host="$ACP_HOST" \
  kubernetes_ca_cert="$SA_CA_CRT"
```

6. 为 tenant namespace 创建 Vault role。

```
vault write "auth/kubernetes/role/csi-kubernetes" \
  bound_service_account_names="ceph-csi-vault-sa" \
  bound_service_account_namespaces="<tenant_namespace>" \
  policies="<policy_name_in_vault>"
```

NOTE

- `csi-kubernetes` 是 ACP 分布式存储在 Vault 中查找的默认 role 名称。
- ACP 分布式存储集群中 tenant namespace 里的默认 ServiceAccount 名称是 `ceph-csi-vault-sa`。
- 可以通过在 tenant namespace 中创建 ConfigMap 来覆盖这些默认值。

7. 使用 `encryptionKMSType: vaulttenantsa` 创建或更新 KMS 连接条目。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-kms-connection-details
  namespace: rook-ceph
data:
  vault-tenant-sa-kms: |
    {
       "encryptionKMSType": "vaulttenantsa",
       "vaultAddress": "<https://hostname_or_ip_of_vault_server:port>",
       "vaultTLSServerName": "<vault TLS server name>",
       "vaultAuthPath": "/v1/auth/kubernetes/login",
       "vaultAuthNamespace": "<vault auth namespace name>",
       "vaultNamespace": "<vault namespace name>",
       "vaultBackendPath": "<vault backend path name>",
       "vaultCAFromSecret": "<secret containing CA cert>",
       "vaultClientCertFromSecret": "<secret containing client cert>",
       "vaultClientCertKeyFromSecret": "<secret containing client priva
te key>",
       "tenantSAName": "<service account name in the tenant namespace>"
    }
```

参数

参数	说明	必需
`encryptionKMSType`	设置为 vaulttenantsa，以使用 ServiceAccount 进行 Vault 认证。	是
`vaultAddress`	带端口号的 Vault server 主机名或 IP 地址。	是

参数	说明	必需
`vaultBackendPath`	Vault 中用于存储加密密钥的后端路径。	否
`vaultTLSServerName`	用于验证 Vault 证书主机名的 TLS server name。	否
`vaultAuthPath`	在 Vault 中启用 kubernetes 认证方式的路径。默认路径是 `kubernetes`。如果认证方式启用在不同于 `kubernetes` 的路径下，则此变量需要设置为 `/v1/auth/<path>/login`。	否
`vaultAuthNamespace`	启用 kubernetes 认证方式的 Vault namespace。	否
`vaultNamespace`	存放所用密钥的后端路径所在的 Vault namespace。	否
`vaultCAFromSecret`	CSI 使用的 Vault CA 证书所在 Secret。	否
`vaultClientCertFromSecret`	用于与 Vault 进行 mTLS 的客户端证书所在 Secret。	否
`vaultClientCertKeyFromSecret`	与 `vaultClientCertFromSecret` 匹配的私钥所在 Secret。	否
`vaultRole`	绑定到 tenant ServiceAccount 和 policy 的 Vault Kubernetes auth role。	否
`tenantSAName`	工作负载使用的 tenant ServiceAccount 名称，通常为 `ceph-csi-vault-sa`。	否

验证步骤

1. 验证 ServiceAccount 创建：

```
kubectl -n <tenant_namespace> get sa ceph-csi-vault-sa
kubectl -n rook-ceph get sa rbd-csi-vault-token-review
```

2. 验证 token review 权限：

```
kubectl auth can-i create tokenreviews.authentication.k8s.io \
  --as=system:serviceaccount:rook-ceph:rbd-csi-vault-token-review
```

3. 验证 `ConfigMap` 条目，并确认 `encryptionKMSType: vaulttenantsa` 和 `tenantSAName`。
4. 在 `<tenant_namespace>` 中创建一个测试加密 PVC，并验证其能够成功绑定。

使用 Thales CipherTrust Manager (`kmip`) 配置访问

前提条件

- 您可以访问已启用 KMIP 的 Thales CipherTrust Manager。
- 您拥有 KMIP endpoint（`<kmip_address>:5696`）、客户端证书链和 key unique identifier（UID）。
- 您有权限在 `rook-ceph` 中创建 Secret 和 ConfigMap。

操作步骤

1. 在 Thales CipherTrust Manager 中，创建一个 KMIP client profile 并注册客户端。
2. 从 profile 中下载客户端证书、私钥和 CA 证书。
3. 为 KMIP 客户端凭证创建一个 Secret。

```
kubectl -n rook-ceph create secret generic thales-kmip-client \
  --from-file=client.crt=<client_crt_path> \
  --from-file=client.key=<client_key_path> \
  --from-file=ca.crt=<ca_crt_path>
```

4. 创建或更新 KMS 连接 `ConfigMap` 条目。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-kms-connection-details
  namespace: rook-ceph
data:
  thales-kmip-kms: |
    {
      "encryptionKMSType": "kmip",
      "kmipAddress": "<kmip_address>",
      "kmipPort": 5696,
      "kmipCAPath": "/etc/ceph-csi-kms-config/ca.crt",
      "keyIdentifier": "<thales_key_uid>"
    }
```

验证步骤

1. 验证 KMIP 凭证 Secret：

```
kubectl -n rook-ceph get secret thales-kmip-client
```

2. 验证 `ConfigMap` 条目，并确认 `encryptionKMSType: kmip`。

3. 确认 KMS 条目中的 key UID 与 Thales CipherTrust Manager 中生成的 key 一致。

> **NOTE**
>
> 对于 `kmip`，关键区别在于：加密密钥通过 KMIP key UID 和 mutual TLS 信任链进行解析，而不是通过 Vault token。

为持久卷加密创建 StorageClass

完成 KMS 访问配置后，创建带有加密参数的 StorageClass。

前提条件

- 您有权限创建 StorageClass。
- 您知道目标 KMS 配置 ID（`<kms_config_id>`）。

操作步骤

1. 进入 **Administrator**。
2. 在左侧导航栏中，点击 **Storage** > **StorageClasses**。
3. 点击 **Create Storage Class**。
   注意：以下内容是表单形式的示例，您也可以选择 YAML 完成该操作。
4. 选择 **CephRBD Block Storage**，然后点击 **Next**。
5. 切换到 YAML 视图
6. 在参数中启用加密：
   - `encrypted: "true"`
   - `encryptionKMSID: "<kms_config_id>"`
7. 点击 **Create**。

验证步骤

1. 验证 StorageClass 参数：

```
kubectl get sc encrypted-rbd-sc -o yaml
```

2. 确认存在 `encrypted: "true"` 和预期的 `encryptionKMSID`。
3. 使用此 StorageClass 创建一个测试 PVC，并确认其进入 `Bound` 状态：

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: encrypted-rbd-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: encrypted-rbd-sc
```

```
kubectl apply -f <encrypted-rbd-pvc.yaml>
kubectl get pvc encrypted-rbd-pvc
```

使用 **tenant ConfigMap** 覆盖 **Vault** 连接详情

可以通过在 ACP namespace 中创建一个 ConfigMap 来按 tenant 重新配置 Vault 连接详情，该 ConfigMap 中的配置选项与 `rook-ceph` namespace 中 `csi-kms-connection-details` ConfigMap 的值不同。该 ConfigMap 需要位于 tenant namespace 中。tenant namespace 中 ConfigMap 的值将覆盖该 namespace 中已创建的加密 Persistent Volume 所使用的 `csi-kms-connection-details` ConfigMap 值。

操作步骤

1. 在 ACP 控制台中，导航到 **Alauda Container Platform** -> **Configuration** -> **ConfigMaps**。
2. 切换到目标 tenant project/namespace。
3. 点击 **Create ConfigMap**，并将名称设置为 `ceph-csi-kms-config`。
4. 添加 tenant 特定的 Vault 覆盖值：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ceph-csi-kms-config
  namespace: <tenant_namespace>
data:
  vaultAddress: "<vault_address:port>"
  vaultBackendPath: "<backend_path>"
  vaultTLSServerName: "<vault_tls_server_name>"
  vaultNamespace: "<vault_namespace>"
```

5. 点击 **Create**。

验证步骤

1. 验证 tenant 覆盖 ConfigMap：

```
kubectl -n <tenant_namespace> get configmap ceph-csi-kms-config -o yaml
```

2. 在同一 namespace 中创建一个新的加密 PVC，并确认其进入 `Bound`。
3. 如果密钥访问失败，请检查 namespace 事件和 CSI provisioner 日志中的 Vault endpoint 和 auth-path 是否不匹配。

启用和禁用密钥轮转（可选）

密钥轮转支持对加密 PV 使用的加密密钥进行周期性刷新。

前提条件

- 请确保以下两个镜像已上传到平台私有镜像仓库：
  - `quay.io/csiaddons/k8s-controller:v0.14.0` -> `<registry>/csiaddons/k8s-controller:v0.14.0`
  - `quay.io/csiaddons/k8s-sidecar:v0.14.0` -> `<registry>/csiaddons/k8s-sidecar:v0.14.0`

为卷复制设置环境

1. 设置 **CsiAddons Controller**

   在 Control 节点上执行以下命令：

   ```
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.14.0/deploy/controller/crds.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.14.0/deploy/controller/setup-controller.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.14.0/deploy/controller/rbac.yaml
   kubectl create -f https://raw.githubusercontent.com/csi-addons/kubernetes-csi-addons/v0.14.0/deploy/controller/csi-addons-config.yaml

   kubectl -n csi-addons-system set image deployment/csi-addons-controller-manager manager=<registry>/csiaddons/k8s-controller:v0.14.0
   ```

   参数：

   - `<registry>`：平台的镜像仓库地址。

2. 启用 **CsiAddons sidecar**

   在 Control 节点上执行以下命令：

   ```
   kubectl patch cm rook-ceph-operator-config -n rook-ceph --type json --patch \
   '[
     {
       "op": "add",
       "path": "/data/CSI_ENABLE_CSIADDONS",
       "value": "true"
     },
     {
       "op": "add",
       "path": "/data/ROOK_CSIADDONS_IMAGE",
       "value": "<registry>/csiaddons/k8s-sidecar:v0.14.0"
     }
   ]'
   ```

   等待所有 csi pod 成功重启

```
kubectl get po -n rook-ceph -w | grep csi
```

（可选）强制密钥轮转优先级

在密钥轮转中，优先级指系统检查计划注解的顺序。默认优先级设置为 storage class（推荐）。这意味着系统只读取 storage class 中的注解。

但是，如果您希望系统先检查 persistent volume claim（PVC），再回退到 storage class，可以在 CSI-addons ConfigMap 中将 `schedule-precedence` 设置为 PVC 来配置此行为。

您可以按如下方式定义 csi-addons 的 ConfigMap：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-addons-config
  namespace: rook-ceph
data:
  schedule-precedence: "pvc"
```

重新启动 csi-addons operator pod 以使更改生效：

```
kubectl apply -f <csi-addons-config.yaml>
kubectl -n rook-ceph delete pod -l app.kubernetes.io/name=csi-addons
```

启用密钥轮转

要启用密钥轮转，请将注解 `keyrotation.csiaddons.openshift.io/schedule: <value>` 添加到 PersistentVolumeClaim、Namespace 或 StorageClass 上（优先级依次降低）。

`<value>` 可以是 `@hourly`、`@daily`、`@weekly`、`@monthly` 或 `@yearly`。如果 `<value>` 为空，则默认值为 `@weekly`。以下示例使用 `@weekly`。

为 **Namespace** 添加注解

```
kubectl annotate namespace <tenant_namespace> \
  "keyrotation.csiaddons.openshift.io/schedule=@weekly"
```

为 **StorageClass** 添加注解

```
kubectl annotate storageclass encrypted-rbd-sc \
  "keyrotation.csiaddons.openshift.io/schedule=@weekly"
```

为 **PersistentVolumeClaims** 添加注解

```
kubectl annotate pvc <encrypted_pvc_name> \
  "keyrotation.csiaddons.openshift.io/schedule=@weekly"
```

禁用密钥轮转

您可以针对以下对象禁用密钥轮转：

- 某个 storage class 下的所有 persistent volume claims（PVCs）
- 某个特定 PVC

为某个 **storage class** 下的所有 **PVC** 禁用密钥轮转

```
kubectl annotate storageclass encrypted-rbd-sc \
  "keyrotation.csiaddons.openshift.io/enable=false" --overwrite
```

为某个特定 **persistent volume claim** 禁用密钥轮转

1. 找出您要禁用密钥轮转的 PVC 对应的 `EncryptionKeyRotationCronJob` CR：

   ```
   kubectl get encryptionkeyrotationcronjob -o jsonpath='{range .items[?
   (@.spec.jobTemplate.spec.target.persistentVolumeClaim=="<PVC_NAME>")]}
   {.metadata.name}{"\n"}{end}'
   ```

   其中 `<PVC_NAME>` 是您要禁用的 PVC 名称。

2. 对上一步中的 `EncryptionKeyRotationCronJob` CR 执行以下操作以禁用密钥轮转：

- 将 csiaddons.openshift.io/state 注解从 managed 更新为 unmanaged：

```
kubectl annotate encryptionkeyrotationcronjob <encryptionkeyrotationcronjob_name> "csiaddons.openshift.io/state=unmanaged" --overwrite=true
```

其中 `<encryptionkeyrotationcronjob_name>` 是 `EncryptionKeyRotationCronJob` CR 的名称。

- 在 spec 字段下添加 `suspend: true`：

```
kubectl patch encryptionkeyrotationcronjob <encryptionkeyrotationcronjob_name> -p '{"spec": {"suspend": true}}' --type=merge
```

3. 保存并退出。该 PVC 的密钥轮转将被禁用。

配置 OSD WAL 和 DB 分区

目录

简介

本文介绍如何为基于主机的 Rook-Ceph Object Storage Daemons (OSDs) 配置一个高速元数据分区。在此配置中，OSD 将用户数据存储在数据设备上，并将 BlueStore 元数据（例如 RocksDB 数据库和写前日志（WAL））存储在更快的设备或分区上。

当基于 HDD 的 OSD 需要更低的元数据操作延迟，并且存储节点上有专用于 OSD 元数据的 SSD 或 NVMe 设备时，请使用此操作步骤。

场景

- 使用 SSD 或 NVMe 设备作为节点上所有所选 OSD 数据设备共享的元数据设备。
- 为某一个特定的 OSD 数据设备使用一个元数据分区。
- 为新创建或重新创建的基于主机的 OSD 规划元数据分区容量。

前提条件

开始之前，请确保满足以下条件：

- 您对集群具有 `cluster-admin` 访问权限。
- 您可以通过 shell 访问每个需要检查或准备本地设备的存储节点。
- OSD 数据设备和元数据设备或分区不包含已挂载的文件系统或应用数据。
- `rook-ceph-tools` Deployment 在 `rook-ceph` 命名空间中可用，或者您可以被允许临时启动它。

限制和约束

> **WARNING**
>
> 更改 `metadataDevice`、`databaseSizeMB` 或 `walSizeMB` 不会就地迁移现有 OSD 的 WAL 或 DB。要更改现有 OSD 的 WAL/DB 布局，请在确认集群具有足够容量且 placement groups 运行正常后，移除并重新创建该 OSD。

- 尽可能使用稳定的设备路径，例如 `/dev/disk/by-id/...`。像 `/dev/sdb` 这样的 Linux 名称在重启或硬件更换后可能会发生变化。
- 当 `metadataDevice` 配置在 `spec.storage.config` 或 `spec.storage.nodes[].config` 下时，Alauda Build of Rook-Ceph 会在同一节点上的

OSD 之间共享该元数据设备，并使用 `lvm batch` 初始化 OSD。在此模式下，不要将分区路径作为 `metadataDevice`。

- 当 `metadataDevice` 配置在特定的 `devices[].config` 下时，Alauda Build of Rook-Ceph 会使用 `lvm prepare` 初始化 OSD。在此模式下，您可以将分区路径用作该特定 OSD 的元数据设备。

规划 WAL 和 DB 容量

BlueStore 可以使用独立的 `block.db` 设备来存放 RocksDB 元数据。如果配置了 DB 设备但未显式配置 WAL 设备，BlueStore 会将 WAL 与 DB 放置在同一设备上。因此，当您为一个 HDD OSD 分配一个专用元数据分区时，应将该分区大小规划为该 OSD 的 DB 和 WAL 容量。对于这种模式，通常不需要在 `CephCluster` CR 中设置 `databaseSizeMB` 或 `walSizeMB`。

请使用以下指导原则为每个元数据分区规划容量：

工作负载类型	每个 **HDD OSD** 建议的元数据分区大小
以 RBD 为主的块存储	HDD 数据设备容量的 1% 到 2%
RGW 对象存储或大量 omap 使用	至少为 HDD 数据设备容量的 4%。如果高速设备容量允许，可使用更大的分区。
混合或不确定的工作负载	以 HDD 数据设备容量的 4% 作为起始基线。如果工作负载可能创建大量 omap key 或小对象，请增大容量。

示例：

HDD 容量	以 **RBD** 为主的元数据分区	**RGW** 最小元数据分区
4 TB	40 GB 到 80 GB	160 GB 或更大
8 TB	80 GB 到 160 GB	320 GB 或更大
12 TB	120 GB 到 240 GB	480 GB 或更大

将表中的 RGW 值视为最小规划基线，而不是上限。如果高速设备容量足够，请分配超过 4% 的容量，以降低 BlueStore 元数据回落到 HDD 的可能性。

当整个高速设备在节点级别被多个 HDD OSD 共享时，所需的高速设备总容量可按以下公式计算：

```
每个 OSD 的元数据容量 x 节点上的 HDD OSD 数量
```

如果共享的元数据设备小于计算出的容量，请减少共享该高速设备的 HDD OSD 数量，或者根据工作负载为每个 OSD 设定更小的目标值。避免依赖非常小的 DB 设备，因为当 DB 设备满时，BlueStore 元数据可能会回落到 HDD。

操作步骤

1 检查主机设备路径

登录到每个存储节点并列出可用的块设备。

```
lsblk -o NAME,PATH,SIZE,TYPE,FSTYPE,MOUNTPOINT,MODEL
```

列出稳定的设备链接。

```
ls -l /dev/disk/by-id/
```

记录每个 OSD 的数据设备和元数据设备路径。例如：

目的	示例路径
OSD 数据设备	`/dev/disk/by-id/ata-ST4000DM004-XXXX`
元数据分区	`/dev/disk/by-id/nvme-SAMSUNG_MZVL21T0-YYYY-part1`

2 为一个 OSD 配置一个元数据分区

当元数据目标是分区时，请使用设备级别的 `metadataDevice`。此模式会将每个 OSD 数据设备映射到其各自的 WAL/DB 分区。分区容量即为该 OSD 的元数据容量，因此在这种配置下不需要 `databaseSizeMB` 和 `walSizeMB`。

```
kubectl -n rook-ceph edit cephcluster ceph-cluster
```

按如下示例更新 `spec.storage` 配置。

```
spec:
  storage:
    useAllNodes: false
    useAllDevices: false
    nodes:
    - name: "worker-1"
      devices:
      - name: "/dev/disk/by-id/ata-ST4000DM004-XXXX"
        config:
          metadataDevice: "/dev/disk/by-id/nvme-SAMSUNG_MZVL21T0-YYYY
-part1"
      - name: "/dev/disk/by-id/ata-ST4000DM004-ZZZZ"
        config:
          metadataDevice: "/dev/disk/by-id/nvme-SAMSUNG_MZVL21T0-YYYY
-part2"
```

在此示例中，每个 HDD 数据设备都使用不同的 NVMe 分区作为其 BlueStore DB。WAL 与 DB 共同放置在同一个元数据分区上。

3 为节点上的 OSD 配置共享元数据设备

仅当元数据目标是整个设备或逻辑卷时，才使用节点级别的 `metadataDevice`。Rook 会在该节点所选的 OSD 数据设备之间共享元数据目标。只有在您需要限制 Rook 从共享元数据设备为每个 OSD 分配的 DB 大小时，才设置 `databaseSizeMB`。在大多数部署中，不要设置 `walSizeMB`；WAL 可以与 DB 共同放置。

```
spec:
  storage:
    useAllNodes: false
    useAllDevices: false
    nodes:
    - name: "worker-1"
      config:
        metadataDevice: "/dev/disk/by-id/nvme-SAMSUNG_MZVL21T0-YYYY"
        databaseSizeMB: "49152"
      devices:
      - name: "/dev/disk/by-id/ata-ST4000DM004-XXXX"
      - name: "/dev/disk/by-id/ata-ST4000DM004-ZZZZ"
```

当整个高速设备被保留给节点上的 OSD 元数据时，请使用此模式。

4 等待 **OSD prepare** 作业完成

监视 OSD prepare 作业和 OSD pod。

```
kubectl -n rook-ceph get jobs,pods -l app=rook-ceph-osd-prepare
kubectl -n rook-ceph get pods -l app=rook-ceph-osd
```

如果某个 OSD prepare 作业失败，请检查该作业日志。

```
kubectl -n rook-ceph logs job/rook-ceph-osd-prepare-<node-name>
```

验证

如果 tools pod 未运行，请先启动它。

```
kubectl -n rook-ceph scale deploy rook-ceph-tools --replicas=1
```

检查集群健康状态。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

确认 OSD 已启动并分配到预期的主机。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd tree
```

如需检查本地 BlueStore 元数据布局，请在 OSD pod 中或在准备该 OSD 的存储节点上运行 `ceph-volume`。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-osd-<osd-id> -- ceph-volum
e lvm list
```

在输出中，确认该 OSD 具有 `block.db`，并且在 Ceph 单独配置时，还具有指向预期元数据设备或分区的 `block.wal` 条目。

配置 Multus 多 NIC 网络

本文介绍如何使用 Multus 将 ACP 分布式存储（Ceph）连接到专用容器网络，从而将 Ceph **public** 和 **cluster** 流量分离到不同的网络接口（NIC）上。

目录

概述

默认情况下，ACP Ceph 存储集群使用主机网络（`provider: host`）。在 Web 控制台中创建存储集群时，也可以选择 **Container Network**，将 Ceph 流量放到单一容器网络中（分配给 `rook-ceph` 命名空间的一个 `NetworkAttachmentDefinition`）。有关基础单网络选项，请参见 创建标准存储服务。

Multus 多 NIC 网络在容器网络模型的基础上进行了扩展，使你可以将两个 Ceph 网络角色映射到不同的网络上：

- **public network** — Ceph 客户端（CSI、应用）与集群之间的客户端流量。
- **cluster network** — OSD 之间的内部流量，例如复制和恢复。

将这些角色分离到不同的 NIC 上可以提高隔离性，并避免大量的复制/恢复流量与客户端 I/O 争用。

> **NOTE**
>
> 所有到 **OSD** 和来自 **OSD** 的流量都使用 **Multus** 网络——这就是大部分存储数据路径：RBD 客户端 I/O、CephFS 数据和元数据（MDS）、对象数据（RGW 在 OSD 上读写对象），以及 OSD 复制。通过 **Kubernetes Service** 到达的端点——**monitors**（客户端使用 mon Service 获取集群映射）、**manager** 以及 **RGW S3 endpoint**——都通过默认网络提供服务；这些 daemon 即使已为其 Pod 挂载了 Multus 接口，仍会保持其面向 Service 的监听端口在默认网络上。这是 Rook 的行为，不是配置错误。因此，Multus 隔离的是存储数据路径，而不是由 Service 承载的控制面和入口端点。

两个管理界面

ACP 提供旧版和新版 Web 控制台，它们通过不同的资源管理 Ceph 存储。请在管理你的集群的资源上配置网络：

Web 控制台	管理资源	网络配置位置
新版 Web 控制台	`StorageCluster`（由 storage operator 管理）	`StorageCluster.spec.ceph.network` — operator 会将其映射到底层的 `CephCluster.spec.network`

Web 控制台	管理资源	网络配置位置
旧版 Web 控制台	`CephCluster`（Rook，直接管理）	`CephCluster.spec.network`

要确定某个集群由哪个资源管理，请检查是否存在 `StorageCluster`：

```
kubectl get storagecluster -A
```

- 如果你的集群存在 `StorageCluster`，则由新版控制台管理——请在 **StorageCluster** 上配置网络（operator 会撤销对底层 `CephCluster` 的手动编辑）。
- 如果不存在 `StorageCluster`，则该集群由旧版控制台直接作为 **CephCluster** 管理——请在 `CephCluster` 上配置网络。

这两个资源暴露相同的网络字段（`provider`、`selectors`、`addressRanges`）。以下各节分别给出了两个管理界面的 manifest。

网络角色到网络的映射通过 `selectors` 声明：

Selector key	**Ceph** 角色	后端网络
`public`	Ceph public network	通过 `<namespace>/<nad-name>` 引用的 `NetworkAttachmentDefinition`（NAD）
`cluster`	Ceph cluster（复制）网络	通过 `<namespace>/<nad-name>` 引用的 `NetworkAttachmentDefinition`（NAD）

你可以只指定 `public`，也可以同时指定 `public` 和 `cluster`。当 `provider` 为 `multus` 时，至少需要一个 selector。

限制和前提条件

在配置 Multus 多 NIC 网络之前，请注意以下限制：

- 仅支持创建时配置

  - 必须在创建存储集群时配置 Multus 多 NIC 网络。不支持在运行中的集群上更改网络 provider：Ceph 必须对所有 monitor 执行故障切换才能应用新的网络模型，而且 API 不允许将一个非空 provider 直接切换到另一个非空 provider（必须先将 provider 恢复为空值）。请将网络模型视为集群的固定属性。

- 已安装 **Multus CNI plugin**

  - 集群必须使用 Kube-OVN 作为网络 plugin，并且必须安装 Multus CNI plugin。这是一次性的、集群级别的设置。安装方法：进入 **Administrator** > **Marketplace** > **Cluster Plugins**，搜索 `multus`，然后安装 **Alauda Container Platform Networking for Multus**。详情请参见 配置 Kube-OVN 网络以支持 Pod 多网络接口。

- 单一 **IP family**

  - Multus 网络必须使用单一 IP family。支持 IPv4 和 IPv6，但 Multus 不支持双栈（IPv4 和 IPv6 同时使用）。如需使用 IPv6，请在 network block 中设置 `ipFamily: IPv6`。
  - 这与 创建标准存储服务 中的基础 **Container Network** 选项不同，后者不支持 IPv6。该限制仅适用于单网络表单选项；本文描述的 Multus 多 NIC 操作步骤支持 IPv6。

> **WARNING**
>
> Multus 会将存储数据路径（OSD 流量）迁移到专用 NIC，但不会改变工作负载使用存储的方式：应用 **Pod** 不需要加入 **Multus** 网络。 客户端通过默认网络（经由 mon Service）访问 monitor，而在每个 node 上以 host networking 运行的 Ceph CSI driver 负责到 OSD 的数据路径。因此，真正的要求是存储 **node** 能够路由到 **underlay network**，而不是工作负载 Pod 必须挂载到该网络。请在部署前规划好网络角色、NAD 和 node 路由。

在集群创建时配置多 NIC 网络

多 NIC 设置不能在向导的表单字段中表达，但 **Create Cluster** 向导提供了 **YAML mode**。推荐的流程是：先像平常一样完成向导，让其自动填充 manifest，然后切换到 YAML mode，仅编辑网络部分，最后在 YAML mode 中提交。这样可以避免手工编写整个 manifest。

YAML mode 中显示的 manifest 取决于你的控制台（参见 两个管理界面）：新版控制台显示 `StorageCluster`，旧版控制台显示 `CephCluster`。请编辑对应路径上的 network block。

1 创建 Multus 网络

若要在接近主机速度的条件下将存储流量放到专用物理 NIC 上，请使用 **Kube-OVN Underlay** 网络，而不是 overlay 网络。Underlay 会将专用物理 NIC 直接桥接到存储网络，不产生封装开销。

Underlay 需要先准备物理网络：每个 storage node 都需要一个专用 NIC（不承载 SSH 等其他流量），并且必须配置上游交换机端口/VLAN 和网关。然后创建一个 **bridge network**（`ProviderNetwork`）和一个 **VLAN**（`Vlan`），让存储子网挂载到其上。有关物理前提条件以及如何创建 bridge network 和 VLAN，请参见 准备 Kube-OVN Underlay 物理网络 和 配置子网。

对于要隔离的每个 Ceph 网络角色，在 `rook-ceph` 命名空间中创建一个 `NetworkAttachmentDefinition` 和一个对应的 underlay 子网。下面的示例创建 **public** 网络；如果你也想隔离复制流量，请使用不同的名称以及不同的 CIDR/VLAN 再创建 **cluster** 网络。

创建 `NetworkAttachmentDefinition`。`provider` 字段必须使用 `<name>.<namespace>.ovn` 格式：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ceph-public-net
  namespace: rook-ceph
spec:
  config: |-
    {
      "cniVersion": "0.3.0",
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "ceph-public-net.rook-ceph.ovn"
    }
```

创建为该网络提供地址的 underlay Kube-OVN subnet。将 `vlan` 设置为你准备好的 `Vlan`，使用与物理网络匹配的 `cidrBlock` / `gateway`，并让 `provider` 与 `NetworkAttachmentDefinition` 中的值一致：

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: ceph-public-net
spec:
  protocol: IPv4
  cidrBlock: 192.168.10.0/24
  gateway: 192.168.10.1
  vlan: ceph-public-vlan
  provider: ceph-public-net.rook-ceph.ovn
```

应用这两个资源，然后确认 `NetworkAttachmentDefinition` 已存在：

```bash
kubectl get net-attach-def -n rook-ceph
```

在后续步骤中设置的 `selectors` 会以 `<namespace>/<name>` 形式引用这些 NAD，例如 `rook-ceph/ceph-public-net`。

2 打开 **Create Cluster** 向导

按照 创建标准存储服务 中的说明开始创建分布式存储集群。在 **Create Cluster** 步骤中，像平常一样配置 device classes、storage devices 和 component placement，这样向导会填充其余 manifest。

3 切换到 **YAML mode**

将向导从表单模式切换到 **YAML** mode。manifest 已根据你的选择自动填充。

4 编辑网络部分

在 manifest 中添加或替换 network block。设置 `provider: multus`，并配置将 Ceph public/cluster 角色映射到你的 NAD 的 `selectors`。

- 新版 **Web** 控制台 — manifest 是 `StorageCluster`；编辑 `spec.ceph.network`：

```
spec:
  ceph:
    network:
      provider: multus
      selectors:
        public: rook-ceph/ceph-public-net
        cluster: rook-ceph/ceph-cluster-net
```

- 旧版 **Web** 控制台 — manifest 是 `CephCluster`；编辑 `spec.network`（字段相同，但没有 `spec.ceph` 包装层）：

```
spec:
  network:
    provider: multus
    selectors:
      public: rook-ceph/ceph-public-net
      cluster: rook-ceph/ceph-cluster-net
```

如果你只需要将客户端流量迁移到专用网络，请仅保留 `public` selector，省略 `cluster`。

使用与 `selectors` 同级的 `addressRanges` block 明确设置 CIDR 范围。Rook 会尝试自动检测已挂载网络的 CIDR，但在 Kube-OVN underlay 网络上此检测并不可靠，可能会失败，使集群卡在“Configuring Ceph Mons”。指定 `addressRanges` 可以避免此问题：

```
addressRanges:
  public:
    - 10.176.0.0/16
  cluster:
    - 10.176.0.0/16
```

使用你创建的 underlay 子网的 CIDR（如果没有将 cluster network 分离出来，只需要 `public`）。

TIP

如果 `StorageCluster` 一直处于 `Progressing`，并且底层 `CephCluster` 报告 `failed to discover network CIDRs for multus`，请添加或修正 `addressRanges`。

5 从 YAML mode 提交

保持在 YAML mode 下提交向导（点击 **Create Cluster**）。在提交之前不要切回表单模式，因为那样可能不会保留网络字段。

> **WARNING**
>
> 在新版控制台中，请在这里的 `StorageCluster` 中配置网络，而不要在之后去编辑 `CephCluster`。当集群由 `StorageCluster` 管理时，`spec.network` 是一个受管字段，对 `CephCluster` 的手动编辑会被 operator 恢复。

验证

存储集群创建完成后，请分别从 ACP 侧和 Ceph 侧验证网络配置。

（新版控制台）检查 StorageCluster 网络设置

如果你的集群由 `StorageCluster` 管理，请确认所需配置。Ceph `StorageCluster` 在 `rook-ceph` 命名空间中的名称始终为 `ceph-cluster`（这两项都由 admission 强制保证）：

```
kubectl get storagecluster ceph-cluster -n rook-ceph -o yaml
```

输出应包含预期的 provider 和 selectors：

```
spec:
  ceph:
    network:
      provider: multus
      selectors:
        public: rook-ceph/ceph-public-net
        cluster: rook-ceph/ceph-cluster-net
```

检查实际生效的 CephCluster 网络设置

此检查适用于两个控制台，因为 `CephCluster` 是 Rook 实际作用的资源：

```
kubectl get cephcluster ceph-cluster -n rook-ceph -o jsonpath='{.spec.network}' | jq
```

`provider` 和 `selectors` 应与配置一致。在新版控制台中，它们还应与管理该集群的 `StorageCluster` 一致。

检查 Ceph Pod 是否已挂载到网络

确认 Ceph daemon Pod（例如 MON 和 OSD）带有 Multus 网络挂载注解，并报告了预期的接口：

```
kubectl get pod -n rook-ceph -l app=rook-ceph-osd \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/networks}{"\n"}{end}'
```

每个 OSD Pod 都应引用 public 网络，以及（如果已配置）cluster 网络。

检查工作负载挂载

创建或重启一个测试工作负载，使其挂载 CephFS PVC 和 RBD PVC，然后验证：

- Pod 能成功启动
- 文件系统可以正常读写

- CSI 或工作负载日志中没有出现与挂载相关的错误

故障排查建议

如果启用 Multus 后存储集群未能变为健康状态，请先检查以下项目：

1. **NAD** 不存在 — selector 引用的 `NetworkAttachmentDefinition` 在 `rook-ceph` 命名空间中不存在，或者 `<namespace>/<nad-name>` 引用拼写错误。
2. 缺少 **selector** — 已设置 `provider: multus`，但未定义 `public` 或 `cluster` selector。至少需要一个 selector。
3. **CIDR** 检测失败 — Rook 无法自动检测网络范围。请显式指定 `addressRanges`。
4. **IP** 不足 — NAD 背后的 IPAM 池（例如已配置的 CIDR 或 address range）没有可供新 Ceph Pod 使用的空闲地址。
5. **node** 无法访问 **underlay** — Ceph CSI driver 以 host networking 运行，因此每个 **node**（而不是工作负载 Pod）都必须能够路由到 underlay 网络上的 OSD 地址。如果挂载卡住，请检查 node 的 Kube-OVN underlay 路由，以及 node 是否可以访问 OSD 地址。

如果集群无法恢复，请使用修正后的网络配置重新创建。由于无法在运行中的集群上切换网络模型，网络配置错误通常需要通过重新创建存储集群来解决，而不是就地 patch。

MinIO 到 Rook Ceph RGW 数据迁移指南

目录

1. 概述与架构

本文档说明如何使用已安装的 **Alauda Build of VolSync** Operator 镜像中内置的 Rclone 组件，将 Kubernetes 中高可用（HA）MinIO 部署的全部数据迁移到 Rook Ceph RGW。

1.1 架构说明

- 安全镜像复用：此方案会调度一个 Kubernetes Job，复用经过安全补丁加固的 `build-harbor.alauda.cn/acp/volsync` Operator 镜像，覆盖默认入口点，并直接调用内部的 Rclone 二进制文件，执行标准的 **S3-to-S3** API 级对象同步。

1.2 同步策略

采用三阶段策略：**“Full -> Incremental -> Cutover”**：

1. 初始同步（**Full Sync**）：在迁移历史数据的同时保持服务在线。
2. 增量同步（**Incremental Sync**）：保持服务在线，并多次运行以补齐新增或修改的数据。
3. 切换同步（**Cutover Sync**）：停止写入，执行严格的一致性检查，并完成最终切换。

---

2. 前提条件与准备

2.1 验证 Alauda VolSync 安装并提取版本（关键前提）

在执行此方案之前，请确保 **“Alauda Build of VolSync”** Operator 已安装在 `volsync-system` 命名空间中。迁移 Job 镜像版本必须与当前运行的 Operator 版本严格匹配。

有关 **Alauda Build of VolSync** 的详细安装说明，请参见 Configure PVC DR with VolSync。

获取版本的方法： 登录集群管理控制台，进入 **Marketplace / OperatorHub**，打开 **Installed Operators**，找到 VolSync Operator，并记录显示的版本（例如，`v0.8.0` 或 `v0.9.0`）。将该值保存为 `<OPERATOR_VERSION>`，以便后续配置使用。

2.2 收集 MinIO 源端信息

- **Endpoint**：MinIO 内部 Service 地址（例如：`http://minio.tenant-ns.svc:9000`）。
- **Credentials**：在所有 bucket 上具有 `Read` 和 `List` 权限的 Access Key / Secret Key。

2.3 创建 Ceph RGW 用户（目标端）

在 Rook Ceph 集群中应用以下 YAML，创建专用的迁移用户并授予 bucket 创建权限。

```yaml
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: volsync-migration-user
  namespace: rook-ceph
spec:
  store: object-store
  displayName: "VolSync Migration Admin"
  capabilities:
    bucket: "*"
```

为了最小权限原则，仅为该迁移账号保留 bucket 级权限，不要授予 `user` 管理能力。

提取并解码 Ceph 凭据以便后续使用：

```
# Get Access Key
kubectl -n rook-ceph get secret rook-ceph-object-user-object-store-volsyn
c-migration-user -o jsonpath='{.data.AccessKey}' | base64 -d
# Get Secret Key
kubectl -n rook-ceph get secret rook-ceph-object-user-object-store-volsyn
c-migration-user -o jsonpath='{.data.SecretKey}' | base64 -d
```

---

3. 部署配置（Manifests）

建议将迁移任务部署在目标端命名空间（Ceph RGW）中，或者部署在专用的运维命名空间中。

在应用这两个 manifest 之前，先创建它们共同使用的命名空间：

```
kubectl create ns migration-ops
```

> 部署位置建议（重要）
> 在迁移过程中，Rclone 会先在本地读取数据进行处理，然后再上传到目标 S3 集群。因此，运行迁移任务的集群网络位置会直接影响带宽和完成时间。建议将 VolSync Operator/迁移 Job 部署在 **MinIO** 或 **Ceph** 集群 中，以减少跨集群网络开销并提升传输稳定性。

3.1 创建 Rclone Config Secret

将源端和目标端的 S3 凭据写入 Secret。重要：不要在 `endpoint` 或 **secret** 值周围添加引号。

```
apiVersion: v1
kind: Secret
metadata:
  name: volsync-rclone-config-secret
  namespace: migration-ops
type: Opaque
stringData:
  rclone.conf: |
    [source-minio]
    type = s3
    provider = Minio
    env_auth = false
    access_key_id = MINIO_ACCESS_KEY_HERE
    secret_access_key = MINIO_SECRET_KEY_HERE
    endpoint = MINIO_ENDPOINT_HERE
    no_check_certificate = true

    [dest-ceph]
    type = s3
    provider = Ceph
    env_auth = false
    access_key_id = CEPH_ACCESS_KEY_HERE
    secret_access_key = CEPH_SECRET_KEY_HERE
    endpoint = CEPH_ENDPOINT_HERE
    list_version = 2
```

3.2 定义迁移 Job

部署一个 Job，调用经过安全补丁加固的 Alauda VolSync 镜像来执行 S3 数据同步。

> 关于 `remote:bucket[/prefix]` 与 `remote:`（重要）
>
> - 对于 S3 后端，常见的 Rclone 模式是 `remote:bucket` 或 `remote:bucket/prefix`，这也是定义作用范围最清晰、最安全的方式。
> - 本文档故意保留 `source-minio:` -> `dest-ceph:` 的写法，以支持完整实例级迁移（源端所有可见 bucket）。
> - 风险提示：`remote:` 的作用范围更大。如果凭据权限过高，可能会迁移到目标范围之外的 bucket。请务必先在测试环境中验证，并考虑在生产切换时改为显式的

> `remote:bucket[/prefix]` 白名单模式。

请务必将下面 **YAML** 中的 `<OPERATOR_VERSION>` 替换为第 **2.1** 节中发现的实际版本。

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: volsync-s3-sync-job
  namespace: migration-ops
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: rclone
          # Use Alauda's patched volsync image. Version must exactly match OperatorHub.
          image: build-harbor.alauda.cn/acp/volsync:<OPERATOR_VERSION>
          # Note: In ACP clusters, this image reference is automatically rewritten
          # to an internal registry address after Pod creation
          # (for example: registry.alauda.cn:60070/acp/volsync:<OPERATOR_VERSION>).
          # This is expected behavior.
          # Use `kubectl describe pod <pod-name>` to check effective Image / ImageID.
          imagePullPolicy: IfNotPresent
          # Override default entrypoint and directly invoke rclone inside the image
          command: ["rclone"]
          args:
            - "sync"
            - "source-minio:"
            - "dest-ceph:"
            - "--progress"
            - "--create-empty-src-dirs"
            - "--ignore-errors"
            # --- Performance tuning flags ---
            - "--transfers=32"      # parallel file transfers
            - "--checkers=64"       # parallel checkers
            - "--s3-chunk-size=64M" # large-object chunk size to reduce RGW fragmentation
            - "--fast-list"         # use ListObjectsV2 to reduce API requests
            - "--metadata"          # synchronize object metadata
          resources:
```

```
            requests:
              cpu: "2000m"
              memory: "4Gi"
            limits:
              cpu: "4000m"
              memory: "8Gi"
          env:
            - name: RCLONE_CONFIG
              value: "/config/rclone.conf"
          volumeMounts:
            - name: config-volume
              mountPath: /config
              readOnly: true
      volumes:
        - name: config-volume
          secret:
            secretName: volsync-rclone-config-secret
```

4. 执行流程

阶段 1：初始全量同步

1. 创建运维命名空间：`kubectl create ns migration-ops`
2. 应用 Secret：`kubectl apply -f rclone-secret.yaml`
3. 启动 Job：`kubectl apply -f rclone-job.yaml`
4. 监控进度：`kubectl -n migration-ops logs -f job/volsync-s3-sync-job`

阶段 2：增量同步

为了补齐全量同步期间新生成的数据，按需重复此步骤。

1. 删除之前的 Job：`kubectl -n migration-ops delete job volsync-s3-sync-job`
2. 重新创建 Job：`kubectl apply -f rclone-job.yaml`

机制说明：默认情况下，Rclone 通过比较 Size 和 ModTime 来判断是否需要传输。已有且未修改的文件会被跳过。

阶段 3：最终切换

1. 停止写入：在应用层停止向 MinIO 写入，或者通过负载均衡器/Ingress 阻止写流量。
2. 严格同步验证：

- 删除当前 Job：`kubectl -n migration-ops delete job volsync-s3-sync-job`
- 更新 `rclone-job.yaml`：在最终切换运行中移除 `- "--ignore-errors"`，然后在 `args` 中追加 `- "--checksum"`。这样可以避免掩盖失败的对象，并在差异检测时优先使用 Size+Checksum（如果可用）。
- 一致性建议（必需）：不要仅依赖对象数量或 ETag。切换前，请运行 `rclone check source-minio: dest-ceph: --download --checksum`（或者对关键对象执行抽样下载并计算 SHA256），以减少由 multipart/ETag 差异引起的误判。
- 重新应用 Job：`kubectl apply -f rclone-job.yaml`

3. 验证并切换：在 Job 进入 `Completed` 且日志中没有错误后，将应用的 S3 Endpoint 和凭据更新为 Ceph RGW。

---

5. 故障排查与调优建议

症状	技术诊断	解决方案
ImagePullBackOff	节点无法拉取镜像，或者存在版本不匹配。ACP 集群会自动将 `build-harbor.alauda.cn` 重写为内部 registry 地址。	首先运行 `kubectl describe pod <pod-name>`，验证实际生效的 `Image`（是否已重写）和 `ImageID`。然后校验 `<OPERATOR_VERSION>`，并检查实际 registry 的网络/认证（`imagePullSecrets`）。
Pod OOMKilled	`--fast-list` 会在海量小对象场景下批量加载元数据，占用较高内存。	将 Pod 内存限制提升到 8Gi+，或者移除 `--fast-list`（这会增加 API 请求并降低遍历速度）。

症状	技术诊断	解决方案
403 Forbidden	目标 Ceph 用户没有创建 bucket 的权限。	检查 `CephObjectStoreUser` 配置，确保 `capabilities` 中包含 `bucket: "*"`.
dial tcp: lookup "http: no such host"	Secret 配置格式无效；endpoint URL 被加了引号。	编辑 Secret，并移除 **endpoint URL** 周围的引号，确保格式严格为：`endpoint = http://...`.
503 Service Unavailable	迁移并发过高，导致 Ceph RGW 写入压力过大。	降低 Job `args` 中的 `--transfers`，或者添加 `--bwlimit` 来限制传输带宽。

TopoLVM 本地存储

介绍

介绍

安装

安装

前提条件

操作步骤

操作指南

设备管理

前提条件

添加设备

监控与告警

监控

告警

实用指南

使用 Velero 备份和恢复 TopoLV

- 前提条件
- 限制
- 操作步骤

从 ACP TopoLVM 本地存储中移除

- 术语
- 选择正确的场景
- 场景 1：从设备类卷组中移除卷组设备
- 场景 2：从设备类中移除设备类卷组
- 场景 3：移除 TopoLVM 存储节点

配置条带逻辑

- 前提条件
- 操作步骤

介绍

TopoLVM 是一个专为 Kubernetes 设计的 Container Storage Interface (CSI) 插件，旨在提供高效便捷的本地存储卷管理。

主要特性和优势：

- 本地卷管理：TopoLVM 专注于管理 Kubernetes 节点上的本地存储设备（如磁盘和 SSD）。相比传统的网络存储，本地卷具有更低的延迟和更高的性能。
- 拓扑感知：TopoLVM 能够识别 Kubernetes 集群的拓扑结构（例如节点、可用区），根据 Pod 的实际调度位置自动将存储卷分配到同一节点，进一步优化性能。
- 动态卷分配：TopoLVM 支持动态创建、删除和调整存储卷大小，无需人工干预，显著简化操作并降低复杂度。
- 与 **Kubernetes** 深度集成：作为 CSI 插件，TopoLVM 无缝集成 Kubernetes 存储管理 API，使用户能够通过标准的 Kubernetes 资源对象（如 PersistentVolumeClaims）直接管理本地卷。

总之，TopoLVM 解决了 Kubernetes 使用本地存储时常见的手动管理、缺乏拓扑感知和动态分配能力不足等问题，为需要高性能本地存储的应用（如数据库和缓存）提供了更高效且用户友好的解决方案。

安装

Local storage 是一种软件定义的服务器本地存储解决方案，提供简单、易维护且高性能的本地存储服务能力。基于社区的 TopoLVM 方案，通过系统的 LVM 方式实现本地存储的持久卷编排管理。

目录

前提条件

- 存储集群的每个节点必须安装 lvm2 包。若未安装，请在节点上执行 `yum install -y lvm2` 命令。
- 下载对应您平台架构的 **Alauda Container Platform Storage Essentials** 安装包。
- 通过 Upload Packages 机制上传 **Alauda Container Platform Storage Essentials** 安装包。
- 下载对应您平台架构的 **Alauda Build of TopoLVM** 安装包。
- 通过 Upload Packages 机制上传 **Alauda Build of TopoLVM** 安装包。

操作步骤

1 部署 Alauda Container Platform Storage Essentials

1. 登录，进入 **Administrator** 页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Storage Essentials**，点击 **Install**，进入 **Install Alauda Container Platform Storage Essentials** 页面。
   配置参数：

参数	推荐配置
Channel	默认 channel 为 `stable`。
Installation Mode	`Cluster`：集群内所有命名空间共用一个 Operator 实例进行创建和管理，资源占用较低。
Installation Place	选择 `Recommended`，Namespace 仅支持 **acp-storage**。
Upgrade Strategy	`Manual`：当 Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

2 部署存储

1. 进入 **Administrator**。
2. 在左侧导航栏点击 **Storage Management** > **Local Storage**。
3. 点击 **Configure Now**。
4. 在 **Install Operator** 向导页面，点击 **Start Deployment**。
   - 页面自动跳转到下一步，表示 Operator 部署成功。
   - 若部署失败，请根据界面提示排查解决，然后点击 **Clean Up** 并重新部署 Operator。
5. 在 **Create Cluster** 向导页面，添加设备。

参数	说明
Select Node	至少有 1 块裸盘的节点。
Device Class	每个设备类对应一组具有相同特性的存储设备，建议根据磁盘性质填写名称，如 *hdd*、*ssd*。
Device Type	仅支持磁盘类型。
Storage Device	例如 */dev/sda*。若有多块磁盘，可逐一添加。
Snapshot	启用后支持创建 PVC 快照，并使用快照配置新的 PVC，实现业务数据的快速备份与恢复。 若创建存储时未启用快照，后续可在存储集群详情页的 **Operations** 中按需启用。 注意：使用前请确保当前集群已部署 Volume Snapshot Plugin。

- 点击下一步，页面自动跳转表示集群部署成功。
- 若创建失败，请根据界面提示及时清理资源。

6. 在 **Create Storage Class** 向导页面，配置相关参数。

参数	说明
Name	存储类名称，必须在当前集群内唯一。
Display Name	便于识别或筛选的名称，如存储类的中文描述。
Device Class	设备类是 TopoLVM 中对存储设备的分类方式，每个设备类对应一组具有相同特性的存储设备。无特殊需求时，可使用集群中的 **Auto-Allocated** 设备类。
File System	- **XFS** 是高性能的日志文件系统，擅长处理并行 I/O 负载，支持大文件处理并提供流畅的数据传输。 - **EXT4** 是 Linux 中的日志文件系统，采用 extent 文件存储方式，支

参数	说明
	持大文件处理。文件系统容量可达 1 EiB，最大支持文件大小为 16 TiB。
Recycling Policy	持久卷的回收策略。 - Delete：删除持久卷声明时，绑定的持久卷也被删除。 - Retain：即使持久卷声明被删除，绑定的持久卷仍然保留。
Access Mode	ReadWriteOnce (RWO)：可被单个节点以读写模式挂载。
Allocation Project	该类型的持久卷声明只能在特定项目中创建。 若暂未分配项目，后续也可进行更新。

7. 点击 **Next**，等待资源创建完成。

操作指南

设备管理

前提条件

添加设备

监控与告警

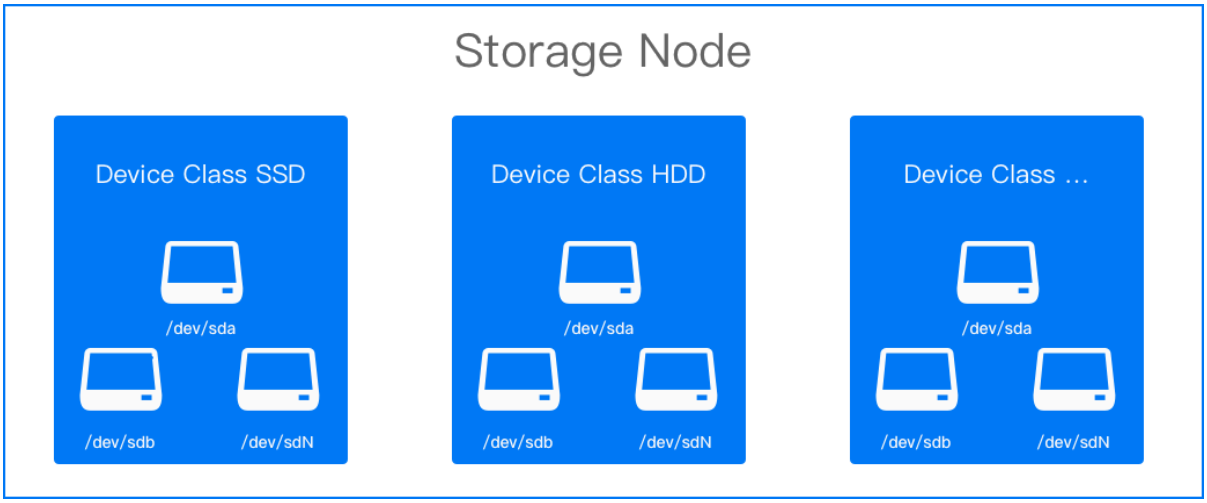
监控

告警

设备管理

无论是初次部署还是资源扩容，都需要将节点上可用的磁盘映射为存储设备以供使用和管理。

具有相似特性的存储设备通常会集中使用，这些设备在本地存储中归类于设备类（**Device Classes**）。使用设备类相当于直接使用磁盘，确保零损耗和高性能，同时减少应用对特定设备的感知和依赖。



目录

前提条件

- 创建本地存储集群时，必须至少添加了 1 个设备类 (deviceClasses.classes)，且设备类中包含设备。
- 节点上必须至少存在 1 个裸盘。

添加设备

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 本地存储。
3. 在 详情 标签页，点击 添加存储节点。
4. 按照以下说明配置相关参数。

参数	说明
存储节点	至少拥有 1 个裸盘的节点。
设备类	每个设备类对应一组具有相同特性的存储设备；建议根据磁盘性质命名，如 *hdd*、*ssd*。
存储设备	例如 */dev/sda*。如果有多个磁盘，可以逐个添加。 注意：存储设备应为整个硬盘，而非硬盘上的分区，否则会导致错误。

5. 点击 添加。
   注意：如果设备类状态因未添加设备而显示为 `Unavailable`，可以继续执行以下操作。
6. 切换到 存储设备 标签页，点击 添加存储设备。
7. 根据界面提示添加设备。
8. 点击 添加。

监控与告警

本地存储提供开箱即用的监控指标采集和告警能力。启用平台监控组件后，可基于存储集群、存储性能和存储容量配置监控和告警，并支持配置通知策略。

直观呈现的监控数据可用于支持运维巡检或性能调优的决策，完善的告警机制则有助于确保存储系统的稳定运行。

目录

监控

性能监控

平台默认采集本地存储常用的性能监控指标，如读写带宽、IOPS 和延迟。这些指标的实时监控数据可在 存储管理 下的 本地存储 页面中的 监控 标签页查看。平台通过图表直观展示这些指标，便于管理员清晰观察当前存储性能，快速识别潜在问题。

容量监控

由于本地存储只能使用节点本地可用的存储资源，用户在声明本地存储前必须确保节点上有足够的可用容量，避免因过度声明导致的问题。

为此，平台在本地存储的 详情 部分提供了按设备类型分类的详细容量监控。用户可以清晰查看以数值和图形形式展示的可用存储空间。如果某类设备显示可用容量不足，应先清理空间或添加磁盘设备后再使用本地存储。

告警

平台内置了一套默认的告警策略。当资源异常或监控数据达到告警阈值时，会自动触发告警。预配置的告警策略有效覆盖了常见的运维需求，包括集群健康状态和设备类型容量的告警。

配置通知

为确保告警能够及时接收，应在运维中心配置通知策略。通知可通过邮件、短信或其他方式发送给相关人员，促使其及时关注并解决问题或防止故障发生。用户可直接从运维中心界面访问通知策略设置。关于告警配置的详细说明，请参见 [Creating Alert Policies] 文档。

处理告警

- 当存储集群的健康状态变为 `Alert` 时，管理员应立即排查。详情 部分提供排查和解决问题的信息。常见原因包括节点服务异常或特定设备类型存在问题。

检查项	对应状态	原因说明
健康状态	Alert	由节点服务异常或设备类型问题引起。
服务状态	Unknown	节点处于 `notready` 状态，可能因网络故障或断电导致。

检查项	对应状态	原因说明
设备类型状态	Unavailable	使用的磁盘可能不是裸盘，或磁盘缺失。

- 告警 标签页触发的实时告警需要及时关注，即使存储集群当前状态显示为 `Healthy`。快速响应可防止问题升级为更严重的故障。以下表格列出了告警级别及其含义：

告警级别	含义
Critical	表示存在重大问题，导致平台服务中断或数据丢失，影响严重。
Major	已知问题，可能影响平台功能和正常业务运行。
Warning	存在潜在风险，需要及时干预以避免影响正常业务运行。

事后分析

告警历史 记录了所有已触发且当前不需立即处理的告警。事后分析时应考虑：

- 事件发生时具体观察到了哪些异常？
- 是否存在特定告警反复出现的规律？如何在未来主动预防？
- 是否在特定时间段告警激增，是否与外部因素或运维事件相关？是否需要调整运维策略？

实用指南

使用 Velero 备份和恢复 TopoLVM 文件系统 PVC

Velero 支持对 TopoLVM 文件系统的 Persistent Volume Claims (PVCs) 和 Persistent Volumes (PVs) 进行备份和恢复。此功能已集成到平台中。

本指南专门针对 TopoLVM 文件系统的 PVC。

目录

前提条件

1. 通过 Marketplace/Cluster Plugins 部署 “Alauda Container Platform Data Backup for Velero”。
2. 为 Velero 的 `BackupStorageLocation` 配置一个兼容 S3 的存储。可使用平台提供的 Ceph 或 MinIO 对象存储。

限制

1. S3 存储必须有足够的剩余空间以存储目标集群的所有 PV 数据。
2. 恢复时，命名空间配额和存储类必须支持所有 PVC 的总容量。

操作步骤

第 1 步：配置备份仓库

1. 确保已准备好兼容 S3 的存储。
2. 进入 管理员 > 集群管理 > 备份与恢复 > 备份仓库。
3. 使用对象存储的凭据创建备份仓库。

第 2 步：执行备份

1. 给需要备份的 PVC 和关联的 Pod 打标签：
   Velero 需要一个 Pod 来恢复文件系统 PVC。该 Pod 挂载 PVC，供 Velero 导入数据；如果没有 Pod，PVC 会保持 Pending 状态。对于复杂应用，建议先暂停应用，将 PVC 挂载到一个轻量级 Pod（例如 Nginx）上进行备份/恢复，恢复完成后再恢复原应用配置。

   ```
   kubectl label pvc -n <namespace> <pvc-name> velero-backup=true
   kubectl label pod -n <namespace> <pod-name> velero-backup=true
   ```

2. 进入 备份与恢复，创建新的备份：
   - 选择 备份 **Kubernetes** 资源和 **PVC** 数据卷。
   - 选择包含需要备份数据的命名空间。
   - 按以下配置设置备份：

```yaml
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: <backup-name>
  namespace: cpaas-system
  annotations:
    cpaas.io/description: ''
spec:
  template:
    includedNamespaces:
      - <namespace>
    includedResources:
      - '*'
    labelSelector:
      matchLabels:
        velero-backup: 'true'
    excludedNamespaces: []
    excludedResources: []
    defaultVolumesToFsBackup: true
    storageLocation: default
    ttl: 720h
  schedule: '@every 876000h'
  skipImmediately: false
status:
  phase: Enabled
```

3. 备份完成后，验证 S3 桶中的数据（例如 MinIO）：

```
mc ls <minio-alias>/<bucket-name>/<backup-path>/<namespace>/
```

示例输出：

```
[2025-03-14 00:18:33 CST]  155B STANDARD config
[2025-03-14 09:04:56 CST]   0B data/
[2025-03-14 09:04:56 CST]   0B index/
[2025-03-14 09:04:56 CST]   0B keys/
[2025-03-14 09:04:56 CST]   0B snapshots/
```

第 3 步：恢复集群

1. 在目标集群中，配置与备份时相同的 S3 桶，Velero 会自动检测已有备份。
2. 进入 备份与恢复，创建恢复任务：
   - 选择需要恢复的命名空间。
   - 在高级配置中，如有需要，将原命名空间映射到目标命名空间。
3. 执行恢复操作。
4. 恢复完成后，验证：
   - PVC 名称与原集群一致。
   - PVC 中的应用数据完整且可访问。

从 ACP TopoLVM 本地存储中移除磁盘、设备类卷组或节点

本文档介绍如何移除故障磁盘、从设备类中移除设备类卷组，以及移除 ACP TopoLVM 本地存储中的存储节点。

根据故障范围，本文档涵盖以下场景：

- 从设备类卷组中移除一个 `卷组设备`
- 从设备类中移除一个 `设备类卷组`
- 移除一个 TopoLVM 存储节点

> 风险警告
>
> - 本文档中的操作步骤会直接删除 PVC、PV 以及 `logicalvolumes.topolvm.cybozu.com` 等存储资源。删除后，受影响卷中的数据通常无法恢复，请提前备份数据。
> - 操作前请确认已正确识别目标磁盘、设备类卷组或节点，并为受影响的工作负载安排维护时间窗口。

目录

第 1 步：停止 topolvm-operator

第 2 步：查找受影响的 PVC 和 PV

第 3 步：停止使用受影响 PVC 的工作负载

第 4 步：删除受影响的 Kubernetes 存储资源

第 5 步：更新 TopolvmCluster 资源

第 6 步：更新 lvmdconfig ConfigMap

第 7 步：启动 topolvm-operator

第 8 步：验证节点已移除

术语

术语	说明
device class	由不同节点上的一个或多个设备类卷组组成的逻辑存储类。
device-class volume group	节点上的一个 `LVM 卷组`，表示该节点上设备类的存储资源。
volume group device	节点上的一个磁盘。在 LVM 中对应物理卷。

选择正确的场景

请使用下表判断适用于您环境的场景。

条件	场景 1：从设备类卷组中移除卷组设备	场景 2：从设备类中移除设备类卷组	场景 3：移除 TopoLVM 存储节点
节点可访问性	节点仍可访问。	节点仍可访问。	节点不可恢复，或您决定永久从 TopolvmCluster 中移除该节点。
卷组状态	目标 `LVM 卷组` 仍存在且可识别。	目标 `LVM 卷组` 不再存在或不可识别，但节点仍保留。	节点及其上所有设备类卷组将一并移除。

条件	场景 1：从设备类卷组中移除卷组设备	场景 2：从设备类中移除设备类卷组	场景 3：移除 TopoLVM 存储节点
移除范围	从卷组中移除一个或多个故障磁盘。	从保留的节点中移除一个设备类卷组。	从 TopolvmCluster 中移除整个节点。
保留计划	保留节点和设备类卷组。	保留节点，但不保留目标设备类卷组。	不保留节点及其上任何设备类卷组。

场景 1：从设备类卷组中移除卷组设备

用户场景

- 当设备类卷组未完全损坏，但卷组中一个或多个卷组设备故障需要移除时，使用此操作步骤。

前提条件

- 目标节点仍在集群中且可访问。
- 目标设备类的 `Provision Type` 为 `Thick`。
- 目标设备类卷组未完全损坏，卷组中至少还有一个健康的卷组设备。

检查设备类卷组是否仍可恢复

在目标节点执行以下命令：

```
vgs <vg-name>
```

参数：

- `<vg-name>`：目标 LVM 卷组名称。

若命令返回正常输出，或仅提示部分 PV 缺失但卷组仍存在，则设备类卷组未完全损坏，可继续执行本操作步骤。

示例输出：

```
WARNING: Couldn't find device with uuid VJes6j-2a8V-8Cxf-eW84-yEJK-K24A-y
Bc9OD.
WARNING: VG hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33 is missing PV VJes6j
-2a8V-8Cxf-eW84-yEJK-K24A-yBc9OD (last written to /dev/vdb).
WARNING: Couldn't find all devices for LV hdd-2ab8f0a2-7d1d-42d7-ba6b-da9
4c6185c33/b6b331f0-5242-420f-a531-df90628bef80 while checking used and as
sumed devices.
```

操作步骤

第 1 步：停止 **topolvm-operator**

在 control-plane 节点执行以下命令，防止操作过程中 operator 进行资源调和：

```
kubectl -n nativestor-system scale --replicas 0 deployment topolvm-op
erator
```

第 2 步：查找受影响的 **LVM** 逻辑卷

在目标节点执行以下命令，查找使用故障磁盘的逻辑卷：

```
lvs -a -o +devices <vg-name> | egrep "<path-to-disks>|unknown device"
| awk '{print $1}'
```

参数：

- `<vg-name>`：目标 LVM 卷组名称。
- `<path-to-disks>`：故障磁盘的设备路径，如 `/dev/vdb`。若匹配多个磁盘，用 `|` 分隔。

例如：

```
lvs -a -o +devices hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33 2>/dev/null | egrep "/dev/vdb|unknown" | awk '{print $1}'
```

示例输出：

```
b6b331f0-5242-420f-a531-df90628bef80
```

第 3 步：查找受影响的 PVC 和 PV

在 control-plane 节点执行以下命令，根据逻辑卷名称查找关联的 PV 和 PVC：

```
kubectl get pv -o json | jq -r --arg HANDLE <lv-name> '
  .items[]
  | select(.spec.csi.volumeHandle == $HANDLE)
  | [.metadata.name, .spec.claimRef.namespace, .spec.claimRef.name]
  | @tsv
'
```

参数：

- `<lv-name>`：受影响的 LVM 逻辑卷名称。

示例输出：

```
pvc-e11f6c18-0e15-4c70-9a24-e7136fabfb2f    demo-space  pvc-topolvm
```

输出说明：

- 第 1 列为 `PersistentVolume` 名称。
- 第 2 列为 `PersistentVolumeClaim` 的命名空间。
- 第 3 列为 `PersistentVolumeClaim` 名称。

第 4 步：停止使用受影响 PVC 的工作负载

确认受影响的 PVC 后，停止使用它们的工作负载，确保所有相关 Pod 已停止，然后继续操作。

第 5 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

```
kubectl delete pvc -n <pvc-namespace> <pvc-name>
kubectl delete pv <pv-name>
kubectl delete logicalvolumes.topolvm.cybozu.com <logicalvolume-name>
```

参数：

- `<pvc-namespace>`：受影响 PVC 的命名空间。
- `<pvc-name>`：受影响 PVC 的名称。
- `<pv-name>`：受影响 PV 的名称。
- `<logicalvolume-name>`：TopoLVM `logicalvolumes.topolvm.cybozu.com` 资源名称，与 `<pv-name>` 相同。

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 6 步：清理残留的 LVM 逻辑卷

在目标节点执行以下命令，检查是否仍有逻辑卷残留：

```
lvs -a -o +devices <vg-name> 2>/dev/null | egrep "<path-to-disks>|unk
nown device" | awk '{print $1}'
```

参数：

- `<vg-name>`：目标 LVM 卷组名称。
- `<path-to-disks>`：故障磁盘的设备路径，如 `/dev/vdb`。若匹配多个磁盘，用 `|` 分隔。

若命令仍有输出，逐一删除残留逻辑卷：

```
lvremove <vg-name>/<lv-name>
```

参数：

- `<vg-name>`：目标 LVM 卷组名称。
- `<lv-name>`：要删除的逻辑卷名称。

如有需要，可添加 `--force`。

第 7 步：从 LVM 卷组中移除缺失的物理卷

在目标节点执行以下命令：

```
vgreduce --removemissing <vg-name>
```

参数：

- `<vg-name>`：目标 LVM 卷组名称。

如有需要，可添加 `--force`。

第 8 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit topolvmclusters.topolvm.cybozu.com
topolvm
```

在编辑器中，从目标节点的 `devices` 列表中移除故障卷组设备。例如，从 `nodeName: 192.168.133.50` 的配置中移除 `/dev/vdb`。

之前：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: hdd
            default: true
            devices:
              - name: /dev/vdb
                type: disk
              - name: /dev/vdc
                type: disk
            volumeGroup: hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33
        nodeName: 192.168.133.50
```

之后：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: hdd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33
        nodeName: 192.168.133.50
```

第 9 步：更新 **lvmdconfig ConfigMap**

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit configmaps lvmdconfig-<node-name>
```

参数：

- `<node-name>`：目标节点名称。

在编辑器中，从 `status.json` 中移除故障卷组设备的状态。例如，移除 `/dev/vdb` 的 `deviceStates` 条目。

之前：

```
status.json: '{"node":"192.168.133.50","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-2ab8f0a2-7d1d-42d7-b
a6b-da94c6185c33","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdb","state":"Online"},{"name":"/dev/vdc","s
tate":"Online"}]}],"loops":[],"raids":[]}'
```

之后：

```
status.json: '{"node":"192.168.133.50","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-2ab8f0a2-7d1d-42d7-b
a6b-da94c6185c33","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdc","state":"Online"}]}],"loops":[],"raid
s":[]}'
```

第 10 步：启动 topolvm-operator

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-op
erator
```

第 11 步：验证卷组设备已移除

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com t
opolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认目标节点的 `deviceStates` 中不再出现已移除的卷组设备。

场景 2：从设备类中移除设备类卷组

用户场景

- 当节点上的设备类卷组完全损坏，无法仅通过移除单个卷组设备恢复时，使用此操作步骤。

前提条件

- 目标节点仍在集群中且可访问。
- 目标设备类卷组已完全损坏。
- 移除目标设备类卷组后，节点上至少还保留有另一个设备类卷组。

检查设备类卷组是否完全损坏

在目标节点执行以下命令：

```
vgs
```

若输出中不再出现目标 `LVM 卷组`，则设备类卷组已完全损坏，可继续执行本操作步骤。

> **注意**
>
> 本场景移除的是节点上的整个设备类卷组，而非仅移除该卷组中的单个卷组设备。

操作步骤

第 1 步：停止 topolvm-operator

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 0 deployment topolvm-operator
```

第 2 步：查找受影响的 PVC 和 PV

在 control-plane 节点执行以下命令，查找目标节点上指定存储类关联的 PV、PVC 及 `logicalvolumes.topolvm.cybozu.com` 资源：

```
kubectl get pv -o json | jq -r --arg NODE <node-name> --arg SC <storageclass-name> '
  .items[]
  | select(.spec.nodeAffinity.required.nodeSelectorTerms[]?.matchExpressions[]? | select(.key=="topology.topolvm.cybozu.com/node") | .values[]? == $NODE)
  | select(.spec.storageClassName == $SC)
  | [.metadata.name, .spec.claimRef.namespace, .spec.claimRef.name]
  | @tsv
'
```

参数：

- `<node-name>`：目标节点名称。
- `<storageclass-name>`：目标设备类卷组关联的存储类名称。若涉及多个存储类，请分别执行查询。

示例输出：

```
pvc-e11f6c18-0e15-4c70-9a24-e7136fabfb2f    demo-space  pvc-topolvm
```

输出说明：

- 第 1 列为 `PersistentVolume` 和 `logicalvolumes.topolvm.cybozu.com` 资源名称。
- 第 2、3 列为 `PersistentVolumeClaim` 的命名空间和名称。

若目标设备类卷组关联多个存储类，请对每个存储类重复执行查询及后续清理步骤。

第 3 步：停止使用受影响 PVC 的工作负载

确认受影响的 PVC 后，停止使用它们的工作负载，确保所有相关 Pod 已停止，然后继续操作。

第 4 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

```
kubectl delete pvc -n <pvc-namespace> <pvc-name>
kubectl delete pv <pv-name>
kubectl delete logicalvolumes.topolvm.cybozu.com <logicalvolume-name>
```

参数：

- `<pvc-namespace>`：受影响 PVC 的命名空间。
- `<pvc-name>`：受影响 PVC 的名称。
- `<pv-name>`：受影响 PV 的名称。
- `<logicalvolume-name>`：TopoLVM `logicalvolumes.topolvm.cybozu.com` 资源名称，与 `<pv-name>` 相同。

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 5 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit topolvmclusters.topolvm.cybozu.com
topolvm
```

在编辑器中，从目标节点配置中移除设备类卷组。例如，在 `nodeName: 192.168.140.13` 的配置中，移除 `className: hdd` 及其关联的 `volumeGroup` 和 `devices` 配置。

之前：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
          - className: hdd
            devices:
              - name: /dev/vdb
                type: disk
            volumeGroup: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
        nodeName: 192.168.140.13
```

之后：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
        nodeName: 192.168.140.13
```

若被移除的 `className` 条目含有 `default: true`，请指定另一个剩余的类为默认类。

第 6 步：更新 lvmdconfig ConfigMap

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit configmaps lvmdconfig-<node-name>
```

参数：

- `<node-name>`：目标节点名称。

在编辑器中，从 `lvmd.yaml` 和 `status.json` 中移除对应目标设备类卷组的配置。例如，移除 `className: hdd` 的配置。

之前：

```
lvmd.yaml: |
  socket-name: /run/topolvm/lvmd.sock
  device-classes:
  - name: ssd
    volume-group: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
    default: true
    type: thick
  - name: hdd
    volume-group: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
    default: false
    type: thick
status.json: '{"node":"192.168.140.13","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-97dc00f3-1df6-4f64-8
ddc-7b0b6c5d6de5","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdb","state":"Online"}]},{"className":"ss
d","vgName":"ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1","state":"Read
y","message":"create successful","deviceStates":[{"name":"/dev/vd
c","state":"Online"}]}],"loops":[],"raids":[]}'
```

之后：

```
lvmd.yaml: |
  socket-name: /run/topolvm/lvmd.sock
  device-classes:
  - name: ssd
    volume-group: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
    default: true
    type: thick
status.json: '{"node":"192.168.140.13","phase":"","failClasses":[],"s
uccessClasses":[{"className":"ssd","vgName":"ssd-4a8737fc-48d3-4c61-8
82d-0a5bcc6f77a1","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdc","state":"Online"}]}],"loops":[],"raid
s":[]}'
```

若被移除的 `className` 条目含有 `default: true`，请指定另一个剩余的类为默认类。

第 **7** 步：启动 **topolvm-operator**

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-op
erator
```

第 **8** 步：验证设备类卷组已移除

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com t
opolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认目标节点不再显示已移除的设备类卷组，且剩余设备类卷组状态正常。

场景 **3**：移除 **TopoLVM** 存储节点

用户场景

- 目标节点不可恢复，或您决定永久从 TopolvmCluster 中移除该节点。

前提条件

- 您决定不保留目标节点上的任何设备类卷组。
- 使用目标节点本地卷的工作负载已停止或迁移至其他节点。
- 移除目标节点后，剩余节点仍能承载所需工作负载。

操作步骤

第 1 步：停止 **topolvm-operator**

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 0 deployment topolvm-op
erator
```

第 2 步：查找受影响的 **PVC** 和 **PV**

在 control-plane 节点执行以下命令，查找与目标节点关联的 PV、PVC 及 `logicalvolumes.topolvm.cybozu.com` 资源：

```
kubectl get pv -o json | jq -r --arg NODE <node-name> '
  .items[]
  | select(.spec.nodeAffinity.required.nodeSelectorTerms[]?.matchExpr
essions[]? | select(.key=="topology.topolvm.cybozu.com/node") | .valu
es[]? == $NODE)
  | [.metadata.name, .spec.claimRef.namespace, .spec.claimRef.name]
  | @tsv
'
```

参数：

- `<node-name>`：目标节点名称。

第 3 步：停止使用受影响 **PVC** 的工作负载

确认受影响的 PVC 后，停止使用它们的工作负载，确保所有相关 Pod 已停止，然后继续操作。

第 4 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

```
kubectl delete pvc -n <pvc-namespace> <pvc-name>
kubectl delete pv <pv-name>
kubectl delete logicalvolumes.topolvm.cybozu.com <logicalvolume-name>
```

参数：

- `<pvc-namespace>`：受影响 PVC 的命名空间。
- `<pvc-name>`：受影响 PVC 的名称。
- `<pv-name>`：受影响 PV 的名称。
- `<logicalvolume-name>`：TopoLVM `logicalvolumes.topolvm.cybozu.com` 资源名称，与 `<pv-name>` 相同。

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 5 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit topolvmclusters.topolvm.cybozu.com
topolvm
```

在编辑器中，删除目标节点的整个配置块。例如，删除 `nodeName: 192.168.140.13` 的配置块。

之前：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: hdd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33
        nodeName: 192.168.133.50
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
          - className: hdd
            devices:
              - name: /dev/vdb
                type: disk
            volumeGroup: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
        nodeName: 192.168.140.13
```

之后：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: hdd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33
        nodeName: 192.168.133.50
```

第 6 步：更新 **lvmdconfig ConfigMap**

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system edit configmaps lvmdconfig-<node-name>
```

参数：

- `<node-name>`：目标节点名称。

在编辑器中，删除整个 `lvmd.yaml` 和 `status.json` 部分，不保留任何已移除节点的配置或状态。

第 7 步：启动 topolvm-operator

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-operator
```

第 8 步：验证节点已移除

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com topolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认 `status.nodeStorageState` 中不再出现目标节点，例如 `192.168.140.13` 不再显示。

配置条带逻辑卷

当您向 LVM 逻辑卷写入数据时，文件系统会将数据分布到底层的物理卷上。您可以通过创建条带逻辑卷来控制数据写入物理卷的方式。对于大规模的顺序读写操作，这可以提高数据 I/O 的效率。

条带化通过以轮询方式将数据写入预定数量的物理卷来增强性能。使用条带化，I/O 可以并行进行。在某些情况下，每增加一个物理卷，性能几乎呈线性提升。

TopoLVM 通过在 `StorageClass` 中指定 `lvcreate-option-class` 来实现这一点。

目录

前提条件

- 设备类必须在单个节点上包含至少两个设备。

操作步骤

1 使用默认的 `lvcreate-option-class`

1. 进入 管理员。
2. 在左侧边栏，依次进入 存储管理 > 存储类。
3. 点击 创建存储类。
4. 选择 块存储。
5. 选择 **TopoLVM**，然后点击 下一步。
6. 配置所需的存储类参数。
7. 切换到 **YAML** 视图 并添加 `topolvm.io/lvcreate-option-class` 参数：

```
parameters:
  topolvm.io/lvcreate-option-class: striped-default
```

> **NOTE**
>
> `striped-default` 是 Alauda Container Platform 内置的 lvcreate-option-class。当 TopoLVM CSI 驱动创建逻辑卷时，它会自动向 lvcreate 命令追加 `--stripes=2` 和 `--stripesize=64`。

2 创建自定义 **lvcreate-option-class**（可选）

如果内置的 striped-default 类不能满足您的需求，您可以定义自定义的 LVCreateOptionClass：

```
cat << EOF | kubectl apply -f -
apiVersion: topolvm.cybozu.com/v2
kind: LVCreateOption
metadata:
  name: custom
  namespace: nativestor-system
spec:
  options:
  - name: striped-custom
    type: striped
    options:
    - --stripes=3
    - --stripesize=64
EOF
```

该清单创建了一个名为 striped-custom 的自定义 LVCreateOptionClass，具有 3 条条带和 64 KiB 的条带大小。应用后，您可以在 StorageClass 中通过 `topolvm.io/lvcreate-option-class: striped-custom` 引用它。