存储系统

概览

概览

选择存储系统

相关信息

Ceph 分布式存储

介绍

功能概述

规划部署

规划部署

部署架构

安全注意事项

基础设施要求

架构

第 1 章：简介

第 2 章：架构概

第 3 章：Opera

安装

升级

核心概念

实用指南

LVM 存储

介绍

安装

前提条件

操作指南

实用指南

本地存储

介绍

设备发现不会创建卷

相关信息

安装

先决条件

操作步骤

后续步骤

操作指南

实用指南

概览

Alauda Container Platform 提供三种存储系统的文档。请根据卷的配置方式以及数据是否必须在节点故障后保留来选择。

不要将 Local Storage 和 LVM Storage 当作可以互换的名称。Local Storage 是静态本地磁盘置备程序。LVM Storage 是基于 TopoLVM 的动态置备程序。

目录

选择存储系统

相关信息

选择存储系统

存储系统	置备方式	典型用途	数据保护
Local Storage	静态。每个已列出的磁盘或分区对应一个 PersistentVolume。	将特定本地磁盘提供给工作负载使用，也可作为其他存储系统使用的 Block 卷。Marketplace 软件包名称为 **Alauda Container Platform Local Disk Provisioner**。	节点故障会导致卷不可用。磁盘故障可能导致数据丢失。

存储系统	置备方式	典型用途	数据保护
LVM Storage	动态。TopoLVM 从节点卷组中划分逻辑卷。	需要低延迟本地卷和 PVC 扩容的数据库和缓存。Marketplace 软件包名称为 **Alauda Container Platform LVM Volume Provisioner**。	节点故障会导致卷不可用。磁盘故障可能导致数据丢失。
Ceph Distributed Storage	动态。使用副本或纠删码池。	必须在节点故障后仍可使用的共享块、文件和对象存储。	在多个节点间进行副本复制或纠删码保护。

相关信息

- Local Storage
- LVM Storage
- Ceph Distributed Storage

TopoLVM 是 topolvm Community 的项目。 Alauda 是独立供应商。本产品与 topolvm 项目无关联，也未获其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

Ceph 分布式存储

介绍

介绍

功能概述

规划部署

规划部署

规划部署

部署架构

安全注意事项

基础设施要求

网络要求

灾难恢复规划

性能规划

后续步骤

架构

架构

第 1 章：简介
第 2 章：架构概述
第 3 章：Operators
第 4 章：安装概述
第 5 章：升级概述

安装

在 Alauda OS 上部署的先决条件

Alauda OS 要求
Huawei DCS
Huawei Cloud Stack (HCS)
VMware vSphere
裸金属
安装前检查清单

以内部模式部署

先决条件
注意事项
操作步骤
相关操作

以外部模式部署

注意事项
前提条件
操作步骤
后续操作

使用本地存储

先决条件
操作步骤
扩展设备类中的
添加设备类
相关信息

创建 Stretch

术语
典型部署方案
约束和限制
前提条件
操作步骤
相关操作

升级

升级 Operator

开始之前

操作步骤

如果详情页按钮仍处于禁用状态

相关主题

将 Ceph 从 Squid (v19) 升级到 Tentacle (v20)

要求

操作步骤

如果升级未取得进展

相关主题

核心概念

核心概念

Rook Operator

Ceph CSI

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

约束和限制

操作步骤

参考

替换或移除由 PVC 支持的 OSD

先决条件

约束和限制

识别 OSD 和 PVC

移除 OSD（缩容）

替换失败的 OSD（保持容量）

替换或移除

先决条件

约束和限制

操作步骤

参考资料

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

灾难恢复

清理分布式

注意事项

操作步骤

更新优化参

创建 Ceph Object Store 用户

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 Terminal 验证存储池配额

为 Ceph RG

背景

配置传输加密

概述

限制与前提条件

设置 Ceph OSD 满容量阈值

前提条件

操作步骤

建议

持久卷加密

概述

配置 Multus 多网卡网络

概述

限制和前提条件

在创建集群时配置多网卡网络

验证

故障排除建议

使用 Ceph RGW Exporter 监控 Ceph RGW

术语

前提条件

工作原理

操作步骤

自定义 exporter

指标参考

故障排除

卸载

介绍

Alauda Storage 是平台在集群内提供的超融合存储解决方案。该方案基于开源的 Rook + Ceph 存储解决方案，通过分布式存储实现自动管理、自动扩展和自动修复，满足中小型应用对块存储、文件存储和对象存储的需求。

> **NOTE**
>
> 在本文档中，分布式存储指此集群内的 Ceph 存储，而外部存储指此集群外的 Ceph 存储。

目录

功能概述

规划部署

功能概述

- 轻松部署：提供图形化的自动部署和存储集群管理服务；支持计算与存储一体化和解耦两种部署模式。
- 专业运维：提供持久卷快照备份和克隆新卷功能；可视化监控容量、性能和组件级别；内置告警策略，满足大多数存储运维场景的需求。
- 安全可靠：通过分布式和多副本机制确保数据安全可靠；简单可靠的自动化管理支持在线扩展存储资源。

- 卓越性能：提供弹性且高性能的存储服务；支持部署混合磁盘设备，以提升存储系统的性能和效率。

规划部署

请参阅规划部署。如需在 Local Storage、LVM Storage 和 Ceph Distributed Storage 之间进行选择，请参阅概述。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关，未经其认可，也未得到其赞助。所有商标均为其各自所有者的财产，本文仅将其用于标识目的。

规划部署

本主题提供了在 Alauda Container Platform (ACP) 上部署 Ceph 分布式存储的规划检查清单。其中总结了架构选择、安全选项、基础设施规模、网络限制和灾难恢复注意事项，帮助您在执行实际安装之前确定部署模型。

有关产品背景，请参阅简介和架构。有关部署操作步骤，请参阅安装和操作指南下的文档。

如果集群节点运行 Alauda OS，请先查看在 Alauda OS 上创建，然后在安装前阅读在 Alauda OS 上部署的前提条件，并使用该页面中的提供商链接规划磁盘和节点替换。

目录

- 部署架构
  - 内部和外部部署模型
  - 内部模式 OSD 供给
  - 节点角色
- 安全注意事项
  - 存储类加密
  - 传输中加密
- 基础设施要求
  - 最低配置和推荐配置
  - 资源规模
  - 集群总量规划预算
  - 如何估算集群规模
  - Pod 放置

存储设备规划
容量规划
网络要求
IPv6 支持
灾难恢复规划
Regional-DR
Stretch Cluster
性能规划
后续步骤
内部部署
外部部署
相关后续配置

部署架构

ACP 分布式存储基于 Ceph 和 Rook。在较高层面上，该平台组合了以下层：

- MON、MGR、OSD、MDS 和 RGW 等 Ceph 守护进程，用于提供块、文件和对象存储功能
- Rook 和 CSI 组件，用于自动执行部署、配置、扩展和生命周期管理
- ACP 平台集成，用于提供存储池、可观测性和运维入口

在部署之前，请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之一规划 ACP 分布式存储：

部署模式	存储服务运行位置	存储集群管理者	最适用场景	主要权衡
内部、共驻	Ceph 组件运行在同时运行业务工作负载的同一 ACP worker 节点上	ACP 平台团队或集群管理员	早期阶段环境、裸金属集群，或尚未完全明确存储需求的场景	部署更简单，但应用与存储之间更容易发生资源争用
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	运维隔离和规模控制更佳，但需要预留更多节点并进行容量规划
外部	ACP 使用外部 Ceph 环境中的存储类	独立的存储团队、SRE 团队或现有外部存储所有者	大规模环境、多个使用方集群，或已经运行独立 Ceph 集群的组织	所有权边界清晰，但需要进行更多跨集群网络、身份验证和依赖管理

内部部署更易于推出和管理，因为存储服务与使用存储的工作负载均在同一 ACP 环境中进行规划。在内部部署中，首先需要决定存储是否应与业务工作负载共享节点，还是使用专用节点。当您需要在存储集群与应用集群之间实现更强的隔离，或者多个工作负载集群需要共享同一存储后端时，外部部署更为适合。

主要规划决策点如下：

- 如果希望更快推出，并且能够接受存储和应用工作负载共享同一 worker 池，请选择共驻部署。
- 如果存储需求已知，并且希望获得更清晰的容量控制、故障隔离和维护边界，请选择专用节点部署。
- 如果存储已在其他位置进行管理，或者单个外部集群必须为多个 ACP 集群提供服务，请选择外部部署。

内部模式 OSD 供给

选择内部模式后，请选择 OSD 设备的供给方式。不要在同一个 `CephCluster` 上混用这两个后端。

OSD 供给方式	设备选择方式	安装操作步骤	最适用场景
主机设备	内部模式向导会选择节点上的磁盘或分区（`Selection Device` 或 输入设备）。	在内部模式下部署	默认的 Web 控制台安装。
Local Storage 块卷	您先使用 Local Storage 配置块 PersistentVolume，然后 Rook 将其作为 OSD 使用。	使用 Local Storage 在内部模式下部署	您已经使用 Local Storage，或 OSD 必须保留在持有该卷的节点上。

主机设备选择设备仍可使用 Local Storage 发现功能。这与基于 PVC 的 OSD 不同。基于 PVC 的 OSD 需要来自 Local Storage 指南的可用块 PersistentVolume。

节点角色

规划节点放置时，请区分控制平面节点、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，基础设施节点非常适合。
- worker 节点可以在共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争用。

对于生产环境，请为高可用存储服务至少规划三个故障域。应尽可能将存储节点分布在不同的机架、zone 或主机组中。

安全注意事项

在部署之前，请确认存储设计是否需要传输中加密，并在启用之前评估其对运维的影响。

存储类加密

您可以使用存储类加密对 PersistentVolume（仅限块卷）进行加密，并使用外部 Key Management System（KMS）存储设备加密密钥。PersistentVolume 加密仅适用于 RADOS

Block Device（RBD）PersistentVolume。请参阅如何创建带有 PersistentVolume 加密功能的存储类。

Alauda Storage 同时支持 HashiCorp Vault KMS 和 Thales CipherTrust Manager KMS 的存储类加密。

传输中加密

ACP 当前支持对 Ceph 分布式存储进行传输中加密。此功能可保护 Ceph 组件与客户端之间的流量，通常需要结合 Ceph `msgr2` 和集群网络模型进行规划。

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
- 繁忙存储节点上预期的 CPU 开销
- 对目标硬件吞吐量和延迟的影响

有关实施详情，请参阅配置传输中加密。

基础设施要求

最低配置和推荐配置

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	4 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	每个节点配备多个专用设备，类型和大小保持一致
节点分布	有 3 个节点可承载 Ceph 服务	3 个故障域，例如机架或 zone

项目	最低配置	推荐配置
设备使用	系统磁盘与存储磁盘分离	为 Ceph 数据使用专用裸磁盘，并为未来扩展预留空间

集群至少应包含三个节点，并且每个节点上都有一个可用的存储设备。对于生产环境，请将集群部署在至少三个故障域中，并预留足够的可用资源，以应对重新平衡、修复和未来增长。

资源规模

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，然后为恢复、重新平衡、升级和后台任务预留额外空间。

作为基线：

- 高可用集群至少从三个存储节点开始
- 为 MON、MGR、OSD 以及任何启用的 MDS 或 RGW 服务预留足够的 CPU 和内存
- 为新存储池、额外设备和集群恢复事件预留增长空间
- 避免规划在第一天就接近饱和的集群

如果设计使用专用存储节点，资源规划会更加可预测。如果存储与业务工作负载共同运行，请预留额外空间，以便在峰值负载和节点故障期间应对资源争用。

集群总量规划预算

在早期规模规划中，应从集群总量预算开始，而不是仅从各组件的单独数值开始。下表用于在进行特定工作负载调优之前，为四节点高可用集群提供规划参考：

部署模式	为存储预留的总 **CPU**	为存储预留的总内存	说明
内部模式，最低基线	32 个逻辑 CPU	96 GiB	仅满足最低部署目标时的四节点入门级规划基线
内部模式，标准基线	40 个逻辑 CPU	96 GiB	更适合作为常规生产规划和未来扩展的起点

部署模式	为存储预留的总 **CPU**	为存储预留的总内存	说明
内部模式，面向性能的基线	60 个逻辑 CPU	128 GiB	适用于从一开始就需要更高吞吐量或更低延迟的场景
外部使用方集群	仅根据连接和客户端访问进行规模规划	仅根据连接和客户端访问进行规模规划	存储守护进程在 ACP 集群外运行，因此 ACP 集群主要需要网络可达性、凭据和客户端侧容量

这些值应视为集群级规划目标，而不是精确的调度器预留值。要估算四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。

以下建议适用于早期规划：

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

这些值是规划参考，而非严格的调度保证。实际要求取决于设备数量、启用的服务和工作负载强度。

如何估算集群规模

调整集群规模时，请按以下顺序操作：

1. 选择部署模式：共驻、专用节点或外部模式。
2. 确定最小节点数量和故障域布局。
3. 确定是否需要块、文件、对象或混合存储服务。

4. 从集群总量规划预算开始。
5. 为额外设备集、恢复、监控和预期增长增加预留空间。

如果同时需要文件和对象服务，或者集群还将同时承载繁重的业务工作负载，请按高于最低基线的规模进行规划，而不要直接采用最低基线。

Pod 放置

Pod 放置规则会直接影响弹性。请规划集群，使其满足以下条件：

- 高可用组件可以分布在不同故障域中
- 每个故障域都有可访问的存储设备和足够的可分配资源
- 新设备集或未来扩展仍可遵循相同的放置模式

在实践中，仅有三个节点并不足够。这些节点还需要以能够避免单个机架、主机组或 zone 成为单点故障的方式进行分布。

存储设备规划

选择存储设备时，应尽可能统一设备大小和类别。混合设备会增加性能调优和容量规划的复杂度。

请遵循以下原则：

- 为操作系统预留一个系统磁盘，并为 Ceph 数据单独配置存储设备
- 优先使用裸磁盘或专用设备，而不是对共享磁盘进行分区
- 将每个节点上的设备数量保持在可管理范围内，以便恢复和维护仍然可行
- 跟踪可用容量而不是原始容量，因为副本会减少有效存储空间

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

有关相关运维指南，请参阅管理存储池；对于主机设备集群，请参阅添加设备/设备类；当 OSD 使用 Local Storage 块 PersistentVolume 时，请参阅使用 Local Storage 在内部模式下部署。

容量规划

规划集群容量时，请计算可用容量而不是原始磁盘容量。在副本 Ceph 部署中，部分原始存储始终会用于数据保护。

请遵循以下规划原则：

- 让可用容量领先于预期业务增长，而不是等集群几乎已满后才扩展
- 为恢复、重新平衡、快照和数据使用量的临时突发预留额外空间
- 跨节点和故障域以均衡方式扩展存储，避免新增容量造成利用率倾斜
- 在向集群添加新工作负载之前，同时检查当前利用率和预计增长

以下示例可作为早期规划参考，适用于包含三个节点、每个节点一个设备且采用 3 副本数据保护策略的集群：

每个节点的设备大小	集群原始容量	使用 **3** 个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

这些值仅为示例。可用容量会因实际数据保护策略而异，不应将其视为适用于所有集群设计的通用规则。

在日常运维中，应在集群达到告警级别之前检查容量。如果增长可预测，请尽早扩展，而不要等待集群接近满载或完全满载。

网络要求

Ceph 对网络质量十分敏感。部署前，请验证以下事项：

- 集群网络能够为复制和恢复流量提供稳定的吞吐量
- 故障域之间的延迟符合所选部署模型的支持范围
- 存储节点与使用存储的集群之间已开放所需端口

- 是否采用专用网络设计（例如基于 Multus 的网络隔离）已提前确定

如果计划将存储流量与常规原生应用流量隔离，请在部署前确认网络接口、路由策略和运维责任归属。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的集群网络协议栈。

- 单栈 IPv6 环境支持 IPv6。
- 双栈规划必须在存储部署前，根据 ACP 集群网络设计进行验证。
- 存储节点和客户端节点应采用相同的地址族策略，以避免连接和服务发现问题。

如果环境使用 IPv6，请在安装前确认以下事项：

- ACP 集群网络已配置为支持 IPv6
- 所有存储节点均可通过所需的 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。不要假设现有的 IPv4 存储设计可以在之后直接转换，而无需重新验证。

灾难恢复规划

ACP 分布式存储可以采用不同的恢复目标进行规划。请根据恢复点目标 (RPO)、恢复时间目标 (RTO) 和站点拓扑选择相应模型。

Regional-DR

ACP 支持 Regional-DR，适用于跨地域或跨站点灾难恢复场景，且此类场景能够接受异步复制和少量潜在数据丢失。

规划 Regional-DR 时，请提前确认以下事项：

- 源集群和目标集群的存储及网络设计兼容
- 复制延迟和故障转移预期符合业务恢复目标

- 明确受保护的工作负载类型，例如块数据、文件系统或对象数据

有关实施详情，请参阅灾难恢复。

Stretch Cluster

仅当站点间延迟受到严格控制，且拓扑专为此模式设计时，才适合采用 Stretch Cluster。通常应规划以下内容：

- 两个数据站点和一个仲裁站点或仲裁器站点
- 三个区域中至少五个节点
- 在创建集群前手动明确设置故障域标签
- 每个数据站点均有足够的节点，以保持存储服务的可用性
- 区域间延迟处于低延迟设计范围内，通常要求数据站点之间的 RTT 不超过 10 ms

> **WARNING**
>
> 不要将 Stretch Cluster 视为适用于远距离、高延迟、多数据中心部署的通用解决方案。如果站点间延迟未受到严格控制，请改用专用的灾难恢复架构。

有关 ACP 专用的 Stretch Cluster 部署指南，请参阅创建 Stretch Type Cluster。

性能规划

性能规划应基于工作负载特征，而不应仅依据原始设备数量。在部署前，请明确：

- 主要工作负载是面向块、文件还是对象
- 工作负载属于延迟敏感型、吞吐量敏感型，还是容量密集型
- 热数据、备份流量或分析作业是否会占用集群的主要资源

同时确认是否需要特殊调优或针对特定功能进行设计。例如，对象工作负载可能需要单独规划网关容量，某些环境可能还需要面向缓存或专用集群的设计。

后续步骤

完成规划后，请根据所选部署模型继续执行相应的部署指南：

内部部署

- 对于在主机设备上共置部署，请参阅在 Internal Mode 中部署。
- 对于在 Local Storage Block 卷上共置部署，请参阅使用 Local Storage 在 Internal Mode 中部署。
- 对于 Stretch Cluster 部署，请参阅创建 Stretch Type Cluster。
- 对于专用节点部署，请参阅为分布式存储配置专用集群。

外部部署

- 若要使用其他集群或外部 Ceph 环境提供的存储服务，请参阅在 External Mode 中部署。

相关后续配置

- 若要为已部署的存储服务启用加密网络流量，请参阅配置传输中加密。
- 若要在部署后配置灾难恢复，请参阅灾难恢复。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品不隶属于 LF Projects, LLC 或 The Linux Foundation，也不受其认可或赞助。所有商标均归其各自所有者所有，本文仅将其用于标识目的。

架构

目录

第 1 章：简介
第 2 章：架构概述
第 3 章：Operators
Rook-Ceph operator
组件
设计图
职责
资源
生命周期
ACP storage operator
组件
设计图
职责
资源
生命周期
第 4 章：安装概述
operator 安装
Ceph 集群创建概述
CephCluster 创建
内部模式

外部模式

第 5 章：升级概述

第 1 章：简介

Alauda Storage 是 Alauda Container Platform (ACP) 的云原生存储层。它以 Kubernetes 原生系统的形式运行，并通过 operator、自定义资源和 CSI 驱动程序提供持久化存储服务。

从应用的角度来看，该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：为对延迟敏感或类似数据库的工作负载提供底层块设备，这类工作负载需要专用卷。
- 对象存储：提供兼容 S3 的对象访问，用于非结构化数据、备份制品和数据平台集成。

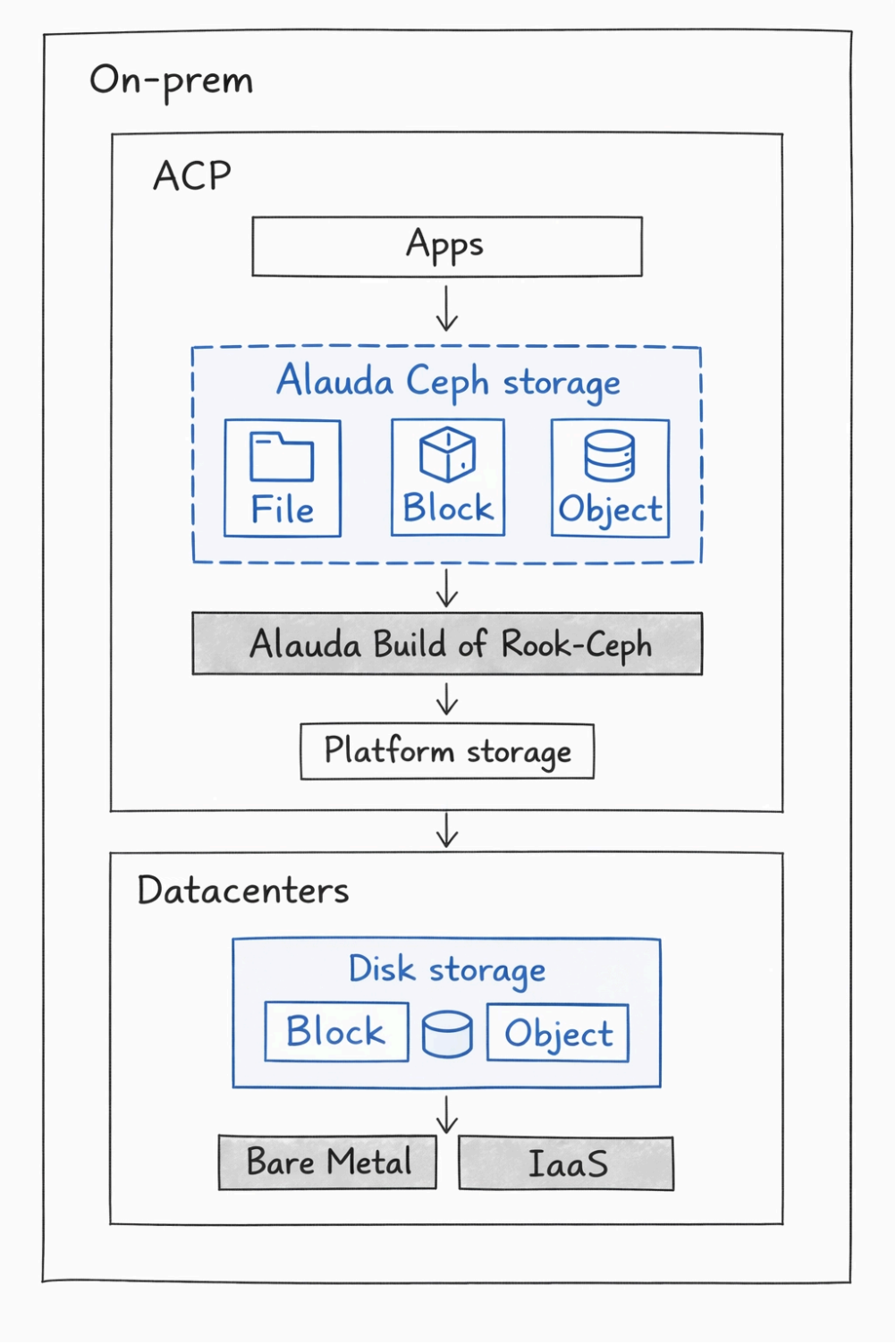
Alauda Storage 集成了以下核心开源项目：

- Ceph：提供块、文件和对象存储功能的分布式存储引擎。
- Rook：用于部署和管理 Ceph 集群的 Kubernetes operator 框架。
- Ceph CSI Operator：用于管理 Ceph CSI 驱动程序组件及其生命周期的 operator。
- Ceph CSI：Kubernetes 用于 Ceph 后端卷动态置备、连接/挂载及生命周期操作的 CSI 实现。

第 2 章：架构概述

从较高层次来看，**Alauda Storage** 完全运行在 **Alauda Container Platform** 内部。

Alauda Storage 架构



Alauda Storage 使用故障域配置来确定数据副本在集群中的分布方式。故障域表示一个物理边界，例如主机、机架或可用区，数据副本会分布在该边界内以确保高可用性。**Alauda Storage** 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 Storage 由以下两个 Operator Lifecycle Manager (OLM) operator bundle 组成，其中部署的 operator 会将管理任务和自定义资源编码为规范，以便轻松实现任务和资源特征的自动

化：

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

管理员定义集群的期望最终状态，operator 则确保集群达到或趋近该状态，同时尽量减少管理员的干预。

Rook-Ceph operator

`rook-ceph-operator` 是 **Alauda Storage** 中用于 Ceph 的 operator。它使 Ceph 存储系统能够作为由 Kubernetes 管理的服务运行在 **Alauda Container Platform** 上。

`rook-ceph-operator` 容器会自动引导存储集群，并监控存储守护进程以维护集群运行状况。

组件

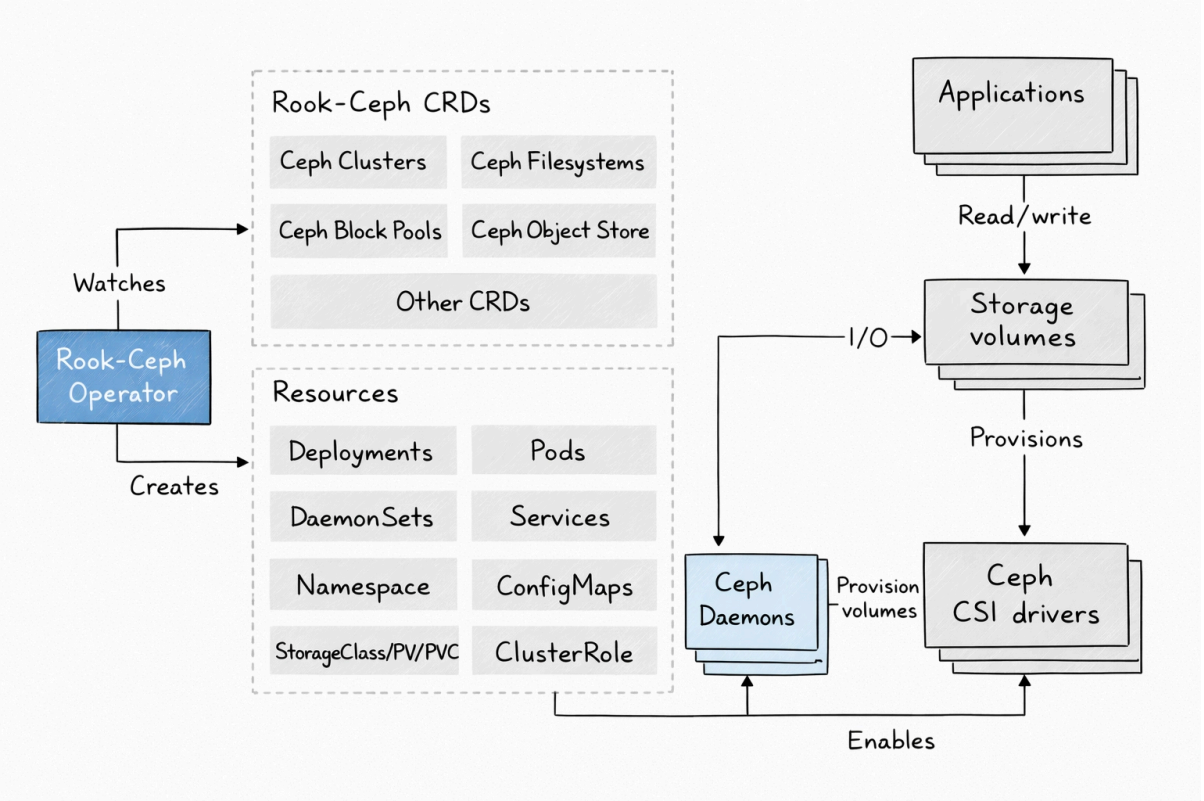
`rook-ceph-operator` 管理以下 Ceph 守护进程组件：

- Mons：Ceph 监视器（`mons`）负责维护集群元数据和仲裁。
- OSDs：对象存储守护进程（`OSDs`）负责存储和复制数据。
- Mgr：Ceph manager（`mgr`）提供指标和内部管理功能。
- RGW：RADOS Gateway（`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 与 ACP 集成。

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，具体职责如下：

- 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 和制品：
  - 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，以保持存储可用且运行正常。
- 部署和管理监视器的放置，并在集群规模变化时更新监视器配置。
- 监视通过 API 服务发出的期望状态变更请求，并应用这些变更。
- 初始化 Ceph-CSI 驱动程序。
- 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。它不会修改数据路径，也不会公开所有底层 Ceph 配置；为了简化运维模式，placement group 和 CRUSH map 等高级 Ceph 结构仍保持抽象。

资源

operator 会在 `rook-ceph` 命名空间中为其创建的资源添加所有者引用。卸载期间，这些引用可确保相关资源也被删除，包括 `ConfigMaps`、`Secrets`、`Services`、`Deployments` 和 `DaemonSets`。

operator 会监视并协调以下自定义资源：

- `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 Driver：
  - 一个 deployment 管理两个 provisioner pod。
  - 一个 DaemonSet 管理每个节点上的一个 plugin pod。
- CephFS CSI Driver：
  - 一个 deployment 管理两个 provisioner pod。
  - 一个 DaemonSet 管理每个节点上的一个 plugin pod。
- 监视器（`mons`）：
  - 三个监视器 pod，每个 pod 都有自己的 deployment。
  - 在 Ceph 拉伸集群模式下，共有五个监视器 pod：仲裁区一个，每个数据区两个。
- Manager（`mgr`）：
  - 标准 deployment 中有两个 manager pod。

- 对象存储守护进程（`OSDs`）：
  - 初始创建至少三个 OSD。
  - 随着集群容量扩展，会添加更多 OSD。
- 元数据服务器（`MDS`）：
  - 两个元数据服务器 pod。
- RADOS Gateway（`RGW`）：
  - 至少两个 gateway pod。

ACP storage operator

`acp-storage-operator` 是 ACP 专用 operator，负责协调 `CephCluster` 资源，并管理 ACP 内置的存储集成，涵盖运维、监控和故障排除。

> **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 协调控制器。
- 两个 `MutatingWebhookConfiguration` 条目：
  - 一个用于 `CephCluster`
  - 一个用于 `CephFilesystem`

设计图

`acp-storage-operator` 在 ACP 中作为控制平面 operator 运行，并通过 ACP 原生监控和运维资源扩展 Ceph 集群生命周期。

职责

`acp-storage-operator` 负责执行以下操作：

- 监视并协调 `CephCluster` 资源。
- 通过变更型 webhook 在创建时修改 `CephCluster` 和 `CephFilesystem` 对象。
- 在变更过程中初始化 ACP 专用默认值，包括 `resources`、`placement` 和相关运行时设置。
- 创建 ACP 所需 Ceph manager 功能的内置 `.mgr` 池。
- 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

对于每个协调的 `CephCluster`，`acp-storage-operator` 都会创建和管理以下资源：

- 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 deployment 启动，并为 `CephCluster` 和 `CephFilesystem` 注册控制器和变更型准入 webhook。
2. 准入阶段：收到创建请求时，webhook 会在对象持久化之前注入 ACP 默认值（例如 `resources` 和 `placement`）。
3. 协调阶段：`CephCluster` 可用后，operator 会持续协调 ACP 内置资源，包括 `.mgr` 池、`AlertRule`、`MonitorDashboard` 和 `rook-ceph-tools` pod。
4. 清理阶段：删除 `CephCluster` 时，operator 会移除其为该集群创建的所有由 ACP 管理的资源。

第 4 章：安装概述

本章介绍内部模式和外部模式下的安装流程。

operator 安装

- OLM 为 ACP 内置存储集成安装 `acp-storage-operator`。
- OLM 从其软件包安装 rook-ceph-operator，并创建 operator deployment。
- 该 deployment 包含协调 `CephCluster` 和 Ceph 相关资源所需的控制组件。
- `acp-storage-operator` 为 `CephCluster` 和 `CephFilesystem` 对象注册变更准入 webhook。
- 启动后，operator 开始监视 CRD，并准备存储使用所需的 CSI 组件。

Ceph 集群创建概述

- 管理员创建 `CephCluster` 自定义资源。
- operator 使用该声明编排 Ceph 相关资源和 CSI 服务。
- 在内部模式下，`acp-storage-operator` 还会围绕同一个 `CephCluster` 协调 ACP 内置存储资源。
- 具体的协调路径取决于部署模式：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 集群内部。

1. operator 在存储命名空间中接收 `CephCluster`。
2. `acp-storage-operator` 变更准入 webhook 为 `CephCluster` 和 `CephFilesystem` 资源初始化 ACP 默认值（例如 `resources` 和 `placement`）。
3. `rook-ceph-operator` 引导 monitor 仲裁组并创建 manager 守护进程。
4. OSD 准备作业发现已配置的设备，然后创建 OSD deployment。

5. 为 RBD 和 CephFS 部署 Ceph CSI provisioner 和节点插件 pod。
6. `acp-storage-operator` 协调 ACP 内置资源，包括 `.mgr` 池、内置 `AlertRule`、内置 `MonitorDashboard` 和 `rook-ceph-tools` pod。
7. `StorageClass` 对象可通过 PVC 工作流用于动态置备。
8. 集群运行状况和就绪状态会通过 `CephCluster` 状态持续报告。

外部模式

在外部模式下，Ceph 数据平面运行在外部 provider 集群中，而 ACP 承载使用方的控制集成。

1. operator 在使用方 ACP 集群中安装并协调 CSI 组件。
2. 导入外部 Ceph 连接数据（例如 monitor 端点、集群标识符和客户端 secret）。
3. 在外部模式下协调 `CephCluster`（`external: true`），以表示远程 Ceph 后端。
4. 不会在使用方集群中创建本地 monitor 或 OSD 守护进程。
5. RBD 和 CephFS CSI 驱动程序使用导入的连接信息，从外部 Ceph 集群置备并挂载卷。
6. 根据使用方组件和外部连接检查的结果协调状态和运行状况。

第 5 章：升级概述

Alauda Storage 通过 `Subscription`、`InstallPlan` 和 `ClusterServiceVersion`（CSV）资源遵循由 OLM 管理的升级工作流。有关受支持的 operator 与数据平面升级操作步骤，请参阅升级。

当前 channel 中出现新 bundle 后，OLM 会创建一个 `InstallPlan`，其执行取决于自动或手动批准策略。

在批准 `InstallPlan` 后，OLM 执行 CSV 协调：

- 安装新的 CSV。
- 将 operator Deployment 更新为新镜像并重启。
- 成功完成转换检查后，替换旧 CSV。

安装的 CSV 变为 `Succeeded` 后，平台会将唯一的 CSV `relatedImages` 条目写入内部 `CephCluster.spec.cephVersion.image`，其 `name` 为 `ceph`。如果该镜像已经是正在运行

的值，则不会滚动更新 Ceph 守护进程。外部模式集群不会接收本地 Ceph 镜像写入。

随后，operator 级别的协调会完成剩余配置：

- Operator 验证并重新应用来自面向用户的 CR 的期望状态。
- 检查 Ceph 和 CSI 资源，确保其配置与版本一致。
- 持续纠正偏差，直到集群状态报告运行正常且已收敛。

> **WARNING**
>
> 不要依赖对生成的 CSV 内容进行手动编辑，将其作为持久化自定义机制。在后续升级期间，OLM 协调可能会覆盖这些临时更改。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立的供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 没有任何关联，也未获得其认可或赞助。所有商标均归其各自所有者所有，此处仅用于标识目的。

安装

Internal Mode 可以通过以下任一方式为 OSD 提供后端存储：

- 在 Web 控制台中选择主机设备。请参阅在 Internal Mode 中部署。
- 使用 Local Storage 中的块 PersistentVolumes。请先在 Local Storage 指南 中配置卷，然后参阅使用 Local Storage 在 Internal Mode 中部署。

请勿在同一个 `CephCluster` 上混用这两种 OSD 后端。External Mode 会使用现有的 Ceph 服务，而不是创建新的服务。

在 **Alauda OS** 上部署的先决条件

Alauda OS 要求

Huawei DCS

Huawei Cloud Stack (HCS)

VMware vSphere

裸金属

安装前检查清单

以内部模式部署

先决条件

注意事项

操作步骤

相关操作

以外部模式部署

注意事项

前提条件

操作步骤

后续操作

使用本地存储

先决条件

操作步骤

扩展设备类中的

添加设备类

相关信息

创建 **Stretch**

术语

典型部署方案

约束和限制

前提条件

操作步骤

相关操作

对于节点运行 Alauda OS 的集群，请先阅读在 Alauda OS 上创建，然后在安装前阅读在 Alauda OS 上部署的先决条件。

如需升级现有的 **Alauda Storage** Operator，请参阅升级。

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 没有关联，也未获其认可或赞助。所有商标均归其各自所有者所有，此处仅用于标识目的。

在 Alauda OS 上部署的先决条件

本页面介绍在节点运行 Alauda OS 的集群上部署 Alauda Storage 的先决条件。

本页面仅介绍在 Alauda OS 集群上部署 Alauda Storage 特有的存储持久化和节点替换要求。持久化 MON/Rook 数据和本地 OSD 磁盘的要求适用于 Internal Mode，在该模式下 Ceph 守护进程运行于集群节点上。External Mode 使用远程 Ceph 数据平面，并遵循该远程系统的持久化要求。有关常规 Ceph 要求（例如节点数量、CPU 和内存、网络、时间同步、容量、版本和运行状况检查），请参阅规划部署以及适用的安装文档。在 Alauda OS 集群上，操作系统、Kubernetes 和节点配置通过替换节点进行更新，因此存储数据不得依赖于节点本地的临时状态。

涵盖以下基础设施类型：

- Huawei DCS
- Huawei Cloud Stack (HCS)
- VMware vSphere
- 裸金属

对于 global 集群，请在 global 集群中创建和更新 MachineTemplate、ConfigPool、MachineDeployment 及相关资源，通常位于 `cpaas-system` 命名空间中。有关概述，请参阅在 Alauda OS 上创建。有关提供商契约，请参阅基础设施资源、在 Alauda OS 集群上管理节点和 API 参考。

以下提供商示例仅展示为 Ceph 建立持久化存储的字段。它们不是完整的、可直接应用的清单；请根据您环境的提供商文档补充其余必需的提供商字段。

目录

- Alauda OS 要求
  - 持久性和节点替换
    - MON/Rook 数据目录
    - OSD 数据磁盘
  - 滚动节点替换
- Huawei DCS
- Huawei Cloud Stack (HCS)
- VMware vSphere
- 裸金属
- 安装前检查清单

Alauda OS 要求

持久性和节点替换

MON/Rook 数据目录

当 Ceph 集群使用主机路径 MON 存储时，MON 数据库和 Rook 管理数据存储在 `dataDirHostPath` 下。此目录必须在节点替换后保留，因此必须位于由提供商管理的持久文件系统上。不得将其放置在节点系统盘、临时磁盘或原始 OSD 设备上。如果改用 MON PVC 模板，请遵循该 CSI 支持卷的持久性和调度要求。

请选择以下配置之一：

- 将持久文件系统挂载到产品支持的默认路径 `/var/lib/rook`，并设置 `CephCluster.spec.dataDirHostPath: /var/lib/rook`。这是 Internal Mode 安装流程支持的路径。
- 仅当通过 YAML/API 配置资源时，才支持使用自定义路径，例如 `/var/cpaas/rook`。Internal Mode UI 不提供更改此路径的控件；请勿在 UI 安装流程中使用自定义路径。

```
spec:
  dataDirHostPath: /var/cpaas/rook
```

Ceph 集群创建后，将 `CephCluster.spec.dataDirHostPath` 视为不可变配置。在创建 `CephCluster` 之前，选择最终的持久挂载路径。

持久文件系统可以使用专用磁盘，也可以与其他非 OSD 持久目录共享磁盘。请根据通用 Ceph 规划要求确定其大小。

OSD 数据磁盘

OSD 数据磁盘必须由提供商的持久磁盘机制管理，并在节点替换后重新挂载到对应的工作节点。使用磁盘 UUID、URN、插槽或提供商定义的等效标识，确认替换前后使用的是同一磁盘。

OSD 数据磁盘必须使用原始块模式，并且不得同时提供包含 `dataDirHostPath` 的文件系统。提供商必须在创建 Internal Mode Ceph 安装之前，将独立的原始设备呈现给节点。提供商示例中的容量仅用于展示提供商字段，并非通用的容量规划建议。

在 Internal Mode 向导中，使用提供商提供的稳定设备路径，将提供商呈现的原始磁盘添加到 OSD 设备类，最好使用 `/dev/disk/by-id/...`。不要将 MON/Rook 文件系统磁盘用作 OSD 设备。有关设备类配置，请参阅在 Internal Mode 中部署。

提供商磁盘呈现方式	**Internal Mode** 配置	设备值
提供商呈现空白原始磁盘	将其添加到 OSD 设备类	输入稳定的提供商路径，最好使用 `/dev/disk/by-id/...`；不要使用简短的 `/dev/sdX` 名称
提供商将 MON/Rook 数据磁盘呈现为已挂载的文件系统	不要将其添加到 OSD 设备类	将其保持挂载在 `/var/lib/rook`，供 `dataDirHostPath` 使用

滚动节点替换

当 Ceph 数据磁盘由基础设施提供商池管理时，工作节点 `MachineDeployment` 必须使用固定插槽模型。`maxSurge: 0` 和 `maxUnavailable: 1` 是必需的，因为 surge VM 无法安全地使用与其替换的节点相同的主机名、IP 或持久磁盘插槽，而多个不可用的工作节点会同时移除多个存储插槽。应用控制平面控制器所需的等效先删除后创建发布设置；不要在未检查该控制器 API 的情况下，将工作节点的 `MachineDeployment` 字段复制到 `KubeadmControlPlane`：

```
strategy:
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

此外，请确保：

- 未配置有限的 `nodeDrainTimeout`。
- Kubernetes Eviction API、PDB 和 Rook disruption controller 仍处于保护链路中。
- 不会通过删除 PDB 或 finalizer，或强制清理 OSD 来强制执行替换。

对于新创建的 Internal Mode `CephCluster`，请保持 PDB 管理启用（`disruptionManagement.managePodBudgets: true`）。当集群将使用 Alauda OS 节点替换时，请勿显式禁用它。已禁用 PDB 管理的现有集群并非在所有情况下都无效，但不符合本页面中的受保护替换要求，必须在替换前进行审查。PDB 仅保护 Eviction API 路径；强制删除、直接删除 Pod 或 Machine，以及节点故障不受该保护机制约束。有关 PDB 的一般行为，请参阅使用 PodDisruptionBudgets。

替换节点之前，请确认 Ceph 集群运行状况良好，所有 OSD 均为 `up` 和 `in`，placement group 未处于降级状态，并且相关 PDB 允许驱逐。一次只替换一个存储节点。替换后，请确认提供商已重新挂载原始持久磁盘、OSD 标识未发生变化、OSD 再次处于 `up` 和 `in` 状态，并且恢复已完成，然后再继续操作。如果任何检查失败，请停止发布；不要通过清除 OSD 或移除 Ceph 节点条目来走捷径。

Huawei DCS

DCS 环境必须通过 `DCSIpHostnamePool` 声明 Rook 可用的持久磁盘。在 global 集群中配置该资源，通常位于 `cpaas-system`。有关提供商特定的基础设施、节点管理和 API 详细信息，请参阅 DCS Infrastructure Resources、DCS Node Management 和 DCS Persistent IP Hostname Pool API：

此示例中的 IP 地址、网关和 DNS 值展示了池插槽与节点网络身份之间的关系。请将其替换为环境中的值。它们是提供商节点设置，而非 Rook 特定的网络要求。

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: ceph-worker-pool
  namespace: cpaas-system
spec:
  pool:
    - ip: 192.0.2.101
      mask: "24"
      gateway: 192.0.2.1
      dns: 192.0.2.53
      hostname: worker-01
      persistentDisk:
        - slot: 0
          quantityGB: 30
          datastoreClusterName: worker-datastore-cluster
          path: /var/lib/rook
          format: xfs
        - slot: 1
          quantityGB: 300
          datastoreClusterName: worker-datastore-cluster
    - ip: 192.0.2.102
      mask: "24"
      gateway: 192.0.2.1
      dns: 192.0.2.53
      hostname: worker-02
      persistentDisk:
        - slot: 0
          quantityGB: 30
          datastoreClusterName: worker-datastore-cluster
          path: /var/lib/rook
          format: xfs
        - slot: 1
          quantityGB: 300
          datastoreClusterName: worker-datastore-cluster
    - ip: 192.0.2.103
      mask: "24"
      gateway: 192.0.2.1
      dns: 192.0.2.53
      hostname: worker-03
      persistentDisk:
        - slot: 0
          quantityGB: 30
```

```
      datastoreClusterName: worker-datastore-cluster
      path: /var/lib/rook
      format: xfs
    - slot: 1
      quantityGB: 300
      datastoreClusterName: worker-datastore-cluster
```

当 DCS 机器模板使用此池时，必须通过 `ipHostPoolRef` 引用该池。此代码片段省略了其他必需的模板字段，包括 VM 模板选择：

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: ceph-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      ipHostPoolRef:
        name: ceph-worker-pool
      # Other VM, network, and resource settings omitted
```

主要要求：

- 在 `DCSIpHostnamePool.spec.pool[].persistentDisk` 中配置 Rook 的磁盘；不要仅在 `DCSMachine` 模板中配置普通虚拟磁盘。
- DCS 通过 `(IP, slot)` 重用持久磁盘身份。节点替换后，请确认每个原始 OSD 数据磁盘仍映射到相同的持久磁盘身份。
- 为 MON/Rook 数据目录磁盘设置 `path: /var/lib/rook` 和 `format: xfs`。对于 OSD 磁盘，请省略 `path` 和 `format`，使其保持为原始块设备。
- 在 `datastoreClusterName` 和 `datastoreName` 中恰好设置一个；此示例使用 datastore 集群。DCS 通过池条目中的 `(IP, slot)` 重用持久磁盘，并在 `status.persistentDiskStatus` 中记录卷 URN。

Huawei Cloud Stack (HCS)

对于 HCS，请确认提供商为每个存储工作节点提供持久且可重新挂载的 Ceph 数据磁盘。在 global 集群中配置该资源，通常位于 `cpaas-system`。有关提供商详细信息，请参阅 HCS Infrastructure Resources、HCS Node Management 和 HCS Machine Config Pool API：

此示例中的网络设置展示了节点配置和固定 IP。请将其替换为环境中的网络参数。如果已存在可用的 ConfigPool，请保留其网络设置，并确认节点替换仍会重用相同的主机名和插槽。

HCS 提供商通过 `HCSMachineConfigPool` 为固定的机器配置声明持久磁盘：

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: hcs-acp-biz-worker-pool
  namespace: cpaas-system
spec:
  configs:
    - hostname: worker-01
      networks:
        - ipAddress: 192.0.2.101
          subnetName: worker-subnet
      persistentDisks:
        - slot: 0
          size: 30
          type: SSD
          mountPath: /var/lib/rook
          format: xfs
        - slot: 1
          size: 300
          type: SSD
    - hostname: worker-02
      networks:
        - ipAddress: 192.0.2.102
          subnetName: worker-subnet
      persistentDisks:
        - slot: 0
          size: 30
          type: SSD
          mountPath: /var/lib/rook
          format: xfs
        - slot: 1
          size: 300
          type: SSD
    - hostname: worker-03
      networks:
        - ipAddress: 192.0.2.103
          subnetName: worker-subnet
      persistentDisks:
        - slot: 0
          size: 30
          type: SSD
          mountPath: /var/lib/rook
          format: xfs
```

```
      - slot: 1
        size: 300
        type: SSD
```

`persistentDisks[]` 中的 `slot` 值必须连续且从 0 开始。它们与主机名共同构成持久磁盘身份。在此示例中，插槽 0 是 MON/Rook 数据目录磁盘，插槽 1 是原始 OSD 磁盘。请在 ConfigPool 中配置持久磁盘，而不要仅在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中配置；后者包含会随 ECS 实例一同移除的临时数据磁盘。

主要要求：

- 在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中配置持久磁盘，而不仅仅是在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中配置。
- 插槽 0 是 MON/Rook 数据目录磁盘，必须使用 `mountPath: /var/lib/rook` 和 `format: xfs`；插槽 1 是独立的原始 OSD 磁盘，不得具有文件系统挂载。
- 节点替换后必须重用 `(hostname, slot)` 身份，以便重新挂载原始持久磁盘。

相应的 `HCSMachineTemplate` 必须通过 `configPoolRef` 引用此池，并且也必须在 global 集群中创建，通常位于 `cpaas-system`：

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: ceph-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      configPoolRef:
        name: hcs-acp-biz-worker-pool
      # Other ECS, image, and network settings omitted
```

请根据目标版本，遵循提供商 API 和节点管理文档中关于 HCS 磁盘类型、挂载策略和机器模板字段的说明。

VMware vSphere

vSphere provider 使用 `VSphereMachineConfigPool` 声明持久磁盘，这些磁盘在固定机器配置下会在 VM 删除后保留。请在 global 集群中配置此资源，通常位于 `cpaas-system`。请参阅 vSphere 基础设施资源、vSphere 节点管理 和 vSphere Machine Config Pool API。`VSphereMachineTemplate` 随后引用此池：

此示例还列出了网络设置，以展示如何通过池插槽重用固定 IP。请将 IP、网关、DNS 和网络名称替换为环境中的值。这些网络字段并非 Rook 专用，但节点替换必须保留节点网络身份。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineConfigPool
metadata:
  name: ceph-worker-pool
  namespace: cpaas-system
spec:
  clusterRef:
    name: demo-cluster
  datacenter: dc-01
  configs:
    - hostname: worker-01
      network:
        primary:
          ip: 192.0.2.101/24
          gateway: 192.0.2.1
          dns:
            - 192.0.2.53
          networkName: workload-network
      persistentDisks:
        - name: rook-mon-data
          sizeGiB: 30
          datastore: ceph-datastore
          mountPath: /var/lib/rook
          fsFormat: xfs
          unitNumber: 1
          wipeFilesystem: false
        - name: rook-osd
          sizeGiB: 300
          datastore: ceph-datastore
          unitNumber: 2
          wipeFilesystem: false
    - hostname: worker-02
      network:
        primary:
          ip: 192.0.2.102/24
          gateway: 192.0.2.1
          dns:
            - 192.0.2.53
          networkName: workload-network
      persistentDisks:
        - name: rook-mon-data
          sizeGiB: 30
          datastore: ceph-datastore
```

```
        mountPath: /var/lib/rook
        fsFormat: xfs
        unitNumber: 1
        wipeFilesystem: false
      - name: rook-osd
        sizeGiB: 300
        datastore: ceph-datastore
        unitNumber: 2
        wipeFilesystem: false
  - hostname: worker-03
    network:
      primary:
        ip: 192.0.2.103/24
        gateway: 192.0.2.1
        dns:
          - 192.0.2.53
        networkName: workload-network
    persistentDisks:
      - name: rook-mon-data
        sizeGiB: 30
        datastore: ceph-datastore
        mountPath: /var/lib/rook
        fsFormat: xfs
        unitNumber: 1
        wipeFilesystem: false
      - name: rook-osd
        sizeGiB: 300
        datastore: ceph-datastore
        unitNumber: 2
        wipeFilesystem: false
```

请使用 vSphere provider 针对已安装 provider API 所记录的 machine-template-to-pool 引用方式。在应用此资源之前，provider API 参考和创建工作流必须保持一致；本页面不重复介绍 provider 专用的模板 schema。

主要要求：

- Ceph 数据磁盘不得是会随 VM 一同删除的临时磁盘。
- 必须在 `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` 中配置持久磁盘，而不仅仅是在 MachineTemplate 中将其配置为临时 `dataDisks`。
- 在 `VSphereMachineConfigPool.spec.configs[].network.primary.ip` 中配置固定 IP。当节点替换重用相同的池插槽时，provider 会继续为新 VM 使用该 IP。每个 IP 都必须在网

络中唯一可用，且不得依赖 DHCP 分配。

- 设置 `wipeFilesystem: false`，并使用固定的 `unitNumber`，以便在重建 VM 时保留数据并保持设备顺序稳定。
- 节点替换后，必须识别并重新挂载原始 OSD 数据磁盘，而不是使用容量相同的新磁盘替换它。
- `dataDirHostPath` 必须指向持久 MON/Rook VMDK 的挂载点。请勿格式化 OSD VMDK 并将其用于此路径。

裸金属

裸金属环境必须提供与 Alauda OS 节点替换模型等效的持久性。在创建 `CephCluster` 之前，请根据 provider 记录的清单和重新置备工作流进行确认：

- MON/Rook 文件系统是持久的，已挂载到所选的 `CephCluster.spec.dataDirHostPath`，并且在 Ceph Pod 启动前可写；
- 每个 OSD 设备在节点重新安装或替换期间都保持为稳定的原始设备；
- provider 不会在此过程中格式化或回收 OSD 设备；并且
- 在替换主机上可以重新发现相同的设备身份。

不要根据 `MachineInventoryPool` 成员关系推断磁盘持久性，也不要使用目标 provider CRD 中不存在的 provider 字段。本页面不定义裸金属磁盘绑定 schema。有关 provider 支持的清单和重新置备行为，请参阅裸金属 Provider API 和裸金属节点管理。

安装前检查清单

在安装 Alauda Storage 之前，请为 Alauda OS 集群确认以下事项：

- ☐ provider 将 OSD 磁盘作为持久原始块磁盘进行管理，并与节点系统磁盘分离。
- ☐ 对于受支持的 Internal Mode 流程，MON/Rook 数据目录配置为 `CephCluster.spec.dataDirHostPath: /var/lib/rook`；使用其他路径时，则通过 YAML/API 显式配置；该目录位于 provider 管理的持久文件系统上，在节点上可写，并且不使用原始 OSD 设备。
- ☐ 已确认节点替换后数据磁盘的稳定身份和重新挂载流程。

- ☐ worker MachineDeployment 使用 `maxSurge: 0` 和 `maxUnavailable: 1`。
- ☐ 未配置有限的 `nodeDrainTimeout`，为新创建的 Internal Mode CephCluster 保留 `disruptionManagement.managePodBudgets: true`，并保留 PDB、Eviction API 和 Rook disruption controller 保护路径。
- ☐ 已将持久性验证为节点替换保护，而不是将其视为 Ceph 或应用一致性备份的替代方案。
- ☐ 已查阅目标基础设施 provider 的资源配置文档。

满足这些条件后，在目标业务集群中创建 `CephCluster`，然后继续执行在 Internal Mode 中部署。上述介绍的 provider MachineTemplate、ConfigPool 和 MachineDeployment 资源属于 global 集群，通常位于 `cpaas-system` 中。Internal Mode UI 使用 `/var/lib/rook`；如果需要自定义 `CephCluster.spec.dataDirHostPath`，请改用 YAML/API 配置。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品不隶属于 LF Projects, LLC 或 The Linux Foundation，也未获得其认可或赞助。所有商标均为其各自所有者的财产，本文仅将其用于标识目的。

以内部模式部署

内部模式（UI 中的 Standard 集群类型）是最常见的 Ceph 存储部署方式。它会在当前业务集群中创建本地 Ceph 集群，并将数据副本分布到不同主机上的硬盘中，以便在单主机故障期间保持可用性。

本页面使用 创建集群向导中的主机设备。如果要改用 Local Storage Block PersistentVolumes 为 OSD 提供后端存储，请参阅使用本地存储以内部模式部署。

INFO

对于节点运行 Alauda OS 的集群，请在 global 集群中创建 provider MachineTemplate、ConfigPool 和 MachineDeployment 资源，通常位于 `cpaas-system`，并在目标业务集群中创建 StorageCluster/CephCluster。对 OSD 设备使用由 provider 管理的持久磁盘和稳定的 `/dev/disk/by-id/...` 标识。对于原位节点替换，请勿按照替换或移除存储节点中的步骤操作：不要将 OSD 标记为 `out`，不要移除节点条目，也不要清理 OSD。

目录

先决条件
准备软件包
准备基础设施
注意事项
操作步骤
部署 Alauda Container Platform Storage Essentials
（可选）安装 Alauda Container Platform Local Disk Provisioner

部署 Operator

创建集群

创建存储池

相关操作

创建 Stretch 类型集群

清理分布式存储

先决条件

准备软件包

- 下载与平台架构对应的 **Alauda Container Platform Storage Essentials** 安装软件包。
- 使用“上架软件包”机制上传 **Alauda Container Platform Storage Essentials** 安装软件包。
- 下载与平台架构对应的 **Alauda Storage** 安装软件包。
- 使用“上架软件包”机制上传 **Alauda Storage** 安装软件包。

准备基础设施

- 存储集群中至少需要 3 个节点。
- 每个节点必须至少有 1 个空白硬盘或 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。
- 如果使用以 Containerd 作为运行时组件的附加 Kubernetes 集群，请确保在集群的所有节点上，将 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值配置为 `1048576` ，以确保分布式存储成功部署。有关配置说明，请参阅修改 Containerd 配置信息。
  注意：从早于 v3.10.2 的版本升级到当前版本时，如果需要在以 Containerd 作为运行时组件的自定义 Kubernetes 集群上部署 Ceph 分布式存储，还必须在集群的所有节点上，将 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值设置为 `1048576` 。

注意事项

Internal mode 部署和 external mode 部署互斥。一个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. 登录并进入管理员页面。
2. 单击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Storage Essentials**，单击 安装，进入 安装 **Alauda Container Platform Storage Essentials** 页面。
   配置参数：

参数	推荐配置
Channel	默认 Channel 为 `stable`。
安装模式	`Cluster`：集群中的所有命名空间共享一个用于创建和管理的 Operator 实例，从而降低资源使用量。
安装位置	选择 `Recommended`，Namespace 仅支持 **acp-storage**。
升级策略	`Manual`：Operator Hub 中有新版本时，需要手动确认才能将 Operator 升级到最新版本。

2 （可选）安装 Alauda Container Platform Local Disk Provisioner

如需在添加主机设备时使用选择设备，请先安装 **Alauda Container Platform Local Disk Provisioner**。该组件会发现集群节点上的未使用磁盘，并写入 `LocalVolumeDiscoveryResult` 对象。输入设备无需使用此 Operator。

请参阅安装本地存储。如需置备 Block PersistentVolumes 并从这些卷创建 OSD，请勿继续执行此向导。在本地存储指南之后，使用使用本地存储以 Internal mode 部署。

3 部署 Operator

1. 进入管理员页面。
2. 在左侧边栏中，单击存储管理 > 分布式存储。
3. 单击立即配置。
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，请根据界面上的提示清理已部署的信息并重试，重新部署 Operator；如果希望返回分布式存储选择页面，请单击应用商店，先卸载已部署的 **rook-operator** 中的资源，然后再卸载 **rook-operator**。
   - 如需后续升级现有 Operator，请在上传新软件包前将 `Subscription` 升级策略设置为 手动。请参阅升级 Operator。

4 创建集群

1. 在创建集群向导页面中配置相关参数，然后单击右下角的创建集群按钮。

参数	说明
集群类型	选择标准（Internal mode）。
设备类别类型	设备类别是硬盘的分组；您可以根据存储需求自定义设备类别，将不同的存储内容分配到性能不同的磁盘上。 • 默认设备类别：平台会自动对集群节点中的硬盘类型进行分类。例如，创建名为 `hdd`、`ssd`、`nvme` 的设备类别。 • 自定义设备类别：为节点中特定磁盘组合自定义设备类别名称；支持添加多个设备类别。同一硬盘只能属于一个设备类别。
设备类别 - 名称	设备类别的名称。选择自定义设备类别时，设备类别不能使用以下名称：`hdd`、`ssd`、`nvme`。

参数	说明
设备类别 - 存储设备	要向设备类别添加存储设备，可以选择 `Selection Device` 和 `Input Device` 方法： • 选择设备： 从可用存储设备中选择。如果设备满足以下条件，则视为可用： • 设备类型为 disk 或 mpath • 未检测到文件系统（fsType 为空） • 容量超过 10 GiB rbd、nbd 和 dm-* 等设备不会显示在可选择的可用设备列表中。 注意：需要预先部署 Alauda Container Platform Local Disk Provisioner。 • 输入设备： 手动输入节点下的空白设备路径，建议使用稳定路径，例如 `/dev/disk/by-id/...`。对于 Alauda OS 上基础设施提供商的磁盘，请勿使用简短的 `/dev/sdX` 名称，因为替换后名称可能会发生变化。 注意：为获得最佳性能和管理效果，请使用裸磁盘，而不是单独的分区。
快照	启用后，支持创建 PVC 快照，并使用快照配置新的 PVC，以便快速备份和恢复业务数据。 如果在创建存储时未启用快照，仍可根据需要在存储集群详情页面的操作部分启用快照。 注意：使用前请确保已为当前集群部署卷快照插件。
监控告警	启用后，将提供开箱即用的监控指标采集和告警功能，请参阅监控与告警。 注意：如果此时未启用，则需要为存储监控和告警寻找替代方案。例如，在运维中心手动配置监控仪表盘和告警策略。

2. 单击高级配置，配置高级组件。

参数	说明
网络配置	• 主机网络：存储集群将使用主机网络，您应在优化参数栏中填写相关网络优化参数，例如配置 `public` 和 `cluster` 子网。如果留空，将使用默认主机子网。 注意：使用主机网络时，数据通过主机端口以未加密（明文）方式传输，可能会带来安全风险。请联系平台支持团队获取加密传输解决方案。 • 容器网络：存储集群将使用容器网络；您可以在网络管理中创建子网，并将其分配给 `rook-ceph` 命名空间。如果留空，将使用默认子网。 注意： 不支持 IPv6。 使用容器网络时，存储只能在集群内部访问。 Ceph CSI Pod 故障或重启可能导致服务中断。
优化参数	支持以 Ceph 配置文件格式填写参数；系统将根据提供的内容覆盖默认参数。 注意：首次填写或修改初始化参数后，请单击初始化参数；初始化成功后才能创建集群。
组件定点部署	可以将组件部署到指定节点；至少需要三个节点以确保最低可用性。支持进行定点部署配置的组件包括 MON、MGR、MDS、RGW。

- 当页面自动进入下一步时，表示 Ceph 集群已成功部署。
- 如果创建失败，可以单击清理已创建的信息或重试，自动清理资源并重新创建集群；也可以根据文档清理分布式存储服务资源手动清理资源。

5 创建存储池

1. 在创建存储池向导页面中，配置相关参数，然后点击右下角的创建存储池按钮。

参数	说明
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 S3 接口的存储服务，适用于大数据、备份归档、云存储等场景。
副本数量	副本数量越多，冗余性和数据安全性越高，但存储利用率会降低。通常设置为 3，以满足大多数需求。
设备类	对相同类型的设备或具有相同业务逻辑的磁盘进行统一分类，从上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。 • 如果未选择设备类，数据将在存储池中的所有设备之间随机存储。

如果存储类型为对象存储，还需要配置以下参数：

参数	说明
地域	指定存储池所在的地域。
网关类型	默认为 S3，且无法修改。
内部端口	指定集群内访问所使用的端口。
外部访问	启用或禁用外部访问将创建或销毁 Nodeport 类型的 Service。
实例数量	对象存储的资源实例数量。

- 当页面自动进入下一步时，表示存储池已成功部署。
- 如果部署失败，请根据界面提示检查核心组件，然后点击清理已创建信息并重试，重新创建存储池。

> 2. 点击创建存储池。在详情页签中，可以查看已创建存储池的信息。

相关操作

创建 **Stretch** 类型集群

详情请参阅创建 Stretch 类型集群。

清理分布式存储

详情请参阅清理分布式存储。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获得其认可或赞助。所有商标均归其各自所有者所有，此处仅用于标识目的。

使用本地存储部署 Internal Mode

此 Internal Mode 路径会在当前业务集群中创建 Ceph 集群，并使用来自 **Local Storage** 的块 PersistentVolume 为 OSD 提供存储。在 Distributed Storage 向导中配置集群类型、快照、监控和高级选项，然后切换到 YAML 并替换 `spec.storage`，以便 Rook 使用 Local Storage 卷，而不是主机设备。

不要在同一个 `CephCluster` 上混用主机设备（`spec.storage.nodes`）和 `storageClassDeviceSets`。

目录

先决条件

操作步骤

1. 安装 Alauda Container Platform Local Disk Provisioner 并预配块卷
2. 在向导中部署 Alauda Storage operator
3. 创建集群：填写向导，然后编辑 YAML 存储配置

扩展设备类中的 OSD

1. 配置额外的 Block 卷
2. 确认卷处于 Available 状态
3. 增加现有设备集的数量

添加设备类

1. 为新磁盘配置 Block 卷
2. 确认新卷
3. 添加 storageClassDeviceSets 项

4. 验证新的 OSD
5. 在池上选择新类

相关信息

先决条件

- 至少有三个节点运行 OSD。
- 这些节点各自至少有一个未使用、未挂载且没有分区的磁盘。
- 下载并上传 **Alauda Container Platform Storage Essentials**、**Alauda Storage** 和 **Alauda Container Platform Local Disk Provisioner** 的安装包。

操作步骤

1

1. 安装 Alauda Container Platform Local Disk Provisioner 并预配块卷

按照安装 Local Storage中的说明安装 operator。

确认 operator 正在运行：

```
kubectl -n acp-storage get csv | grep local-storage
```

示例输出：

```
local-storage-operator.v4.5.0   Alauda Container Platform Local Disk
Provisioner   Succeeded
```

为每个 OSD 预配一个块 PersistentVolume，并使用 `LocalVolume` 明确指定每个磁盘路径。将主机名替换为 `kubectl get node` 值。从发现本地磁盘（`status.discoveredDevices[].deviceID`）复制每个 `/dev/disk/by-id/...` 路

径。不要使用 `/dev/sdX` 或 `/dev/disk/by-path/...`。这些名称可能会在节点重启后发生变化。生产集群必须使用 `/dev/disk/by-id/...`。

```
cat << EOF | kubectl apply -f -
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: local-block-disks
  namespace: acp-storage
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-01
              - worker-02
              - worker-03
  storageClassDevices:
    - storageClassName: localblock
      volumeMode: Block
      devicePaths:
        - /dev/disk/by-id/<wwn-worker-01>
        - /dev/disk/by-id/<wwn-worker-02>
        - /dev/disk/by-id/<wwn-worker-03>
EOF
```

检查 `LocalVolume` 和 StorageClass：

```
kubectl -n acp-storage get localvolume local-block-disks
kubectl get sc localblock
```

示例输出：

```
NAME                AGE
local-block-disks   45s

NAME          PROVISIONER                     RECLAIMPOLICY   VOLUMEBIN
DINGMODE      ALLOWVOLUMEEXPANSION   AGE
localblock    kubernetes.io/no-provisioner    Delete          WaitForFi
rstConsumer   false                  45s
```

检查块 PersistentVolume 是否为 `Available`。数量至少应与稍后要设置的 OSD `count` 一样多（此示例中为三个）：

```
kubectl get pv -o custom-columns=NAME:.metadata.name,STATUS:.status.p
hase,SC:.spec.storageClassName,MODE:.spec.volumeMode,CAPACITY:.spec.c
apacity.storage
```

示例输出：

```
NAME                STATUS      SC           MODE    CAPACITY
local-pv-1cec77cf   Available   localblock   Block   1Ti
local-pv-2ef7cd2a   Available   localblock   Block   1Ti
local-pv-3fa1c73    Available   localblock   Block   1Ti
```

如果缺少卷，请确认 `by-id` 路径位于所选节点上，且磁盘未被使用。如果发现结果中没有列出 `deviceID`，则该磁盘没有稳定的 `/dev/disk/by-id` 标识，不适合作为生产环境中的 LocalVolume 路径。如需使用基于筛选条件的预配方式，而非显式指定路径，请参阅使用 LocalVolumeSet 进行预配。

2 2. 在向导中部署 Alauda Storage operator

1. 登录并前往 **Administrator** 页面。
2. 在左侧边栏中，单击 **Storage Management** > **Distributed Storage**。
3. 单击 **Configure Now**。
4. 如果尚未安装 **Alauda Container Platform Storage Essentials**，向导会先安装它。使用与使用 Internal Mode 部署相同的 OperatorHub 值：通道 `stable`、安装模式 `Cluster`、命名空间 **acp-storage**、升级策略 `Manual`。

5. 在 **Install Operator** 向导页面上，单击 **Alauda Storage** 对应的 **Start Deployment**（或 **Deploy Operator**）。

   - 页面进入 **Create Cluster** 后，即表示已安装 Operator。
   - 如果部署失败，请按照屏幕提示操作，单击 **Clean Up**，然后重新部署 Operator。

3 3. 创建集群：填写向导，然后编辑 YAML 存储配置

1. 在 **Create Cluster** 向导页面上，将 **Cluster Type** 设置为 **Standard**（Internal Mode）。
2. 配置除主机存储设备以外的所有内容：

   - 根据需要配置 **Snapshot** 和 **Monitoring Alarm**。
   - **Advanced Configuration**：配置网络、优化参数和组件放置。使用定点部署时，将 MON/MGR/MDS/RGW 放置在至少三个节点上。

   不要添加 **Device Class** 主机设备（**Selection Device** 或 **Input Device**）。这些字段会写入 `spec.storage.nodes`，与由 PVC 提供存储的 OSD 冲突。
3. 将页面从 **Form** 切换到 **YAML**。编辑器会显示向导将要创建的 `CephCluster`。保留自动生成的 `cephVersion`、`network`、`mon`、`monitoring` 和其他非存储字段。
4. 替换整个 `spec.storage` 部分。如果存在 `spec.storage.nodes`，请将其删除。使用步骤 1 中的 StorageClass（此示例中为 `localblock`）。将 `count` 设置为要使用的 `Available` 块 PersistentVolume 数量。将 `crushDeviceClass` 设置为这些 OSD 必须使用的类名称。名称可以是 `ssd`、`hdd`、`nvme`，也可以是创建池时要选择的任意自定义名称。

```yaml
spec:
  storage:
    storageClassDeviceSets:
      - name: local-osds
        count: 3 (1)
        portable: false (2)
        tuneFastDeviceClass: true (3)
        resources: (4)
          limits:
            cpu: "2"
            memory: 5Gi
          requests:
            cpu: "1"
            memory: 5Gi
        volumeClaimTemplates:
          - metadata:
              name: data
              annotations:
                crushDeviceClass: ssd (5)
            spec:
              storageClassName: localblock (6)
              volumeMode: Block
              accessModes:
                - ReadWriteOnce
              resources:
                requests:
                  storage: 1Gi (7)
```

(1) OSD 数量。每个 OSD 绑定一个块 PersistentVolume。设置的数量不得超过 `Available` 卷的数量。

(2) Local Storage 必须设置为 `false`。本地卷无法迁移到其他节点。

(3) 根据物理介质设置，而非根据 CRUSH 类名称设置。如果底层磁盘为 SSD、NVMe 或云磁盘，请设置 `true`。仅当底层磁盘为 HDD 时，才设置 `false`。

(4) OSD Pod 的 CPU 和内存。必填。

(5) 这些 OSD 的 Ceph CRUSH 设备类。必填。使用创建池时要选择的任意类名称。名称不限于 `hdd`、`ssd` 或 `nvme`。之后创建池时会选择此类。OSD 创建后更改类需要执行 `spec.storage.allowDeviceClassUpdate: true`。

(6) 由 `LocalVolume` 创建的 StorageClass。必须为 Block。

7 PVC 请求，仅用于绑定一个 `Available` 块 PersistentVolume。Kubernetes 要求填写此字段；请勿省略。OSD 使用整个磁盘（例如 `1Ti`），而不是此请求。设置的值必须小于或等于匹配的最小 PersistentVolume。

5. 单击 **Create Cluster**，并等待向导完成。
6. 验证 OSD 和 Rook 创建的 PVC：

```
kubectl -n rook-ceph get pod -l app=rook-ceph-osd
kubectl -n rook-ceph get pvc
kubectl -n rook-ceph get cephcluster ceph-cluster -o jsonpath='{.s
tatus.ceph.health}{"\n"}'
```

使用 `1Gi` 请求绑定 1Ti 本地卷后的 PVC 示例输出：

```
NAME                      STATUS   VOLUME              CAPACITY   ACC
ESS MODES   STORAGECLASS   AGE
local-osds-data-0-0-0   Bound    local-pv-1cec77cf  1Ti        RWO
localblock     2m
local-osds-data-1-0-0   Bound    local-pv-2ef7cd2a  1Ti        RWO
localblock     2m
local-osds-data-2-0-0   Bound    local-pv-3fa1c73   1Ti        RWO
localblock     2m
```

`CAPACITY` 是 PersistentVolume 的大小（磁盘大小）。PVC 上的 `.spec.resources.requests.storage` 保持为模板中的请求值（此示例中为 `1Gi`）。
等待三个 OSD Pod 均处于运行状态，且健康状态为 `HEALTH_OK`；如果只有预期的启动提示，则健康状态可以是 `HEALTH_WARN`。

扩展设备类中的 OSD

使用此操作步骤添加 OSD，使其与现有 `storageClassDeviceSets` 条目保持相同的 CRUSH 设备类。

1 1. 配置额外的 Block 卷

使用相同的 StorageClass 配置额外的 Block PersistentVolumes。请参阅指南。使用 `/dev/disk/by-id/...` 路径。

2 2. 确认卷处于 Available 状态

确认新的 PersistentVolumes 处于 `Available` 状态。

3 3. 增加现有设备集的数量

编辑集群，并将该设备集上的 `spec.storage.storageClassDeviceSets[].count` 增加到与新总数匹配。只更改 `count`。不要替换设备集的其余部分，也不要更改 `crushDeviceClass`。

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

不要降低 `count` 来移除 OSD。本页不支持通过此方式缩减规模。

添加设备类

使用此操作步骤在集群创建后引入新的 CRUSH 设备类。不要只增加现有设备集上的 `count`。不要使用 Web 控制台中的 **Device Classes** 选项卡；该选项卡会写入 `spec.storage.nodes`，并与基于 PVC 的 OSD 冲突。

1 1. 为新磁盘配置 Block 卷

按照安装操作步骤中的第 1 步创建另一个 `LocalVolume`。使用 `/dev/disk/by-id/...` 路径。当磁盘属于不同的介质类型时，为 `LocalVolume` 使用唯一的 StorageClass 名称，以避免新的 OSD 绑定到错误的卷。

```
cat << EOF | kubectl apply -f -
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: local-block-disks-archive
  namespace: acp-storage
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-01
              - worker-02
              - worker-03
  storageClassDevices:
    - storageClassName: localblock-archive
      volumeMode: Block
      devicePaths:
        - /dev/disk/by-id/<wwn-archive-worker-01>
        - /dev/disk/by-id/<wwn-archive-worker-02>
        - /dev/disk/by-id/<wwn-archive-worker-03>
EOF
```

2 2. 确认新卷

确认新的 Block PersistentVolumes 处于 `Available` 状态，并使用新的 StorageClass：

```
kubectl get pv -o custom-columns=NAME:.metadata.name,STATUS:.status.phase,SC:.spec.storageClassName,MODE:.spec.volumeMode,CAPACITY:.spec.capacity.storage
```

3 3. 添加 storageClassDeviceSets 项

编辑集群，并在 `spec.storage.storageClassDeviceSets` 下追加新项。保持所有现有设备集不变。

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

```
spec:
  storage:
    storageClassDeviceSets:
      - name: local-osds
        count: 3
        portable: false
        tuneFastDeviceClass: true
        resources:
          limits:
            cpu: "2"
            memory: 5Gi
          requests:
            cpu: "1"
            memory: 5Gi
        volumeClaimTemplates:
          - metadata:
              name: data
              annotations:
                crushDeviceClass: ssd
            spec:
              storageClassName: localblock
              volumeMode: Block
              accessModes:
                - ReadWriteOnce
              resources:
                requests:
                  storage: 1Gi
      - name: local-osds-archive (1)
        count: 3 (2)
        portable: false
        tuneFastDeviceClass: false (3)
        resources:
          limits:
            cpu: "2"
            memory: 5Gi
          requests:
            cpu: "1"
            memory: 5Gi
        volumeClaimTemplates:
          - metadata:
              name: data
              annotations:
                crushDeviceClass: archive (4)
```

```
    spec:
      storageClassName: localblock-archive (5)
      volumeMode: Block
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
```

(1) 新设备集。`name` 在 `storageClassDeviceSets` 中必须唯一。

(2) 新类中的 OSD 数量。不要将其设置为高于新 StorageClass 上 `Available` 卷的数量。

(3) 根据新磁盘的物理介质设置，而不是根据 CRUSH 类名称设置。当这些磁盘为 SSD、NVMe 或云磁盘时，设置为 `true`。仅当这些磁盘为 HDD 时，设置为 `false`。

(4) 新的 CRUSH 设备类。必需。使用创建池时要选择的任意类名称。名称不限于 `hdd`、`ssd` 或 `nvme`。

(5) 由新的 `LocalVolume` 创建的 StorageClass。必须为 Block。

4 4. 验证新的 OSD

等待新的 OSD pod 处于运行状态，并且集群运行状况为 `HEALTH_OK`；对于预期的启动提示，仅允许为 `HEALTH_WARN`：

```
kubectl -n rook-ceph get pod -l app=rook-ceph-osd
kubectl -n rook-ceph get pvc
kubectl -n rook-ceph get cephcluster ceph-cluster -o jsonpath='{.stat
us.ceph.health}{"\n"}'
```

5 5. 在池上选择新类

创建或更新池，以便工作负载可以选择新类。请参阅管理存储池。

相关信息

- 以内部模式部署会在 Web 控制台中使用主机设备，而不会替换 `spec.storage`。

- 添加设备/设备类介绍通过控制台使用主机设备的操作路径。如果 OSD 使用 Local Storage Block PersistentVolumes，请勿使用此方法。
- 如果需要移除集群，请参阅清理分布式存储。

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获其认可或赞助。所有商标均为其各自所有者的财产，在此仅用于标识目的。

以外部模式部署

外部模式用于使用现有的 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。您可以连接到平台上另一个业务集群中的分布式存储，也可以集成外部 Ceph 环境。

目录

- 注意事项
- 前提条件
  - 准备软件包
  - 准备存储
  - 开放端口
  - 获取身份验证信息（外部 Ceph）
- 操作步骤
  - 部署 Alauda Container Platform Storage Essentials
  - 配置外部模式
- 后续操作

注意事项

内部模式部署和外部模式部署互斥。对于一个集群，只能选择一种部署方式。

前提条件

准备软件包

- 下载与平台架构对应的 **Alauda Container Platform Storage Essentials** 安装软件包。
- 使用“上架软件包”机制上传 **Alauda Container Platform Storage Essentials** 安装软件包。
- 下载与平台架构对应的 **Alauda Storage** 安装软件包。
- 使用“上架软件包”机制上传 **Alauda Storage** 安装软件包。

准备存储

选择以下其中一项：

- 已在其他业务集群中部署分布式存储，并创建了存储池。请记录存储池名称，以便后续集成使用。
- 已在平台外部创建外部 Ceph 存储（版本 ≥ 14.2.3）及存储池。请记录存储池名称，以便后续集成使用。

开放端口

目标 **IP**	目标端口	源 **IP**	源端口
Ceph 节点的 IP	3300、6789、6800-7300、7480	业务集群中所有节点的 IP	任意

获取身份验证信息（外部 **Ceph**）

如果准备的存储是外部 Ceph 存储，则必须使用以下命令获取身份验证信息。

参数	获取方式
FSID	`ceph fsid`

参数	获取方式
MON 组件信息	`ceph mon dump` 必须采用 {name= IP} 格式，例如 *a=192.168.100.100:6789* 。
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。使用其他业务集群的分布式存储时，操作过程类似。

1 部署 Alauda Container Platform Storage Essentials

1. 登录，然后进入管理员页面。
2. 单击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Storage Essentials**，单击安装，进入 安装 **Alauda Container Platform Storage Essentials** 页面。
   配置参数：

参数	推荐配置
通道	默认通道为 `stable`。
安装模式	`Cluster`：集群中的所有命名空间共享一个 Operator 实例，用于创建和管理，从而降低资源使用量。
安装位置	选择 `Recommended`，Namespace 仅支持 **acp-storage**。

参数	推荐配置
升级策略	`Manual`：Operator Hub 中有新版本时，需要手动确认是否将 Operator 升级到最新版本。

2 配置外部模式

1. 在左侧导航栏中，单击存储管理 > 分布式存储。
2. 单击接入存储。
3. 在接入配置向导页面中，选择外部模式，然后选择 **External Ceph**。

参数	描述
快照	启用后，支持创建 PVC 快照，并使用快照配置新的 PVC，以便快速备份和恢复业务数据。 如果接入存储时未启用快照，之后仍可根据需要在存储集群详情页面的运维部分启用。 注意：使用前，请确保已为当前集群部署卷快照插件。
网络配置	• 主机网络：此集群中的计算组件将使用主机网络访问存储集群。 • 容器网络：此集群中的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，并将该子网分配给 `rook-ceph` 命名空间。如果留空，则使用默认子网。
其他参数	填写先决条件中获取的外部 Ceph 身份验证参数。

4. 在创建 **Storage Class** 向导页面中，完成配置并单击接入。

参数	描述
类型	根据上述创建的存储池类型，默认对应的 Storage Class 为： • 文件存储：CephFS File Storage • 块存储：CephRBD Block Storage

参数	描述
回收策略	持久卷的回收策略。 • 删除：删除持久卷声明时，绑定的持久卷也会被删除。 • 保留：即使删除持久卷声明，绑定的持久卷仍会保留。
项目分配	可以使用此类型存储的项目。 如果目前没有项目需要此类型的存储，可以暂不分配项目，之后再更新。

5. 等待约 1-5 分钟，直至集成成功。

后续操作

- 创建 Storage Class：CephFS File Storage、CephRBD Block Storage
- 使用上述 Storage Class 创建持久卷声明的开发人员，可以通过卷快照和扩展功能扩展使用场景。

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中执行相关操作。

Ceph 是 Red Hat, Inc. 或其在美国及其他国家/地区的子公司的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获得其认可或赞助。所有商标均归其各自所有者所有，仅用于标识目的。

创建 Stretch 类型集群

Stretch 集群可以跨越两个地理位置不同的区域，为存储基础设施提供灾难恢复能力。当发生灾难导致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：Alpha 功能免责声明
> 本文档介绍的功能目前处于 **Alpha** 阶段，仅严格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，您确认并接受以下技术限制：
>
> - 不得用于生产环境：此功能尚未经过全面的系统级验证。在生产环境中部署该功能具有较高的流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，也不提供紧急修复程序。
> - 重大变更与弃用：API 端点、配置架构和核心处理逻辑可能会在未来版本中发生不向后兼容的变更。此功能也可能在不事先通知的情况下被完全弃用。
> - 不提供平滑升级路径：我们不提供版本间的升级脚本或数据迁移工具。升级到后续版本通常需要彻底清除现有资源并重新执行部署。由此产生的任何状态或数据丢失均由用户自行承担。

目录

术语

典型部署方案

组件说明

灾难恢复说明

约束和限制
前提条件
操作步骤
节点标记
创建存储服务
相关操作
在内部模式下部署
清理分布式存储

术语

术语	说明
仲裁可用区	通常位于不承担主要工作负载的独立区域，重点是维护集群一致性，主要用于主数据中心发生故障或出现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，承担运行负载和数据存储任务，与仲裁区域共同构成完整的高可用存储系统。

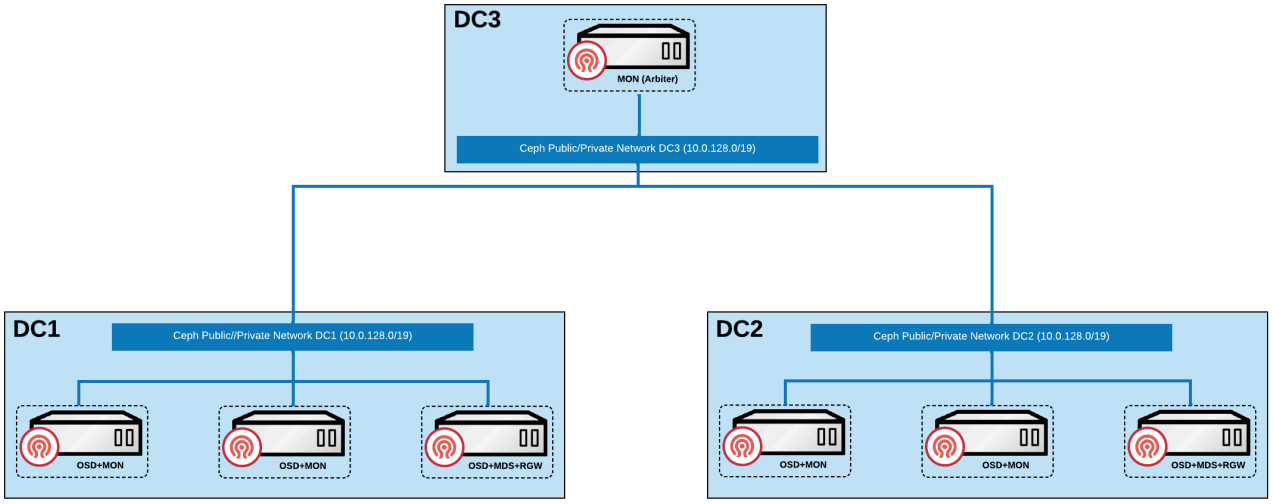
典型部署方案

以下内容介绍 Stretch 集群的典型部署方案，以及组件说明和灾难恢复原理。

组件说明

节点需要分布在三个可用区中，包括两个数据可用区和一个仲裁可用区。

- 两个数据可用区都需要完整部署所有核心 Ceph 组件（MON、OSD、MGR、MDS、RGW），并且每个数据可用区必须配置两个 MON 实例以实现高可用性。当同一数据可用区中的两个 MON 实例均不可用时，系统将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

- 当一个数据可用区完全发生故障时，Ceph 集群将自动进入降级状态并触发告警通知。系统会将存储池中的最小副本数（min_size）从默认值 2 调整为 1。由于另一个数据可用区仍保持双副本，集群仍可用。当发生故障的数据可用区恢复后，系统将自动执行数据同步并恢复到健康状态；如果故障无法修复，建议将其替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到仲裁可用区时，仲裁可用区将根据预设策略在两个数据可用区之间进行仲裁，选择状态较好的一方继续作为主数据区提供服务。

约束和限制

- 存储池限制：不支持纠删码存储池，仅可使用副本机制保护数据。
- 设备分类限制：不支持设备类别功能，无法根据设备特征对存储进行分层。
- 区域部署限制：仅支持两个数据可用区，数据可用区数量不得超过两个。
- 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，以确保数据均衡分布。
- 存储介质要求：仅允许使用全闪存（All-Flash）OSD 配置，以最大限度缩短连接恢复后的恢复时间，并尽可能降低潜在的数据丢失风险。
- 网络延迟要求：两个数据可用区之间的 RTT（往返时间）不得超过 10ms，且仲裁可用区必须满足 ETCD 规范中的延迟要求，以确保仲裁机制的可靠性。

前提条件

请提前将集群中的全部或部分节点划分到三个可用区中，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，仲裁可用区至少需要有一个节点，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控制平面节点。
- 确保至少有 4 个计算节点均匀分布在 2 个数据可用区中，每个数据可用区至少配置 2 个计算节点。
- 尽量确保两个数据可用区中的节点数量和磁盘配置保持一致。

操作步骤

1 节点标记

1. 访问管理员。
2. 在左侧导航栏中，单击集群管理 > 集群。
3. 单击相应的集群名称，进入集群概览页面。
4. 切换到节点页签。
5. 根据前提条件中的规划，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

本文档仅介绍与内部模式部署不同的参数；对于其他参数，请参阅在内部模式下部署。

创建集群

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

创建存储池

参数	描述
副本数量	默认为 4。
实例数量	当存储类型为对象存储时，为确保可用性，实例数量最少为 2，最多为 5。

相关操作

在内部模式下部署

有关详情，请参阅在内部模式下部署。

清理分布式存储

有关详情，请参阅清理分布式存储。

升级

Alauda Storage 的升级通过 OLM 交付。在当前频道中发布新软件包会升级 Operator 及与其绑定的 Ceph 镜像。迁移到下一个 Ceph 主版本需要切换 Operator 频道。

升级 Operator

开始之前

操作步骤

如果详情页按钮仍处于禁用状态

相关主题

将 Ceph 从 Squid (v19) 升级到 Tentacle (v20)

要求

操作步骤

如果升级未取得进展

相关主题

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

升级 Operator

在现有集群中将新版本的 **Alauda Storage** Operator 发布到 OperatorHub 时，请使用此操作步骤。

要将现有集群从 Ceph Squid (v19) 升级到 Ceph Tentacle (v20)，请参阅 将 Ceph 从 Squid (v19) 升级到 Tentacle (v20)。此升级会更改 Operator channel，单独发布软件包不会触发此升级。

批准该 Operator 软件包即表示同时确认 Operator 控制平面和与该软件包绑定的 Ceph 镜像。请勿在控制台中修补 `CephCluster.spec.cephVersion.image`，也无需在详情页等待数据平面完成。

推荐使用 Alauda Storage 集群详情页上的 升级到 按钮。该按钮只会批准 `rook-ceph/rook-operator` 的待处理 `InstallPlan`。按钮不可用时，请使用 `kubectl`。

> **WARNING**
>
> 上传新软件包之前，请确认 Rook Operator `Subscription` 使用 手动批准。如果 `installPlanApproval` 为 `Automatic`，新版本出现在 catalog 中后，OLM 会立即开始升级 Operator。

目录

开始之前

操作步骤

1. 确认手动批准

2. 上传新的 Operator 软件包

3. 批准 Operator 软件包

   推荐：Ceph 详情页

   后备方案：kubectl

4. 验证 Operator 和数据平面

如果详情页按钮仍处于禁用状态

相关主题

开始之前

- 确定 Rook Operator 的 namespace。Distributed Storage 向导创建的默认 namespace 为 `rook-ceph`。
- 下载与集群架构匹配的新 **Alauda Storage** Operator 软件包。
- 对于 内部模式，请确认 `CephCluster` 为 `Ready`，且 `status.ceph.health` 为 `HEALTH_OK`。本地运行状况不是 `HEALTH_OK` 时，详情页按钮会保持禁用状态。
- 对于 外部模式（`spec.external.enable: true`），同一按钮只会升级 Operator，不会写入本地 Ceph 镜像。
- 请勿将生成的 CSV manifest 编辑为持久化自定义内容。后续的 OLM reconcile 可能会覆盖这些编辑。请参阅架构。

操作步骤

1. 确认手动批准

```
kubectl -n rook-ceph get subscription rook-operator \
  -o jsonpath='{.spec.installPlanApproval}{"\n"}'
```

输出必须为 `Manual`。如果不是，请在上传软件包之前进行设置：

```
kubectl -n rook-ceph patch subscription rook-operator --type merge \
  -p '{"spec":{"installPlanApproval":"Manual"}}'
```

你也可以在 **Marketplace** > **OperatorHub** > **Alauda Storage** 中将 升级策略 设置为 手动。

2. 上传新的 Operator 软件包

使用 `violet` 上传新的 **Alauda Storage** 软件包，然后确认 OperatorHub 中显示目标版本。

按照上架软件包以及 Operator 中的 OperatorHub 操作步骤执行。

上传成功后，OperatorHub 会显示新版本可用。在 `installPlanApproval` 为 `Manual` 之前，请勿批准该版本。

3. 批准 Operator 软件包

OLM 会为新的 CSV 创建一个 `InstallPlan`。批准该计划即表示接受该 Operator 版本以及与该软件包绑定的 Ceph 镜像。

推荐：**Ceph** 详情页

1. 登录并切换到 管理员 视图。
2. 打开 Alauda Storage 集群详情页。
3. 按钮启用后，会显示 升级到，后跟目标 CSV 版本。该版本来自待处理 InstallPlan 的目标 CSV（`spec.version`，或 CSV 名称中的版本部分，例如 `rook-ceph.v4.4.1` → `4.4.1`）。
4. 单击该按钮。控制台会将该 InstallPlan `spec.approved` 设置为 `true`。你可以离开此页面。

以下情况会导致按钮保持禁用：

- 当前用户无法更新 `rook-ceph` 中的 `InstallPlan` 对象。
- 集群处于内部模式，且 `status.ceph.health` 不是 `HEALTH_OK`。
- `Subscription.status.installPlanRef` 未引用待处理且未批准的 InstallPlan。如果 OLM 尚未创建该 InstallPlan，仅 catalog 中有较新的 CSV 还不够。

批准的计划正在安装时，按钮可能会显示升级正在进行。待处理计划消失后，按钮会显示没有可用升级。

该按钮不会写入 `spec.cephVersion.image`，也不会解析 PackageManifest 镜像。

后备方案：**kubectl**

```
kubectl -n rook-ceph get subscription rook-operator \
  -o jsonpath='{.status.state}{"\n"}{.status.installedCSV}{"\n"}{.status.
currentCSV}{"\n"}{.status.installPlanRef.name}{"\n"}'
```

`UpgradePending` 和 `InstallPlanPending=True` 表示该计划正在等待批准。请仅批准引用的计划：

```
PLAN="$(kubectl -n rook-ceph get subscription rook-operator -o jsonpath
='{.status.installPlanRef.name}')"
kubectl -n rook-ceph patch installplan "$PLAN" --type merge \
  -p '{"spec":{"approved":true}}'
```

4. 验证 **Operator** 和数据平面

等待新的 CSV 为 `Succeeded`，并且 `Subscription` 为 `AtLatestKnown`：

```
kubectl -n rook-ceph get csv
kubectl -n rook-ceph get subscription rook-operator \
  -o jsonpath='{.status.state}{"\t"}{.status.installedCSV}{"\n"}'
```

对于内部模式，安装的 CSV 为 `Succeeded` 后，平台会将唯一 CSV `relatedImages` 条目中 `name` 为 `ceph` 的值写入 `CephCluster.spec.cephVersion.image`。如果该镜像已经是正在运行的值，Ceph daemon 不会滚动更新。如果软件包没有唯一的 `name=ceph` 镜像，平台不会猜测其他镜像。

确认：

```
kubectl -n rook-ceph get cephcluster <cephcluster-name> \
  -o jsonpath='phase={.status.phase} health={.status.ceph.health} image=
{.spec.cephVersion.image}{"\n"}'
```

- `phase=Ready` 和 `health=HEALTH_OK`。
- `spec.cephVersion.image` 与名为 `ceph` 的唯一 CSV `relatedImages` 条目匹配。
- 现有 PVC 保持为 `Bound`。

对于外部模式，跳过本地镜像检查。现有消费者卷应保持为 `Bound`。

如果详情页按钮仍处于禁用状态

1. 确认用户可以在 `rook-ceph` 中更新 `installplans`。
2. 对于内部模式，确认 `status.ceph.health` 为 `HEALTH_OK`。
3. 检查 Subscription 和其引用的 InstallPlan：

   ```
   kubectl -n rook-ceph get subscription rook-operator
   kubectl -n rook-ceph get installplan
   ```

   如果 catalog 中已有更新的 CSV，但没有待处理且尚未批准的 InstallPlan，请检查 `startingCSV` 是否锁定了 Subscription。不要尝试从详情页修复该锁定。
4. 如果 InstallPlan 已获批准，但 Ceph 镜像未更改，请确认已安装的 CSV 为 `Succeeded`，并且 `relatedImages` 中恰好包含一个名为 `ceph` 的条目。

不要删除 `CephCluster` 或 PV 对象来取消升级。

相关主题

- 架构 — OLM 升级概述
- 将 Ceph 从 Squid (v19) 升级到 Tentacle (v20) — 切换到下一个 Ceph 主版本
- 上架软件包 — 上架 Operator 软件包
- Operator — `Subscription`、`InstallPlan` 和批准策略

- 在内部模式下部署 — 安装时的 升级策略

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。
Rook® 是 The Linux Foundation 的注册商标。
Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，未获得其认可或赞助。所有商标均为其各自所有者的财产，在此仅用于标识目的。

将 Ceph 从 Squid (v19) 升级到 Tentacle (v20)

每个 Ceph 主版本都通过 `rook-operator` 软件包中各自的 Operator channel 发布：

Channel	Rook	Ceph
`alpha`	1.19	19.2 Squid
`stable-1.20`	1.20	20.2 Tentacle

现有集群会继续使用当前的 channel。将 Tentacle 软件包发布到 catalog 不会使集群迁移到该版本。您可以通过切换 `Subscription` channel 来选择升级时间。如果不切换 channel，集群将继续运行 Ceph Squid。

WARNING

从 Squid 升级到 Tentacle 后无法回滚。Ceph daemon 运行 Tentacle 后，请勿将 `Subscription` 切换回 `alpha`，也不要将 `CephCluster.spec.cephVersion.image` 设置为 Squid 镜像。

目录

要求

平台

节点内核

集群状态

操作步骤

1. 检查当前状态
2. 切换 Subscription channel
3. 批准 InstallPlan
4. 等待升级完成
5. 验证

如果升级未取得进展

相关主题

要求

平台

- 集群运行的是 ACP 4.5 或更高版本。Rook 1.20 要求 Kubernetes 1.31 或更高版本，因此请在升级 Alauda Storage 前先升级平台。
- 包含 `stable-1.20` channel 的 `rook-operator` 软件包已上架到 catalog。请参阅上架软件包。

节点内核

运行 Ceph daemon 或挂载 Ceph 卷的所有节点都必须运行 Linux 内核 5.11 或更高版本。

检查每个节点的内核版本：

```
kubectl get nodes -o custom-columns=NODE:.metadata.name,KERNEL:.status.nodeInfo.kernelVersion,OS:.status.nodeInfo.osImage
```

> **NOTE**

此升级已在 Red Hat Enterprise Linux 9.6、内核版本为 `5.14.0-570.12.1.el9_6` 的环境中，对 RBD 和 CephFS 内核客户端进行了验证。

集群状态

- `CephCluster` 为 `Ready`，且 `status.ceph.health` 为 `HEALTH_OK`。
- 所有使用 Ceph storage class 的 PVC 均为 `Bound`。
- `rook-operator` `Subscription` 使用 **Manual** 审批。请参阅确认 Manual 审批。

操作步骤

1. 检查当前状态

```
kubectl -n rook-ceph get subscription rook-operator \
  -o jsonpath='channel={.spec.channel} approval={.spec.installPlanApproval} installed={.status.installedCSV}{"\n"}'
kubectl -n rook-ceph get cephcluster <cephcluster-name> \
  -o jsonpath='phase={.status.phase} health={.status.ceph.health} version={.status.version.version}{"\n"}'
```

预期结果为：`channel=alpha`、`approval=Manual`、`phase=Ready`、`health=HEALTH_OK`，以及一个 `19.2.x` 版本。

确认 catalog 提供 `stable-1.20` channel：

```
kubectl -n cpaas-system get packagemanifest rook-operator \
  -o jsonpath='{range .status.channels[*]}{.name}={.currentCSV}{"\n"}{end}'
```

2. 切换 Subscription channel

```
kubectl -n rook-ceph patch subscription rook-operator --type merge \
  -p '{"spec":{"channel":"stable-1.20"}}'
```

OLM 会为 `stable-1.20` 中最新的 CSV 创建一个 `InstallPlan`。中间版本会被跳过。

3. 批准 InstallPlan

按照批准 Operator 软件包中的说明，从 Ceph 详情页面或使用 `kubectl` 批准待处理的计划。批准该计划后，Ceph 主版本升级将开始。

```
PLAN="$(kubectl -n rook-ceph get subscription rook-operator -o jsonpath
='{.status.installPlanRef.name}')"
kubectl -n rook-ceph get installplan "$PLAN" \
  -o jsonpath='{.spec.clusterServiceVersionNames}{"\n"}'
kubectl -n rook-ceph patch installplan "$PLAN" --type merge \
  -p '{"spec":{"approved":true}}'
```

批准前，确认该计划的目标为 `stable-1.20` CSV。

4. 等待升级完成

监视 daemon 版本，直到每个 daemon 都报告 `20.2.x`：

```
kubectl -n rook-ceph get deployments -l rook_cluster=rook-ceph \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.labels.c
eph-version}{"\n"}{end}'
```

在包含三个节点和六个 OSD 的集群上进行验证时，升级约在 7 分钟内完成。OSD 数量越多，所需时间越长。

5. 验证

```
kubectl -n rook-ceph get csv
kubectl -n rook-ceph get cephcluster <cephcluster-name> \
  -o jsonpath='phase={.status.phase} health={.status.ceph.health} version
={.status.version.version}{"\n"}'
kubectl get pvc -A | grep -v Bound
```

- 新 CSV 为 `Succeeded`。
- `phase=Ready`、`health=HEALTH_OK` 以及该版本为 `20.2.x`。
- 没有 Ceph 支持的 PVC 处于 `Bound` 以外的状态。
- 现有工作负载可以读取其数据，并且可以创建和挂载新的 PVC。

如果 toolbox 可用，请确认所有 daemon 都运行 Tentacle：

```
kubectl -n rook-ceph exec deploy/rook-ceph-tools -- ceph versions
```

如果升级未取得进展

- 切换 **channel** 后没有 **InstallPlan**：确认 `PackageManifest` 中存在 `stable-1.20` channel，并且 `Subscription` 没有用于固定旧版本的 `startingCSV`。
- **Operator** 已升级，但 **Ceph daemon** 未滚动升级：确认新 CSV 为 `Succeeded`，并且其 `relatedImages` 恰好包含一个名为 `ceph` 的条目。
- 所有 **daemon** 重启后，健康状态仍为 `HEALTH_WARN` 或 `HEALTH_ERR`：从 toolbox 运行 `ceph health detail`，并在继续操作前解决报告的问题。

请勿删除 `CephCluster`、Rook Operator 或 PV 对象来恢复升级。

相关主题

- 升级 Operator — 在当前 channel 中发布并批准新的 Operator 版本
- 上架软件包 — 上传 Operator 软件包
- Operator — `Subscription`、`InstallPlan` 和审批策略
- 支持的 OS 和 Kernel 版本

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于识别目的。

核心概念

核心概念

Rook Operator

Ceph CSI

Ceph 模块功能

核心概念

目录

Rook Operator

Ceph CSI

Ceph 模块功能

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
操作步骤
删除存储池
操作步骤
查看对象存储池地址
操作步骤

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。
	• 当集群类型为标准版时：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

操作步骤

重启存储组件

操作步骤

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
注意事项
操作步骤
添加设备
操作步骤
硬盘状态

添加设备类

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

注意事项

当集群类型为 **Extend** 时，不支持添加设备类。

这些操作步骤用于添加主机设备（`spec.storage.nodes`）。OSD 使用 Local Storage Block PersistentVolumes 时不适用。对于此类集群，请按照使用 Local Storage 在内部模式下部署中所述添加设备类。

操作步骤

1. 进入 **Administrator**。
2. 在左侧导航栏中，点击 **Storage Management** > **Distributed Storage**。
3. 点击 **Device Classes** 页签。
4. 点击 **Add Device Class**，并按照以下说明配置相关参数。

参数	描述
名称	设备类的名称。以下名称不能用于设备类：`hdd`、`ssd`、`nvme`。

存储设备	要将存储设备添加到设备类，可以选择 `Selection Device` 和 `Input Device` 两种方法：
	• 选择设备：
	从可用存储设备中进行选择。如果设备满足以下条件，则视为可用：
	  • 设备类型为磁盘或 mpath
	  • 未检测到文件系统（fsType 为空）
	  • 容量超过 10 GiB
	rbd、nbd 和 dm-* 等设备不会显示在可供选择的设备列表中。
	注意：需要预先部署 Alauda Container Platform Local Disk Provisioner。
	• 输入设备：
	手动输入节点下空白设备的名称，例如 `sda`。
	注意：为获得最佳性能和管理效果，强烈建议使用裸磁盘作为存储设备，而不是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，以便使用和管理。

注意：将硬盘添加为存储设备后，不支持通过界面更新或移除硬盘。

操作步骤

1. 进入 **Administrator**。
2. 在左侧导航栏中，点击 **Storage Management** > **Distributed Storage**。
3. 点击 **Device Classes** 页签。
4. 在设备类右侧，点击 **Add Device**，并按照以下说明配置相关参数。

参数	描述

指定磁盘	要将存储设备添加到设备类，可以选择 `Selection Device` 和 `Input Device` 两种方法：
	• 选择设备：
	从可用存储设备中进行选择。如果设备满足以下条件，则视为可用：
	  • 设备类型为磁盘或 mpath
	  • 未检测到文件系统（fsType 为空）
	  • 容量超过 10 GiB
	rbd、nbd 和 dm-* 等设备不会显示在可供选择的设备列表中。
	注意：需要预先部署 Alauda Container Platform Local Disk Provisioner。
	• 输入设备：
	手动输入节点下空白设备的名称，例如 `sda`。
	注意：为获得最佳性能和管理效果，强烈建议使用裸磁盘作为存储设备，而不是使用磁盘上的单个分区。

5. 点击 **Add**。

硬盘状态

- 正常：存储设备对应的状态为 IN+UP。
- 异常：存储设备对应的状态为 IN+DOWN。
- 离线：存储设备对应的状态为 OUT+UP。
- 故障：存储设备对应的状态为 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
- 存储概览
- 性能监控
- 组件监控

告警
- 配置通知
- 处理告警
- 故障复盘

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

替换或移除设备

前提条件

约束和限制

操作步骤

参考

替换或移除由 PVC 支持的 OSD 设备

替换或移除由 PVC 支持的 OSD 设备

先决条件

约束和限制

识别 OSD 和 PVC

移除 OSD（缩容）

替换失败的 OSD（保持容量）

清除残留的 BlueStore 标签

参考

替换或移除存储节点

替换或移除存储节点

先决条件

约束和限制

操作步骤

参考资料

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
- 故障转移

块存储灾难恢复

- 术语
- 备份配置
- 计划迁移
- 灾难恢复

对象存储灾难恢复

- 术语
- 前提条件
- 架构
- 操作步骤
- 故障转移
- 故障恢复

更新优化参数

更新优化参数

- 操作步骤

创建 **Ceph Object Store** 用户

创建 **Ceph Object Store** 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 Terminal 验证存储池配额

为 **Ceph RGW** 启用 **D3N** 缓存

为 **Ceph RGW** 启用 **D3N** 缓存

背景

前提条件

操作概述

为缓存准备本地文件系统

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

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
性能影响

设置 Ceph OSD 满容量阈值

设置 Ceph OSD 满容量阈值

前提条件
操作步骤
建议

持久卷加密

持久卷加密

概述
密钥管理系统（KMS）的访问配置
创建用于持久卷加密的 StorageClass
使用租户 ConfigMap 覆盖 Vault 连接详细信息
启用和禁用密钥轮换（可选）

配置 OSD WAL 和 DB 分区

配置 OSD WAL 和 DB 分区

简介
场景
先决条件
约束和限制
规划 WAL 和 DB 容量
操作步骤
验证

配置 Multus 多网卡网络

配置 Multus 多网卡网络

概述
限制和前提条件
在创建集群时配置多网卡网络
验证
故障排除建议

MinIO 到 Rook Ceph RGW 数据迁移指南

MinIO 到 Rook Ceph RGW 数据迁移指南

1. 概述和架构
2. 前提条件和准备工作
3. 部署配置（Manifest）
4. 执行流程
5. 故障排查与调优建议

使用 Ceph RGW Exporter 监控 Ceph RGW

使用 Ceph RGW Exporter 监控 Ceph RGW

术语
前提条件
工作原理
操作步骤
自定义 exporter
指标参考
故障排除
卸载

替换或移除设备

本文档介绍如何从基于主机的 Rook-Ceph 集群（`CephCluster.spec.storage.nodes`）中移除存储设备（磁盘）。根据剩余 OSD 是否有足够容量来容纳待移除磁盘中的数据，您可能需要先添加替换磁盘。

如果 OSD 由 Local Storage Block PV（`CephCluster.spec.storage.storageClassDeviceSets`）提供支持，请改用替换或移除由 PVC 支持的 OSD 设备。不要编辑 `spec.storage.nodes`。不要擦除仍绑定到 PV 的设备；PV 变为 `Available` 后，按照该页面中的说明清除遗留的 BlueStore 标签，然后再重复使用。

WARNING

这些步骤适用于在以内部模式部署中从主机设备创建的 OSD。不要将其用于从 Local Storage Block PersistentVolumes 创建的 OSD。请参阅使用 Local Storage 以内部模式部署。

目录

前提条件
约束和限制
操作步骤
检查集群状态和容量
添加替换磁盘（如需要）
缩容 Rook operator
将 OSD 标记为 Out 并等待数据迁移

移除 OSD

清理磁盘

扩容 Rook operator

验证集群运行状况

参考

前提条件

- 所有集群组件均正常运行。
- 存储集群不是通过“添加所有空磁盘”选项创建的。运行以下命令进行验证；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 及更高版本。

约束和限制

- 在数据重新平衡期间，集群性能可能会暂时下降。除非绝对必要，否则请避免同时对多个磁盘执行操作。
- 如果集群由于待移除磁盘以外的原因处于 `HEALTH_ERR` 状态，请勿继续操作。在该状态下继续操作可能会进一步削弱数据弹性。
- 如果待移除磁盘是某个特定设备类中的最后一个磁盘，该设备类将不再存在。依赖该设备类的任何存储池或策略都将受到影响。继续操作前，请确保没有存储池专门绑定到该设备类。

操作步骤

1 检查集群状态和容量

1. 验证集群整体运行状况。

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

2. 确定待移除磁盘的 OSD ID 和使用情况。

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

记下目标 OSD 的 **USE** 值。然后确认所有剩余 OSD 的 **AVAIL** 总和（不包括目标 OSD）大于目标 OSD 的 **USE** 值。这可确保剩余 OSD 有足够的可用空间来容纳移除磁盘后的数据。
如果剩余容量不足，请继续下一步以先添加替换磁盘。否则，请跳转到缩容 Rook operator。

2 添加替换磁盘（如需要）

如果剩余 OSD 没有足够的可用容量，请在移除旧磁盘前添加替换磁盘。在此步骤期间，Rook operator 必须处于运行状态。

1. 进入容器平台。
2. 在左侧导航栏中，点击存储管理 > 分布式存储 > 设备类。
3. 点击添加设备，选择安装替换磁盘的节点，选择新磁盘，并将其分配到与待移除磁盘相同的设备类。
4. 等待创建新的 OSD 并完成数据重新平衡。监控进度：

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

等待输出显示 `HEALTH_OK`，且没有 misplaced 或 recovering PG。

3 缩容 **Rook operator**

缩容 Rook operator，防止其干扰移除过程（例如在操作步骤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 将 OSD 标记为 Out 并等待数据迁移

1. 如果 rook-ceph-tools pod 尚未运行，请将其启用。

   ```
   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`。这会指示 Ceph 将 OSD 上的所有数据迁移到剩余的 OSD。

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

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

   ```
   ceph -s
   ```

   在数据迁移完全完成之前，请勿继续操作。 在迁移完成前移除 OSD 将导致数据丢失。

5 移除 OSD

1. 编辑 CephCluster 资源以移除磁盘条目。

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

   在 `spec.storage.nodes` 下找到该磁盘并删除其条目。保存并退出。

2. 删除 OSD 部署。

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

3. 进入 tools pod，并从集群中永久移除 OSD。将 `<id>` 替换为 OSD ID。

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

在 tools pod 中：

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

6 清理磁盘

如果磁盘将继续物理连接到节点，请擦除其元数据，以防 Rook 意外将其识别并使用。在磁盘所在的节点上运行以下命令。将 `/dev/vdb` 替换为实际的设备路径。

```
# Remove any device-mapper entries left by Ceph
dmsetup remove /dev/mapper/ceph--<vg-name>   # Replace with the actua
l mapper name if present

# Wipe the partition table
sgdisk --zap-all /dev/vdb

# Zero the first 100 MB to clear residual Ceph metadata
dd if=/dev/zero of=/dev/vdb bs=1M count=100 oflag=direct,dsync
```

7 扩容 **Rook operator**

集群恢复正常后，恢复 Rook operator。

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

8 验证集群运行状况

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 t
ree
```

2. 验证集群已恢复到正常状态。

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

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

参考

- Rook Ceph OSD 管理↗

替换或移除由 PVC 支持的 OSD 设备

本文档介绍在 Ceph 集群使用由 **PVC** 支持的 **OSD**（`CephCluster.spec.storage.storageClassDeviceSets`）时，如何移除或替换 Object Storage Daemon（OSD）。这些 OSD 使用来自 Local Storage StorageClass 的 Block PersistentVolumeClaims（PVC），例如 `localblock`。

对于基于主机的 OSD（`CephCluster.spec.storage.nodes`），请改用替换或移除设备。

减少 `storageClassDeviceSets[].count` 或移除 `storageClassDeviceSets` 条目不会删除 OSD。它只会降低 OSD 的期望数量，使 Rook 在清除 OSD 后不会重新创建设备。从 Ceph 中移除数据始终需要显式清除特定的 OSD ID。存在多个 PVC 集时，可以移除其中一个集，也可以移除最后一个 PVC 集。

目录

- 先决条件
- 约束和限制
- 识别 OSD 和 PVC
- 移除 OSD（缩容）
  - 降低期望数量
  - 将 OSD 标记为 out 并等待数据迁移
  - 清除 OSD
  - 验证
- 替换失败的 OSD（保持容量）
  - 保持期望数量

清除失败的 OSD

等待 Rook 创建替换的

清除残留的 BlueStore 标签

参考

先决条件

- 除要替换设备时的故障磁盘外，所有集群组件均运行正常。
- 集群是使用由 PVC 支持的 OSD 创建的。确认 `CephCluster` 包含 `spec.storage.storageClassDeviceSets`，并且 `spec.storage.nodes` 为空。

```
kubectl -n rook-ceph get cephcluster ceph-cluster -o jsonpath='{.spec.storage.storageClassDeviceSets}' ; echo
kubectl -n rook-ceph get cephcluster ceph-cluster -o jsonpath='{.spec.storage.nodes}' ; echo
```

- 你具有 `cluster-admin` 访问权限，并且可以进入 `rook-ceph-tools` 执行命令。

约束和限制

- 在数据重新平衡期间，集群性能可能会暂时降低。除非已确认剩余容量和故障域，否则一次不要清除多个 OSD。
- 如果集群因正在移除的 **OSD** 以外的原因处于 `HEALTH_ERR` 状态，请不要继续操作。
- 如果 OSD 是某个设备类中的最后一个设备，则仅使用该设备类的存储池会受到影响。
- 在降低 `count`、移除集或清除 OSD 时，不要将 Rook operator 缩容至零。Rook 必须保持运行，以便在更改期望容量后不会重新创建 OSD。只有在擦除已释放的磁盘时，才可以将其缩容至零，具体说明请参阅清除遗留的 BlueStore 标签。
- 清除后，如果 `count` 仍高于剩余 OSD 的数量，或者设备集仍在 spec 中，Rook 会创建新的 PVC/OSD 来填补缺口。这就是替换路径。对于缩容操作，请在清除之前或清除后立即降低 `count`，或者移除该集。

识别 OSD 和 PVC

1. 列出 OSD Pod，并记下 OSD ID 和节点。

```
kubectl -n rook-ceph get pod -l app=rook-ceph-osd -o wide
```

2. 查找支持该 OSD 的 PVC。

```
kubectl -n rook-ceph get pvc -l ceph.rook.io/DeviceSet=<deviceSet-name>
kubectl -n rook-ceph get deploy rook-ceph-osd-<id> -o jsonpath='{.metad
ata.labels.ceph\.rook\.io/pvc}{"\n"}'
```

3. 确认集群运行状况和剩余容量。

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

剩余 OSD 的 **AVAIL** 总和必须大于要移除 OSD 的 **USE**。

移除 OSD（缩容）

当你希望减少 OSD 数量且不希望 Rook 配置替代 OSD 时，请使用此操作步骤。

1 降低期望数量

修补 `CephCluster` 中匹配的 `storageClassDeviceSets` 条目，使 `count` 等于你要保留的 OSD 数量。例如，将包含 `count: 3` 的集更改为 `2`。

```
kubectl -n rook-ceph patch cephcluster ceph-cluster --type=json \
  -p '[{"op":"replace","path":"/spec/storage/storageClassDeviceSets/
0/count","value":2}]'
```

如果有多个设备集，请更改数组索引，以修补拥有该 OSD 的设备集。要停用整个设备集，请删除该列表条目，而不是降低 `count`：

```
kubectl -n rook-ceph patch cephcluster ceph-cluster --type=json \
  -p '[{"op":"remove","path":"/spec/storage/storageClassDeviceSets/
1"}]'
```

当 `count` 减少或移除设备集时，Rook 不会删除 OSD。

2 将 OSD 标记为 out 并等待数据迁移

1. 如果 OSD 仍在运行，请将其缩容。

   ```
   kubectl -n rook-ceph scale deploy rook-ceph-osd-<id> --replicas=0
   ```

2. 从 tools Pod 中，将 OSD 标记为 `down` 和 `out`。

   ```
   kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
   ceph osd down osd.<id>
   ceph osd out osd.<id>
   ceph -s
   ```

   等待 PG 变为 `active+clean`，并且集群变为 `HEALTH_OK`（或仅存在与此次移除相关的警告）。回填完成前不要清除 **OSD**，除非 OSD 已经发生故障且你必须继续操作。

3 清除 OSD

使用上游的 `rook-ceph-purge-osd` Job。它会在需要时将 OSD 标记为 `out`，运行 `ceph osd purge`，并删除 OSD Deployment、OSD prepare Job 和 OSD PVC。ServiceAccount 已随 Rook 安装。

该 Job 仅适用于已经处于 **down** 状态且存在于 `ceph osd dump` 中的 OSD。它不会等待回填。请先完成将 OSD 标记为 out 并等待数据迁移。如果 OSD 从未加入 Ceph（`not found osd.<id> in OSDDump`），请跳过该 Job，手动删除 Deployment、prepare Job 和 PVC。

替换 `<OSD-ID>`，并使用集群中已运行的 Rook operator 镜像：

```
OSD_ID=<id>
IMG=$(kubectl -n rook-ceph get deploy rook-ceph-operator -o jsonpath
='{.spec.template.spec.containers[0].image}')
kubectl -n rook-ceph delete job rook-ceph-purge-osd --ignore-not-foun
d
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: rook-ceph-purge-osd
  namespace: rook-ceph
  labels:
    app: rook-ceph-purge-osd
spec:
  backoffLimit: 0
  template:
    metadata:
      labels:
        app: rook-ceph-purge-osd
    spec:
      serviceAccountName: rook-ceph-purge-osd
      restartPolicy: Never
      containers:
        - name: osd-removal
          image: ${IMG}
          args:
            - "ceph"
            - "osd"
            - "remove"
            - "--preserve-pvc"
            - "false"
            - "--force-osd-removal"
            - "false"
            - "--osd-ids"
            - "${OSD_ID}"
          env:
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: ROOK_MON_ENDPOINTS
              valueFrom:
                configMapKeyRef:
```

```
                key: data
                name: rook-ceph-mon-endpoints
            - name: ROOK_CEPH_USERNAME
              valueFrom:
                secretKeyRef:
                  key: ceph-username
                  name: rook-ceph-mon
            - name: ROOK_CONFIG_DIR
              value: /var/lib/rook
            - name: ROOK_CEPH_CONFIG_OVERRIDE
              value: /etc/rook/config/override.conf
            - name: ROOK_FSID
              valueFrom:
                secretKeyRef:
                  key: fsid
                  name: rook-ceph-mon
            - name: ROOK_LOG_LEVEL
              value: DEBUG
          volumeMounts:
            - mountPath: /etc/ceph
              name: ceph-conf-emptydir
            - mountPath: /var/lib/rook
              name: rook-config
            - name: ceph-admin-secret
              mountPath: /var/lib/rook-ceph-mon
      volumes:
        - name: ceph-admin-secret
          secret:
            secretName: rook-ceph-mon
            items:
              - key: ceph-secret
                path: secret.keyring
        - emptyDir: {}
          name: ceph-conf-emptydir
        - emptyDir: {}
          name: rook-config
EOF
kubectl -n rook-ceph wait --for=condition=complete job/rook-ceph-purg
e-osd --timeout=180s
kubectl -n rook-ceph logs -l app=rook-ceph-purge-osd --tail=50
```

确认日志包含 `completed removal of OSD`。然后删除 Job：

```
kubectl -n rook-ceph delete job rook-ceph-purge-osd
```

等待 PVC 消失。如果要重复使用该磁盘，请等待其 backing PV 变为 `Available`。

如果 Job 不可用，请从 tools Pod 中清除 OSD，并自行删除残留对象。首先删除已完成的 OSD prepare Job。该 Job 仍挂载 PVC，因此在 Job 删除前，PVC 会保持 `Terminating` 状态。

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd purg
e osd.<id> --yes-i-really-mean-it
kubectl -n rook-ceph delete deploy rook-ceph-osd-<id>
kubectl -n rook-ceph delete job rook-ceph-osd-prepare-<osd-pvc-name>
kubectl -n rook-ceph delete pvc <osd-pvc-name>
```

4 验证

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd tree
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
kubectl -n rook-ceph get pvc -l ceph.rook.io/DeviceSet=<deviceSet-nam
e>
kubectl -n rook-ceph get cephcluster ceph-cluster -o jsonpath='{.spe
c.storage.storageClassDeviceSets}' ; echo
```

被移除的 OSD ID 不得出现在 `ceph osd tree` 中。如果你降低了 `count`，它必须与该设备集的剩余 OSD 数量匹配。如果你移除了设备集，该名称不得出现在 `storageClassDeviceSets` 中。否则 Rook 会创建另一个 OSD。

替换失败的 OSD（保持容量）

当磁盘或 PVC 发生故障且你希望保持 OSD 数量不变时，使用此操作步骤。

1 保持期望数量

不要降低 `storageClassDeviceSets[].count`，也不要移除设备集。

如果你要安装一块新磁盘，请在同一个 StorageClass（`localblock` 或设备集使用的类）中创建一个新的 Block PV。该 PV 必须在合适的节点上处于 `Available` 状态。

如果你要重复使用同一个本地磁盘，请不要创建新的 PV。清除 OSD 后，Local Storage 通常会将现有 PV 恢复为 `Available` 状态。你必须先清除残留的 BlueStore 标签，然后 Rook 才能再次准备该磁盘。如果跳过清除步骤，Rook 会重复使用旧的 OSD 元数据，替换的 OSD 会在 `expand-bluefs` 期间保持 `CrashLoopBackOff` 状态。

2 清除失败的 OSD

按照将 OSD 标记为 out 并等待数据迁移和清除 OSD 中的步骤操作。如果 OSD Pod 已处于 `CrashLoopBackOff` 或 `Error` 状态，则可以跳过 `ceph osd down`。

3 等待 Rook 创建替换的

由于 `count` 未发生变化，Rook 会使用未使用的最高索引创建新的 PVC，并启动新的 OSD。新的 OSD ID 可能不同于旧 OSD。

```
kubectl -n rook-ceph get pvc -l ceph.rook.io/DeviceSet=<deviceSet-nam
e>
kubectl -n rook-ceph get pod -l app=rook-ceph-osd
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph osd tree
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph -s
```

等待新的 OSD 处于 `up` / `in` 状态，并且 PG 处于 `active+clean` 状态。

清除残留的 BlueStore 标签

仅当 OSD PVC 已消失且 backing PV 处于 `Available` 状态时，才使用此操作步骤。不要清除仍绑定到 PVC 的设备。

Rook OSD prepare 不会清除以前使用过的 Block PV。`sgdisk --zap-all`、清除开头或结尾的 100 MiB，以及 `blkdiscard` 都不够。BlueStore 会在 **1 GiB**（`0x40000000`）和 **10 GiB**（`0x280000000`）处保留标签。

1. 查找 PV 节点和本地路径。

```
kubectl get pv <pv-name> -o jsonpath='node={.spec.nodeAffinity.required.nodeSelectorTerms[0].matchExpressions[0].values} path={.spec.local.path}{"\n"}'
```

2. 可选：将 Rook operator 缩放到零，以免在清除期间绑定新的 PVC。

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

3. 在该节点上以 root 身份清除 PV 路径解析到的块设备。将 `/dev/vdc` 替换为实际设备（PV 本地路径上的 `readlink -f`，或 LocalVolume 的 `devicePaths` 条目）。

```
DEV=/dev/vdc
dd if=/dev/zero of=$DEV bs=1M seek=1024 count=16 oflag=direct conv=fsync
dd if=/dev/zero of=$DEV bs=1M seek=10240 count=16 oflag=direct conv=fsync
dd if=/dev/zero of=$DEV bs=1M count=16 oflag=direct conv=fsync
```

   确认标签已清除。`ceph-bluestore-tool show-label --dev $DEV` 必须报告未找到有效的 bdev 标签，并且 `ceph-volume raw list $DEV` 必须为 `{}`。

4. 如果你缩减了 Rook operator，则将其恢复。

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

然后，如果你是在替换 OSD，请返回等待 Rook 创建替换项；如果你是在扩容，请提高 `count`。

参考

- Rook Ceph OSD 管理↗
- 替换或移除设备（基于主机的 OSD）

替换或移除存储节点

本文档介绍如何从基于主机的 Rook-Ceph 集群（`CephCluster.spec.storage.nodes`）中移除存储节点。根据剩余 OSD 是否具有足够容量来接收待移除节点中的数据，您可能需要先添加替换节点。

如果 OSD 由 Local Storage Block PV（`CephCluster.spec.storage.storageClassDeviceSets`）提供支持，请先按照替换或移除由 PVC 支持的 OSD 设备中的说明移除或替换该节点上的每个 OSD，然后再排空并移除节点。请勿删除由 PVC 支持的 OSD 的 `spec.storage.nodes` 条目。

> **WARNING**
>
> 这些步骤适用于在以内部模式部署中从主机设备创建的 OSD。请勿将这些步骤用于从 Local Storage Block PersistentVolumes 创建的 OSD。请参阅使用 Local Storage 以内部模式部署。
>
> 此操作步骤用于永久移除存储节点及其 OSD 数据。不适用于 Alauda OS 节点替换。对于其 provider 管理的持久磁盘正通过池槽重新挂载的 Alauda OS worker，请勿将 OSD 标记为 `out`，删除 CephCluster 节点条目，清除 OSD，或将持久磁盘视为替换磁盘。请改为遵循 Alauda OS 部署先决条件和 provider 的节点替换操作步骤。

目录

- 先决条件
- 约束和限制
- 操作步骤
  - 检查集群状态和容量

添加替换节点（如需要）

调整组件部署配置

将所有 OSD 标记为 Out 并等待数据迁移

移除旧节点的 OSD

验证集群运行状况

参考资料

先决条件

- 所有集群组件（如适用，不包括故障节点）均运行正常。
- 开始之前，请记录旧节点上的磁盘数量，以及每个磁盘所属的设备类。

约束和限制

- 在三节点 Ceph 集群中，丢失一个节点后冗余性将立即降低。请尽快完成操作步骤，以尽量缩短风险窗口。
- 在数据重新平衡期间，集群 I/O 性能可能会暂时降低。
- 如果集群因待移除节点以外的原因处于 `HEALTH_ERR` 状态，请勿继续操作。在该状态下继续操作可能会进一步削弱数据弹性。

操作步骤

1 检查集群状态和容量

1. 确定要移除节点上运行的所有 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. 验证集群整体运行状况。

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

3. 检查所有 OSD 的容量。

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

汇总要移除节点上所有 OSD 的 **USE** 值。然后确认所有剩余 OSD（位于其他节点上）的 **AVAIL** 总和大于该总值。这样可以确保移除节点后，剩余 OSD 有足够的可用空间来接收这些数据。
如果剩余容量不足，请继续执行下一步，先添加替换节点。否则，请跳转到调整组件部署配置。

2 添加替换节点（如需要）

如果剩余 OSD 没有足够的可用容量，请在移除旧节点之前添加替换节点。

1. 进入 **Container Platform**。
2. 使用平台的节点管理功能，将替换计算机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. 单击 **Add Device**，选择新节点并选择适当的磁盘。如果旧节点在不同设备类中包含多个磁盘，请针对每个磁盘/设备类组合重复此步骤，直到所有磁盘均已添加。
5. 等待新 OSD 变为活动状态并完成数据重新平衡。监控进度：

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

请等待集群恢复到 `HEALTH_OK` 状态，且没有处于 misplaced 或 recovering 状态的 PG，然后再继续操作。

3 调整组件部署配置

Rook 管理的 Ceph 守护进程（MON、MGR、MDS）可能已调度到旧节点上。请将旧节点排除在组件调度范围之外，以便 operator 将这些组件重新调度到其他节点上。

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

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

4 将所有 OSD 标记为 Out 并等待数据迁移

1. 如果 rook-ceph-tools pod 尚未运行，请将其启用。

```
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`。这会指示 Ceph 将这些 OSD 上的所有数据迁移到剩余 OSD。

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

对节点上的每个 OSD ID 重复执行此操作。

4. 监控重新平衡进度，直到集群恢复到 `HEALTH_OK` 状态，且没有处于 misplaced 或 recovering 状态的 PG。

```
ceph -s
```

在数据迁移完全完成之前，请勿继续操作。在迁移完成前移除 OSD 将导致数据丢失。

5 移除旧节点的 OSD

1. 编辑 CephCluster 资源，以移除旧节点条目。

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

在 `spec.storage.nodes` 下找到旧节点，并删除整个节点条目。保存并退出。

2. 删除旧节点上每个 OSD 的 OSD 部署。

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

对节点上的每个 OSD ID 重复执行此操作。

3. 进入 tools pod，并从集群中永久移除每个 OSD。

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

在 tools pod 中，对每个 OSD 执行以下操作：

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

6 验证集群运行状况

1. 确认所有已移除的 OSD 不再显示在集群中。

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

2. 验证集群已恢复到正常运行状态。

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

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

参考资料

- Rook Ceph OSD 管理↗

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，平台中的其他业务集群通过集成访问和使用该集群提供的存储服务。
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置。这种分离式部署方式是平台分布式存储推荐的最佳实践。

目录

- 架构
- 基础设施要求
  - 平台要求
  - 集群要求
  - 资源要求
  - 存储设备要求
    - 存储设备类型要求
    - 容量规划
    - 容量监控和扩容
  - 网络要求
    - 网络隔离
    - 网络接口速度要求
- 操作步骤
  - 部署 Operator
  - 创建 ceph 集群
  - 创建存储池

创建文件存储池

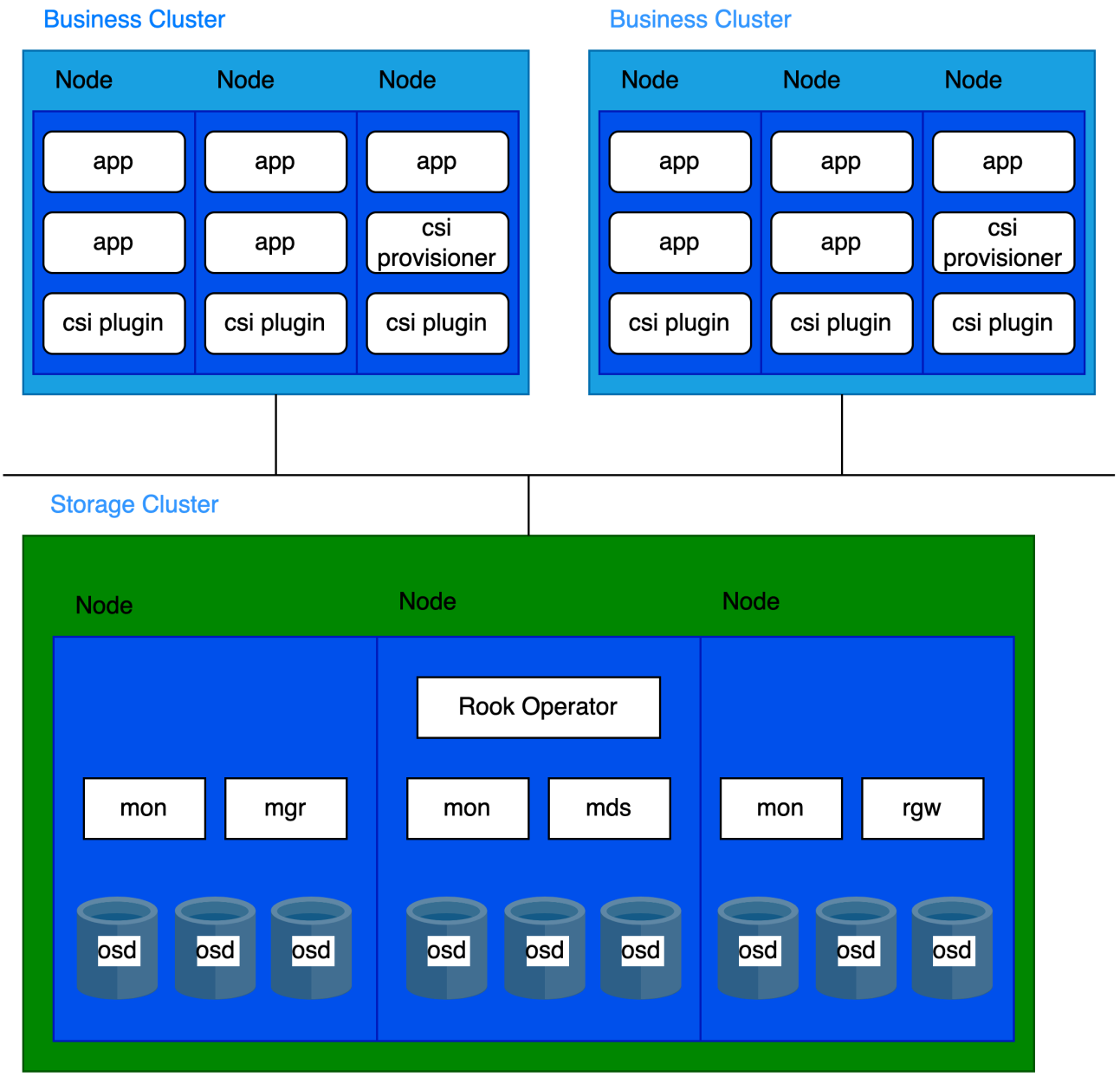
创建块存储池

创建对象存储池

后续操作

架构

存储计算分离架构



基础设施要求

平台要求

3.18 及更高版本支持。

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所需的组件，请参阅核心概念。

每个组件都有不同的 CPU 和内存要求。建议配置如下：

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

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，适用以下每节点资源建议：

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

存储设备要求

建议每个节点部署不超过 12 个存储设备。这有助于缩短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，且每个设备的容量不超过 10TiB，并确保所有磁盘的大小和类型均相同。

容量规划

部署前，必须根据具体业务要求规划存储容量。默认情况下，分布式存储系统采用 3 副本冗余策略。因此，可用容量的计算方式是将总原始存储容量（所有存储设备的容量总和）除以 3。

以 30(N) 个节点为例（副本数 = 3），可用容量情况如下：

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

容量监控和扩容

1. 主动进行容量规划
   始终确保可用存储容量大于使用量。如果存储空间完全耗尽，则需要手动干预才能恢复，无法仅通过删除或迁移数据解决。
2. 容量告警
   集群会在两个阈值处触发告警：

- 利用率达到 **80%**（“接近满载”）：主动释放空间或扩展集群。
- 利用率达到 **95%**（“已满”）：存储空间已完全耗尽，标准命令无法释放空间。请立即联系平台支持。

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. 扩展建议

- 避免：向现有节点添加存储设备。
- 建议：通过添加新的存储节点进行横向扩展。
- 要求：新节点必须使用与现有节点大小、类型和数量均相同的存储设备。

网络要求

分布式存储必须使用 **HostNetwork**。

网络隔离

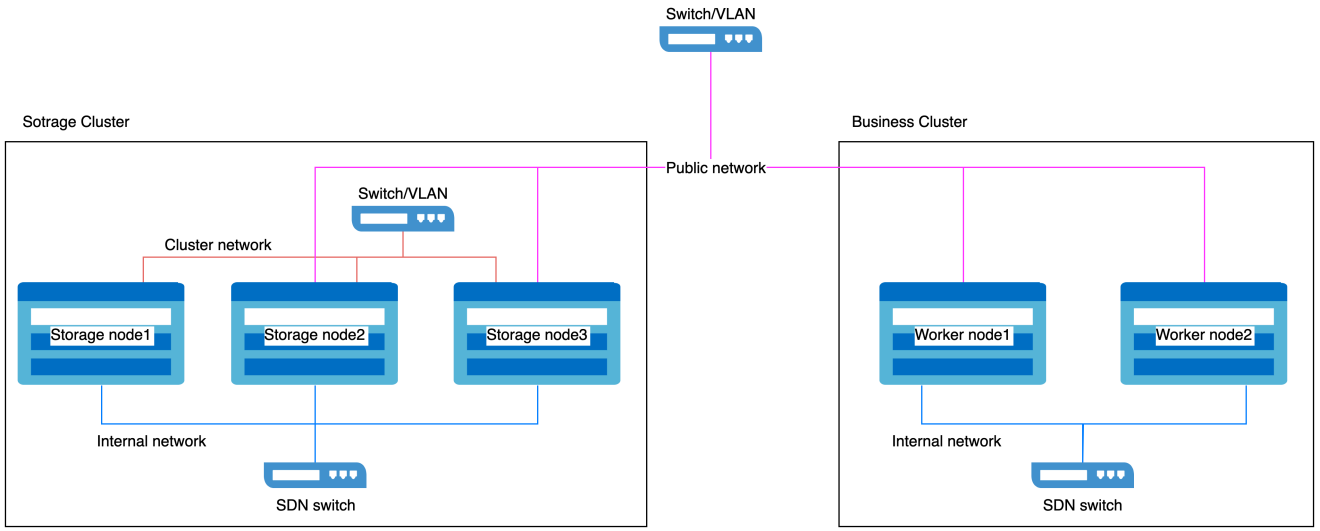
网络分为两种类型：

- 公共网络：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再平衡（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：
   在每个主机上预留两个网络接口：
   - 公共网络：用于客户端和组件通信。
   - 集群网络：用于内部复制和再平衡流量。
2. 对于业务集群：
   在每个主机上预留一个网络接口，用于访问存储公共网络。

网络隔离配置示例



网络接口速度要求

1. 存储节点

   - 公共网络和集群网络要求使用 10GbE 或更高速率的网络接口。

2. 业务集群节点

   - 用于访问存储公共网络的网络接口必须为 10GbE 或更高速率。

操作步骤

1 部署 **Operator**

1. 访问管理员。
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，请参照界面上的提示清理已部署信息并重试，然后重新部署 Operator；如果希望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

在存储集群的控制平面节点上执行命令。

▶ 单击查看

对于直接创建的内部 CephCluster，如果省略字段，平台默认将 `spec.disruptionManagement.managePodBudgets` 设置为 `true`。 仅在选择退出由 Rook 管理的 PDB 保护时，才将其显式设置为 `false`。 平台不会设置 `osdMaintenanceTimeout`；将采用已安装的 Rook 版本的默认值。

参数：

- 公共网络 **CIDR**：存储公共网络的 CIDR（例如 `- 10.0.1.0/24`）。
- 集群网络 **CIDR**：存储集群网络的 CIDR（例如 `- 10.0.2.0/24`）。
- 存储设备：指定分布式存储要使用的存储设备。
  格式示例：

```
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 `sdb` 等可能在重启后发生变化的不稳定设备路径。

3 创建存储池

提供三种存储池类型。请根据业务需求选择并创建适合的存储池。

创建文件存储池

在存储集群的控制平面节点上执行命令。

▶ 单击查看

创建块存储池

在存储集群的控制平面节点上执行命令。

▶ 单击查看

创建对象存储池

在存储集群的控制平面节点上执行命令。

▶ 单击查看

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。
以外部模式部署

清理分布式存储

如果需要删除 rook-ceph 集群并重新部署一个新集群，请按照本文档依次清理分布式存储服务相关资源。

目录

注意事项
操作步骤
删除 VolumeSnapshotClasses
删除 StorageClasses
删除存储池
删除 ceph-cluster
删除 rook-operator
执行清理脚本
清理脚本
注意事项
操作步骤

注意事项

清理 rook-ceph 前，请确保已删除所有使用 Ceph 存储的 PVC 和 PV 资源。

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
```

当这些命令没有输出时，表示清理完成。

2 删除 StorageClasses

1. 转到 管理员。
2. 在左侧导航栏中，点击 存储管理 > 存储类。
3. 点击⋮> 删除，删除所有使用 Ceph 存储解决方案的 StorageClasses。

3 删除存储池

此步骤应在完成上一步后执行。

1. 转到 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。
3. 在存储池区域中，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**，还需要在集群的一个控制平面节点上执行以下命令，以删除已创建的内置存储池。

```
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

此步骤应在完成上一步后执行。

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. 删除执行清理的作业。

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

4. 验证 ceph-cluster 是否已完成清理。

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

当此命令没有输出时，表示清理完成。

5 删除 rook-operator

此步骤应在完成上一步后执行。

1. 删除 rook-operator。

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

2. 验证 rook-operator 是否已完成清理。

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

当此命令没有输出时，表示清理完成。

3. 验证所有 ConfigMaps 是否已清理。

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

当此命令没有输出时，表示清理完成。如果有输出结果，请执行以下命令进行清理，并将 `<configmap>` 替换为实际输出。

```
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
```

当此命令没有输出时，表示清理完成。如果有输出结果，请执行以下命令进行清理，并将 `<secret>` 替换为实际输出。

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

5. 验证 rook-ceph 是否已完成清理。

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

当此命令没有输出时，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，需要清理主机上 rook-ceph 的所有残留内容。

清理脚本

清理脚本 clean-rook.sh 的内容如下：

▶ 点击查看

注意事项

清理脚本依赖 sgdisk 命令，因此请确保在执行清理脚本前已安装该命令。

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

操作步骤

1. 在部署了分布式存储的业务集群中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，系统会提示确认是否确实要清理设备。确认后，输入 yes 开始清理。
2. 使用 `lsblk -f` 命令检查分区信息。当该命令输出中的 `FSTYPE` 列为空时，表示清理完成。

灾难恢复

文件存储灾难恢复

术语

备份配置

故障转移

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

故障转移

故障恢复

文件存储灾难恢复

CephFS Mirror 是 Ceph 文件系统的一项功能，用于实现不同 Ceph 集群之间的异步数据复制，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，确保主集群发生故障时，备份集群能够快速接管服务。

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）。主集群与备份集群之间的差异数据通常相当于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。请结合使用平台的 备份与恢复 功能来备份 PVC 和 PV 资源。

目录

术语

备份配置

前提条件

操作步骤

在次集群中为文件存储池启用 Mirror

获取 Peer Token

在主集群中创建 Peer Secret

在主集群中为文件存储池启用 Mirror

在主集群中部署 Mirror Daemon

故障转移

先决条件

术语

术语	说明
主集群	当前提供存储服务的集群。
次集群	用于备份的集群。

备份配置

前提条件

- 准备两个适合部署 Alauda Storage 的集群，即主集群和次集群，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 及更高版本）必须一致。
- 在主集群和次集群中创建分布式存储服务
- 在主集群和次集群中创建名称相同的文件存储池。

操作步骤

1 在次集群中为文件存储池启用 **Mirror**

在次集群的控制平面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>`：文件存储池的名称。

2 获取 Peer Token

此令牌是建立两个集群之间镜像连接的关键凭证。

在次集群的控制平面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>`：文件存储池的名称。

3 在主集群中创建 Peer Secret

从次集群获取 Peer Token 后，需要在主集群中创建 Peer Secret。

在主集群的控制平面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>`：第 2 步中获取的令牌。
- `<fs-name>`：文件存储池的名称。

4 在主集群中为文件存储池启用 Mirror

在主集群的控制平面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>`：文件存储池的名称。
- `<schedule-interval>`：快照执行周期。有关详细信息，请参阅官方文档↗。
- `<retention-policy>`：快照保留策略。有关详细信息，请参阅官方文档↗。

5 在主集群中部署 Mirror Daemon

Mirror Daemon 持续监控文件存储池中的数据变化（已启用 Mirror）。它会定期创建快照，并通过网络将快照差异推送到次集群。

在主集群的控制平面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
```

故障转移

如果主集群发生故障，您可以直接继续使用次集群中的 CephFS。

先决条件

主集群的 Kubernetes 资源已备份并恢复到次集群，包括应用的 PVC、PV 和工作负载。

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。
Rook® 是 The Linux Foundation 的注册商标。
Alauda 是独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 没有关联，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

块存储灾难恢复

RBD Mirror 是 Ceph 块存储 (RBD) 的一项功能，可在不同的 Ceph 集群之间进行异步数据复制，从而实现跨集群灾难恢复 (DR)。其核心功能是以主备模式同步数据，确保主集群发生故障时，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）。主集群与备份集群之间的差异数据通常对应一个快照周期内的写入数据。
- RBD Mirror 仅提供底层存储数据备份，不处理 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
备份配置
前提条件
网络前提条件
操作步骤
Bootstrap Peers(`Primary <-> Secondary`)
设置卷复制环境
为 PVC 启用镜像
计划迁移

迁移
先决条件
操作步骤
灾难恢复
故障切换（突然关机）
先决条件
操作步骤
故障恢复（灾后恢复）
先决条件
操作步骤

术语

术语	说明
主集群	当前提供存储服务的集群。
辅助集群	用于备份的备用集群。

备份配置

前提条件

- 准备两个能够部署 Alauda Storage 的集群：一个主集群和一个辅助集群。
- 两个集群必须运行相同的平台版本（v3.12 或更高版本）。
- 在主集群和辅助集群中创建分布式存储服务。
- 在主集群和辅助集群中创建名称相同的块存储池。
- 请确保以下两个镜像已上传到平台的私有镜像仓库：

- `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

主集群与辅助集群之间通过 Ceph 的 Public 网络进行同步。因此，站点间的 Public 网络必须满足以下要求：

- 每个集群都必须有专用的 Public 网络网段。主集群和辅助集群的 Public 网络必须完全可路由且互相可达。
- Public 网络的持续带宽利用率不应超过 60%，以确保为复制突发流量和故障转移场景留有充足余量。
- 两个站点之间的往返时间 (RTT) 应小于 30 ms。
- 两个 Public 网络之间的丢包率应低于 0.05%，以确保复制性能稳定且可预测。

操作步骤

Bootstrap Peers(`Primary <-> Secondary`)

1. 为主集群的块存储池启用镜像
   在主集群和辅助集群的控制平面节点上执行以下命令：

Command

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

Output

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

参数：

- `<block-pool-name>`：块存储池名称。

2. 获取对等令牌

   此令牌是建立集群间镜像连接的关键凭证。

   在主集群和辅助集群的控制平面节点上执行以下命令：

   Command

```
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
```

   Output

```
# Output truncated due to sensitive information
eyJmc2lkIjoiMjc2N2I3ZmEtY2YwYi00N...
```

3. 在对等集群中创建对等令牌 **Secret**

   在主集群和辅助集群的控制平面节点上执行以下命令：

   Command

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

   Output

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

参数：

- `<token>`：步骤 2中获取的令牌。
  在主集群中，使用从辅助集群获取的令牌配置此字段。
  在辅助集群中，使用从主集群获取的令牌配置此字段。
- `<block-pool-name>`：块存储池名称。

4. 为块存储池修补对等 **Secret**
   在主集群和辅助集群的控制平面节点上执行以下命令：

Command

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

Output

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

参数：

- `<block-pool-name>`：块存储池名称。

5. 部署镜像守护进程

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

   在主集群和辅助集群的控制平面节点上执行以下命令：

   Command

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

   Output

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

6. 验证镜像状态

   在主集群和辅助集群的控制平面节点上执行以下命令：

   Command

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

   Output

```
# All "OK" statuses indicate normal operation
{"daemon_health":"OK","health":"OK","image_health":"OK","state
s":{}}
```

参数：

- `<block-pool-name>`：块存储池名称。

设置卷复制环境

此功能可在不中断原生应用操作的情况下高效复制和同步数据，从而提高系统的可靠性和可用性。

1. 设置 **CsiAddons Controller**

   在主集群和辅助集群的控制平面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：

   - `<registry>`：平台的 Registry 地址。

2. 启用 **CsiAddons sidecar**

   在主集群和辅助集群的控制平面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**

   在主集群和辅助集群的控制平面节点上执行以下命令：

   Command

```
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
```

Output

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

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

为 **PVC** 启用镜像

在主集群的控制平面节点上执行以下命令：

Command

```
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
```

Output

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

1 `<vr-name>`：VolumeReplication 对象的名称，建议与 PVC 名称相同。

2 `<namespace>`：VolumeReplication 所属的命名空间，必须与 PVC 的命名空间相同。

3 `<pvc-name>`：需要启用镜像的 PVC 名称。

注意 启用后，辅助集群中的 RBD 镜像将变为只读。

计划迁移

用例：数据中心维护、技术更新、灾难规避等。

迁移

迁移操作是将生产环境切换到备用设施（通常为恢复站点），或反向切换的过程。

进行迁移时，应停止访问主站点上的镜像。此时应将辅助集群上的镜像设为主镜像，以便在那里恢复访问。

先决条件

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

操作步骤

按照以下步骤，将工作负载从主集群计划迁移到辅助集群：

1. 缩减主集群上所有使用已启用镜像的 PVC 的应用 Pod。
2. 更新主集群上所有已启用镜像的 PVC 对应的 VolumeReplication。
   将 `spec.replicationState` 设置为 `secondary`。
3. 为辅助集群上所有已启用镜像的 PVC 创建 VolumeReplication。

Example

```
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 对象的名称，建议与 PVC 名称相同。
2. `<namespace>`：VolumeReplication 所属的命名空间，必须与 PVC 的命名空间相同。
3. `<pvc-name>`：需要启用镜像的 PVC 名称。

4. 检查 VolumeReplication CR 状态，确认辅助站点上的镜像是否已标记为 `primary`。
5. 镜像标记为 `primary` 后，PVC 即可使用。此时可以扩容应用，以使用该 PVC。

灾难恢复

用例：自然灾害、电力故障、系统故障和崩溃等。

故障切换（突然关机）

发生灾难需要恢复时，在辅助站点创建 VolumeReplication CR。

由于与主站点的连接已断开，operator 会自动向驱动程序发送 GRPC 请求，强制将辅助站点上的 dataSource 标记为 `primary`。

先决条件

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

操作步骤

1. 为辅助集群上所有已启用镜像的 PVC 创建 VolumeReplication。

Example

```
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 对象的名称，建议与 PVC 名称相同。
2. `<namespace>`：VolumeReplication 所属的命名空间，必须与 PVC 的命名空间相同。
3. `<pvc-name>`：需要启用镜像的 PVC 名称。

2. 检查 VolumeReplication CR 状态，确认辅助站点上的镜像是否已标记为 `primary`。
3. 镜像标记为 `primary` 后，PVC 即可使用。此时可以扩容应用，以使用该 PVC。

故障恢复（灾后恢复）

主站点上的故障集群恢复后，如果要从辅助站点执行故障恢复，请按照以下步骤操作：

先决条件

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

操作步骤

1. 缩减主站点上正在运行的应用（如果有）。确保工作负载使用的所有持久卷在主集群上均已不再使用。
2. 在主站点上，将 VolumeReplication CR 的 replicationState 从 primary 更新为 secondary。
3. 缩减辅助站点上的应用。

4. 在辅助站点上，将 VolumeReplication CR 的 replicationState 状态从 `primary` 更新为 `secondary`。
5. 在主站点上，确认 VolumeReplication 状态已标记为卷可供使用。
6. 卷标记为可供使用后，在主站点上将 replicationState 状态从 secondary 更改为 primary。
7. 再次扩容主站点上的应用。

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

对象存储灾难恢复

Ceph RGW Multi-Site 功能是一种跨集群异步数据复制机制，旨在同步地理位置分散的 Ceph 集群之间的对象存储数据，从而提供高可用性（HA）和灾难恢复（DR）功能。

目录

术语
前提条件
架构
操作步骤
在主集群中创建对象存储
为主 Zone 配置外部访问
获取 `access-key` 和 `secret-key`
创建次 Zone 并配置 Realm 同步
为次 Zone 配置外部访问
检查 Ceph 对象存储同步状态
故障转移
操作步骤
故障恢复
操作步骤

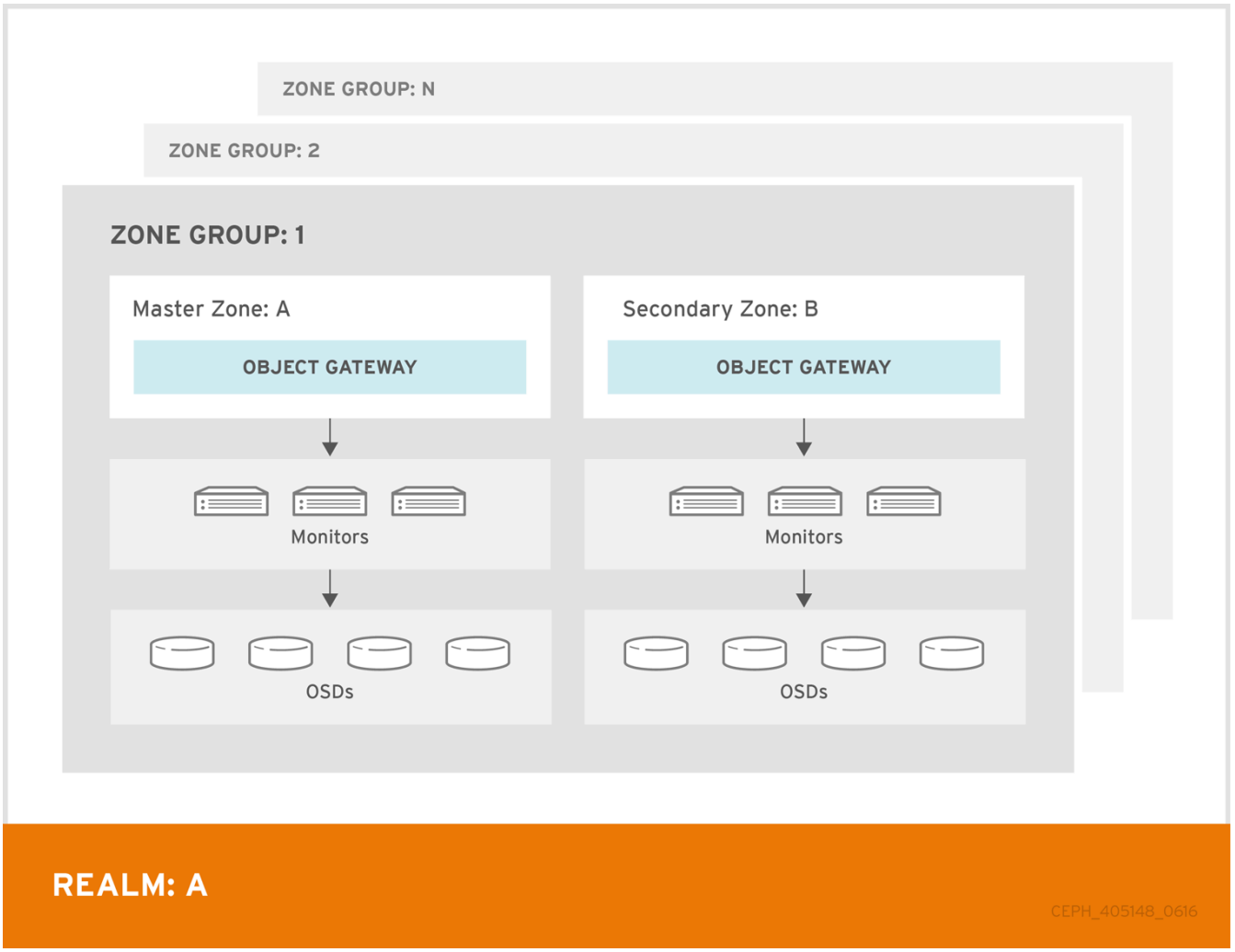
术语

术语	说明
主集群	当前提供存储服务的集群。
次集群	用于备份的备用集群。
Realm、ZoneGroup、Zone	• Realm：Ceph 对象存储中的最高级别逻辑分组，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 支持跨 Zone 的数据同步和复制，通常位于同一地理区域内。 • Zone：ZoneGroup 内实际存储数据的逻辑分组。每个 Zone 独立管理和存储对象，并且可以拥有自己的数据池和元数据池配置。

前提条件

- 准备两个可用于部署 Rook-Ceph 的集群（主集群和次集群），并确保两者之间具有网络连接。
- 两个集群必须使用相同的平台版本（v3.12 或更高版本）。
- 确保主集群和次集群上均未部署 Ceph 对象存储。
- 参考创建存储服务文档部署 Operator 并创建集群。创建集群后，请勿通过向导继续创建对象存储池。请改用 CLI 工具按照下述说明进行配置。

架构



操作步骤

本指南提供同一 ZoneGroup 中两个 Zone 之间的同步解决方案。

1 在主集群中创建对象存储

此步骤将创建 Realm、ZoneGroup、主 Zone 以及主 Zone 的网关资源。

在主集群的控制平面节点上执行以下命令：

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>`：主 Zone 名称。
- `<primary-object-store-name>`：网关名称。

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 为主 Zone 配置外部访问

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 创建次 Zone 并配置 Realm 同步

本节介绍如何创建次 Zone，并通过从主集群拉取 Realm 信息来配置同步。

在次集群的控制平面节点上执行以下命令：

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>`：从主集群获取的外部地址。
- `<access-key>`：从此处获取 AK。
- `<secret-key>`：从此处获取 SK。
- `<secondary-zone-name>`：次 Zone 名称。
- `<secondary-object-store-name>`：次网关名称。

2. 创建次 **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 为次 **Zone** 配置外部访问

1. 获取次网关的 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. 向次 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

在次集群的 `rook-ceph-tools` pod 中执行以下命令

```
# enter 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。

故障转移

当主集群发生故障时，需要将次区域提升为主区域。切换后，次区域的网关可以继续提供对象存储服务。

操作步骤

在次集群的 `rook-ceph-tools` pod 中执行以下命令

```
# enter 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

# Make the recovered zone the master and default zone
radosgw-admin zone modify --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name> --rgw-zone=<secondary-zone-name> --master

# Update the period so that the changes take effect
radosgw-admin period update --commit --rgw-realm=<realm-name> --rgw-zonegroup=<zone-group-name>
```

参数

- `<realm-name>`：Realm 名称。
- `<zone-group-name>`：次区域组名称。
- `<secondary-zone-name>`：次区域名称。

故障恢复

主站点上的故障集群恢复后，如果要从次站点切回，请按照以下步骤操作

操作步骤

在主集群的 `rook-ceph-tools` pod 中执行以下命令

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

# check sync status, wait for sync from secondary site
radosgw-admin sync status

#           realm 962d6b80-b218-4fc8-8198-e498fabc4e9d (relam-primary)
#       zonegroup 9de3acd7-0b01-4a04-ac84-1421c6103a16 (zonegroup-primar
y)
#            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-seco
ndary)
#                         syncing
#                         full sync: 0/128 shards
#                         incremental sync: 128/128 shards
#                         data is caught up with source

# Make the recovered zone the master and default zone
radosgw-admin zone modify --rgw-realm=<realm-name> --rgw-zonegroup=<zone-
group-name> --rgw-zone=<primary-zone-name> --master

# Update the period so that the changes take effect
radosgw-admin period update --commit --rgw-realm=<realm-name> --rgw-zoneg
roup=<zone-group-name>
```

参数

- `<realm-name>`：Realm 名称。
- `<zone-group-name>`：区域组名称。

- `<primary-zone-name>`：主区域名称。

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

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 Object Store 用户

我们支持通过自定义资源定义（CRD）创建和自定义 Object Store 用户。

目录

前提条件
操作步骤
创建用户
允许在其他命名空间中创建用户
获取用户信息

前提条件

- 已创建对象存储池

操作步骤

1 创建用户

在集群的控制平面节点上执行命令。

```
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

参数	描述
name	要创建的 Object Store 用户的名称。
namespace	创建 Object Store 用户所在的命名空间。
displayName	显示名称。
clusterNamespace	父级 `CephCluster` 和 `CephObjectStore` 所在的命名空间。如果未指定，则用户必须与集群和 Object Store 位于同一命名空间。要启用此功能，CephObjectStore 的 `allowUsersInNamespaces` 必须包含此用户所在的命名空间。
ObjectStore	要创建用户的 Object Store。此名称与对象存储池的名称一致。
quotas	可选 表示可以为用户设置的配额限制。 • maxBuckets：用户的最大存储桶数限制。默认为 `100`。

参数	描述
	• maxSize：用户所有存储桶中全部对象的大小上限。设置为 `-1` 表示不限制。 • maxObjects：用户所有存储桶中的对象数量上限。设置为 `-1` 表示不限制。
capabilities	可选 Ceph 允许为用户授予其他权限。此设置目前只能在创建 Object Store 用户时使用。如果需要修改用户的权限，则必须删除并重新创建该用户。有关更多信息，请参阅 Ceph 文档↗。我们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中创建用户

如果在 Rook 集群命名空间以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 Object Store 凭据的应用程序非常有用。

在集群的控制平面节点上执行命令。

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

3 获取用户信息

在集群的控制平面节点上执行命令。

```
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

池配额是在 Ceph 存储池级别应用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

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 Terminal 验证存储池配额

前提条件

- 已安装 Ceph 集群
- 已创建 Ceph 池

为文件存储池设置池配额

在集群的控制平面节点上执行命令。

命令

```
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\"}]"
```

示例

```
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

参数	描述
size	以带数量后缀的字符串表示的字节配额（例如 "10Gi"）
fs-pool-name	文件存储池的名称。

为块存储池设置池配额

在集群的控制平面节点上执行命令。

命令

```
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\"}]"
```

示例

```
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

参数	描述
size	以带数量后缀的字符串表示的字节配额（例如 "10Gi"）
block-pool-name	块存储池的名称。

为对象存储池设置池配额

在集群的控制平面节点上执行命令。

命令

```
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\"}]"
```

示例

```
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

参数	描述
size	以带数量后缀的字符串表示的字节配额（例如 "10Gi"）
object-pool-name	对象存储池的名称。

通过 **Ceph Terminal** 验证存储池配额

```
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）缓存通过将 **RGW** 节点上的本地 **NVMe/SSD** 磁盘用作缓存层，加速对热点对象的访问。 它可减少对后端 OSD 的频繁读取操作，并显著提升热点数据的读取性能。

目录

背景
前提条件
操作概述
为缓存准备本地文件系统
在 CephObjectStore 中启用 D3N 缓存
参数
验证 D3N 配置
通过 Ceph 工具检查 RGW 配置
验证 RGW 日志
验证缓存行为

背景

D3N 的核心理念简单而有效： RGW 不再反复从 Ceph 后端存储获取对象数据，而是直接从位于快速磁盘（NVMe/SSD）上、连接到 RGW 节点的 本地持久化缓存 中提供热点数据。

这种设计尤其适用于：

- 大型对象下载
- 对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 Alauda Storage
- 已创建对象存储（RGW / CephObjectStore）
- RGW 节点上有可用的高性能本地磁盘（NVMe / SSD）

操作概述

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，并将其挂载到某个目录
3. 配置 `CephObjectStore` 以：
   - 通过 `hostPath` 挂载目录
   - 启用与 D3N 相关的 RGW 参数

为缓存准备本地文件系统

在每个运行 RGW 服务的节点上挂载一个目录。建议使用高性能专用磁盘（而非分区），并将其格式化为 **XFS** 文件系统，同时配置 `/etc/fstab`，以确保重启后仍会保留挂载。请在后续文档中将 `</path/to/cache/dir>` 替换为你配置的实际目录路径。

在 **CephObjectStore** 中启用 **D3N** 缓存

编辑 `CephObjectStore` 资源：

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

在 `gateway` 下添加以下配置：

```
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

参数	描述
`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`	缓存的最大大小（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，应会出现 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，缓存文件会出现在主机缓存目录中：

```
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
...
```

这些文件的存在表明 RGW 正在从本地 D3N 缓存提供数据。

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 Linux Foundation 无关，亦未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

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

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

步骤 1. 确认节点内核版本

步骤 2. 在 CephCluster 中启用加密

步骤 3. 等待配置生效

禁用传输加密

在现有集群中禁用加密

验证

检查 CephCluster 设置

检查客户端兼容性

检查工作负载挂载

故障排查建议

性能影响

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

操作步骤

检查当前集群状态

通过 ceph CLI 设置阈值

通过更新 `CephCluster` CR 设置阈值

验证应用的值

恢复或重新基线阈值

建议

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

密钥管理系统（KMS）的访问配置

使用 `vaulttokens` 配置访问

使用 `vaulttenantsa` 配置访问

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

创建用于持久卷加密的 StorageClass

先决条件

操作步骤

验证步骤

使用租户 ConfigMap 覆盖 Vault 连接详细信息

启用和禁用密钥轮换（可选）

前提条件

设置卷复制环境

（可选）强制设置密钥轮换的优先级

启用密钥轮换

禁用密钥轮换

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 置备的 PV 的静态数据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 访问凭证和连接详细信息
- 创建启用加密的 StorageClass
- 启用和禁用密钥轮换策略

> **WARNING**
>
> 本章中的持久卷加密适用于 RBD 支持的 PV。不涵盖对象存储桶。

密钥管理系统（KMS）的访问配置

首先配置 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。

每种提供商模式的身份验证和元数据模型不同：

模式	`encryptionKMSType`	身份验证材料
Vault 令牌模式	`vaulttokens`	存储在 Secret 中的静态 Vault 令牌
Vault 租户 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 + 密钥 UID

> ⚠️ 重要：**Alpha** 功能免责声明
> 使用 `vaulttenantsa` 访问 KMS 目前处于 **Alpha** 阶段，仅严格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，您确认并接受以下技术限制：

> - 不适用于生产环境： 此功能尚未经过全面的系统级验证。在生产环境中部署此功能存在较高的流程失败或数据损坏风险。
> - 不保证 **SLA**： 此功能不属于标准服务级别协议（SLA）的范围。我们不保证技术支持响应时间，也不提供紧急修复程序。
> - 重大变更和弃用： API 端点、配置架构和核心处理逻辑可能会在未来版本中发生不向后兼容的变化。此功能也可能在不事先通知的情况下被完全弃用。
> - 不提供平滑升级路径： 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。由此造成的任何状态或数据丢失均由用户自行承担。

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。
- 您拥有可访问目标后端路径的有效 Vault 令牌。

操作步骤

1. 创建用于存储 Vault 令牌的 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	描述	必需
`encryptionKMSType`	KMS 提供商模式。对于静态 Vault 令牌身份验证，请使用 `vaulttokens`。	是
`vaultAddress`	Vault 服务端点，包括协议和端口。	是
`vaultBackend`	Vault Secret 引擎，默认为 kv-v2	否
`vaultBackendPath`	用于数据加密密钥（DEK）的 Vault Secret 引擎路径。默认为 secret/	否
`vaultTLSServerName`	用于 SNI 和证书主机名验证的 TLS 服务器名称。	否

参数	描述	必需
`vaultCAVerify`	连接到 Vault 服务器时是否需要 CA 身份验证，默认为 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. 验证令牌 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）上：
  - 确保已存在策略，并且 Vault 中的密钥值后端路径已启用。
  - 确保 Vault 服务器使用签名证书。

操作步骤

在 ACP 分布式存储能够向 `Vault` 进行身份验证并开始使用它之前，您需要配置 Kubernetes 身份验证方法。以下说明将创建并配置 `serviceAccount`、`ClusterRole` 和 `ClusterRoleBinding`，以允许 ACP 分布式存储向 `Vault` 进行身份验证。

1. 在工作负载命名空间中创建租户 ServiceAccount。

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

2. 在存储命名空间中应用用于 Vault 令牌审查的 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. 为令牌审查器 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. 读取审查器 JWT、集群 CA 和 API 端点。

```
SA_JWT_TOKEN=$(kubectl -n rook-ceph get secret rbd-csi-vault-token-review-token \
  -o jsonpath="{.data['token']}" | base64 --decode; echo)
SA_CA_CRT=$(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="{.clusters[0].cluster.server}")
```

5. 在 Vault 中配置 Kubernetes 身份验证。

```
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. 为租户命名空间创建 Vault 角色。

```
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 中查找的默认角色名称。
- ACP 分布式存储集群中租户命名空间的默认服务账户名称为 `ceph-csi-vault-sa`。
- 可以通过在租户命名空间中创建 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	描述	必需
`encryptionKMSType`	设置为 vaulttenantsa，以使用服务账户向 Vault 进行身份验证。	是
`vaultAddress`	Vault 服务器的主机名或 IP 地址及端口号。	是

参数	描述	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 服务器名称。	否
`vaultAuthPath`	在 Vault 中启用 kubernetes 身份验证方法的路径。默认路径为 `kubernetes`。如果身份验证方法启用在不同于 `kubernetes` 的路径中，则需要将此变量设置为 `/v1/auth/<path>/login`。	否
`vaultAuthNamespace`	启用 kubernetes 身份验证方法的 Vault 命名空间。	否
`vaultNamespace`	存在用于存储密钥的后端路径的 Vault 命名空间。	否
`vaultCAFromSecret`	包含 CSI 使用的 Vault CA 证书的 Secret。	否
`vaultClientCertFromSecret`	包含用于与 Vault 建立 mTLS 连接的客户端证书的 Secret。	否
`vaultClientCertKeyFromSecret`	包含与 `vaultClientCertFromSecret` 匹配的私钥的 Secret。	否
`vaultRole`	绑定到租户 ServiceAccount 和策略的 Vault Kubernetes 身份验证角色。	否
`tenantSAName`	工作负载使用的租户 ServiceAccount 名称，通常为	否

参数	描述	必需
	`ceph-csi-vault-sa`。	

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. 验证令牌审查权限：

```
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>` 中创建测试加密 PVC，并验证其是否成功绑定。

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 端点（`<kmip_address>:5696`）、客户端证书链和密钥唯一标识符（UID）。
- 您具有在 `rook-ceph` 中创建 Secret 和 ConfigMap 的权限。

操作步骤

1. 在 Thales CipherTrust Manager 中创建 KMIP 客户端配置文件并注册客户端。
2. 从配置文件中下载客户端证书、私钥和 CA 证书。
3. 为 KMIP 客户端凭证创建 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 条目中的密钥 UID 与 Thales CipherTrust Manager 中生成的密钥匹配。

> **NOTE**
>
> 对于 `kmip`，关键区别在于，加密密钥通过 KMIP 密钥 UID 和双向 TLS 信任链解析，而不是通过 Vault 令牌解析。

创建用于持久卷加密的 StorageClass

配置 KMS 访问权限后，创建包含加密参数的 StorageClass。

先决条件

- 你具有创建 StorageClass 的权限。
- 你知道目标 KMS 配置 ID（`<kms_config_id>`）。

操作步骤

1. 转到管理员。
2. 在左侧导航栏中，点击存储 > **StorageClasses**。
3. 点击创建 **Storage Class**。
   注意：以下内容是表单格式示例，你也可以选择 YAML 完成操作。
4. 选择 **CephRBD** 块存储，然后点击下一步。
5. 切换到 YAML 视图
6. 在参数中启用加密：
   - `encrypted: "true"`
   - `encryptionKMSID: "<kms_config_id>"`
7. 点击创建。

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 创建测试 PVC，并确认其达到 `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
```

使用租户 **ConfigMap** 覆盖 **Vault** 连接详细信息

通过在 ACP 命名空间中创建 ConfigMap，可以按租户重新配置 Vault 连接详细信息，并使用与 `rook-ceph` 命名空间中 `csi-kms-connection-details` ConfigMap 中设置的值不同的配置选项。ConfigMap 必须位于租户命名空间中。租户命名空间中的 ConfigMap 值将覆盖 `csi-kms-connection-details` ConfigMap 中为该命名空间创建的加密持久卷所设置的值。

操作步骤

1. 在 ACP 控制台中，导航到**Alauda Container Platform** -> 配置 -> **ConfigMaps**。
2. 切换到目标租户项目/命名空间。
3. 点击创建 **ConfigMap**，并输入 `ceph-csi-kms-config` 作为名称。
4. 添加特定于租户的 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. 点击创建。

验证步骤

1. 验证租户覆盖 ConfigMap：

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

2. 在同一命名空间中创建新的加密 PVC，并确认其达到 `Bound`。
3. 如果密钥访问失败，请检查命名空间事件和 CSI provisioner 日志，确认 Vault 端点和身份验证路径是否匹配。

启用和禁用密钥轮换（可选）

密钥轮换支持定期刷新加密 PV 使用的加密密钥。

前提条件

- 请确保以下两个镜像已上传到平台的私有镜像仓库：
  - `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`

设置卷复制环境

1. 设置 **CsiAddons Controller**

   在控制平面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 Registry 地址。

2. 启用 **CsiAddons sidecar**

   在控制平面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）强制设置密钥轮换的优先级

在密钥轮换中，优先级是指系统检查计划注解的顺序。默认优先级设置为 StorageClass（推荐）。这意味着系统只从 StorageClass 读取注解。

但是，如果希望系统先检查持久卷声明（PVC），然后回退到 StorageClass，则可以在 CSI-addons ConfigMap 中将 schedule-precedence 设置为 PVC，以配置此行为。

可以按如下所示定义 csi-addons 的 ConfigMap：

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

重启 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
```

启用密钥轮换

要启用密钥轮换，请按优先级从高到低的顺序，将注解 `keyrotation.csiaddons.openshift.io/schedule: <value>` 添加到 PersistentVolumeClaims、Namespace 或 StorageClass。

`<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"
```

禁用密钥轮换

可以针对以下对象禁用密钥轮换：

- StorageClass 的所有持久卷声明（PVC）
- 特定 PVC

为 **StorageClass** 的所有 **PVC** 禁用密钥轮换

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

为特定持久卷声明禁用密钥轮换

1. 确定要禁用密钥轮换的 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>` 是要禁用密钥轮换的 PVC 名称。

2. 对上一步中的 `EncryptionKeyRotationCronJob` CR 应用以下更改，以禁用密钥轮换：

- 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。PVC 的密钥轮换将被禁用。

配置 OSD WAL 和 DB 分区

目录

简介
场景
先决条件
约束和限制
规划 WAL 和 DB 容量
操作步骤
检查主机设备路径
为一个 OSD 配置一个元数据分区
为节点上的 OSD 配置共享元数据设备
等待 OSD 准备作业完成
验证

简介

本主题介绍如何为基于主机的 Rook-Ceph 对象存储守护进程 (OSD) 配置快速元数据分区。在此配置中，OSD 将用户数据存储在数据设备上，并将 RocksDB 数据库和预写日志 (WAL) 等 BlueStore 元数据存储在更快的设备或分区上。

当 HDD 支持的 OSD 需要降低元数据操作的延迟，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

- 将 SSD 或 NVMe 设备用作节点上所有选定 OSD 数据设备的共享元数据设备。
- 为一个特定的 OSD 数据设备使用一个元数据分区。
- 规划新建或重新创建的基于主机的 OSD 的元数据分区容量。

先决条件

开始之前，请确保满足以下条件：

- 你拥有对集群的 `cluster-admin` 访问权限。
- 你可以通过 shell 访问需要检查或准备本地设备的每个存储节点。
- 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原生应用数据。
- `rook-ceph-tools` 部署在 `rook-ceph` 命名空间中可用，或者你有权临时启动该部署。

约束和限制

> **WARNING**
>
> 更改 `metadataDevice`、`databaseSizeMB` 或 `walSizeMB` 不会就地移动现有 OSD 的 WAL 或 DB。若要更改现有 OSD 的 WAL/DB 布局，请在确认集群有足够容量且放置组运行正常后，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/...`。`/dev/sdb` 等 Linux 名称可能会在重启或更换硬件后发生变化。
- 当 `metadataDevice` 配置为 `spec.storage.config` 或 `spec.storage.nodes[].config` 时，Alauda Storage 会在同一节点上的多个 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，请勿将分区路径用作 `metadataDevice`。

- 当 `metadataDevice` 配置在特定的 `devices[].config` 下时，Alauda Storage 会使用 `lvm prepare` 初始化 OSD。在此模式下，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` 设备存储 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，当你为一个 HDD OSD 分配一个专用元数据分区时，请将该分区的容量设置为该 OSD 的 DB 和 WAL 容量。对于此模式，通常无需在 `CephCluster` CR 中设置 `databaseSizeMB` 或 `walSizeMB`。

请使用以下指南确定每个元数据分区的大小：

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% 至 2%
RGW 对象存储或大量使用 omap	至少为 HDD 数据设备容量的 4%。当快速设备容量允许时，请使用更大的分区。
混合或不确定的工作负载	以 HDD 数据设备容量的 4% 作为初始基线。当工作负载可能创建大量 omap 键或小对象时，请增大分区大小。

示例：

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

请将表中的 RGW 数值视为最低规划基线，而不是上限。如果快速设备具有足够的容量，请分配超过 4% 的容量，以降低 BlueStore 元数据回溢到 HDD 的可能性。

当节点级别的多个 HDD OSD 共享一个完整的快速设备时，请按以下公式计算所需的快速设备总容量：

```
metadata capacity per OSD x number of HDD OSDs on the node
```

如果共享元数据设备小于计算出的容量，请减少共享该快速设备的 HDD OSD 数量，或根据工作负载使用更小的每个 OSD 目标值。请避免依赖容量非常小的 DB 设备，因为 DB 设备已满时，BlueStore 元数据可能会回溢到 HDD。

操作步骤

1 检查主机设备路径

登录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：

用途	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

当元数据目标为分区时，使用设备级别的 `metadataDevice`。此模式将每个 OSD 数据设备映射到其自身的 WAL/DB 分区。分区容量即为该 OSD 的元数据容量，因此在此配置中不需要 `databaseSizeMB` 和 `walSizeMB`。

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

更新 `spec.storage`，配置类似于以下示例。

```
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 共置于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

仅当元数据目标为整个设备或逻辑卷时，才使用节点级别的 `metadataDevice`。Rook 会在该节点上选定的 OSD 数据设备之间共享元数据目标。仅当需要限制 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，请勿设置 `walSizeMB`；WAL 可以与 DB 共置。

```
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"
```

当整个快速设备专用于节点上的 OSD 元数据时，请使用此模式。

4 等待 **OSD** 准备作业完成

查看 OSD 准备作业和 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 OSD 准备作业失败，请检查作业日志。

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

验证

如果 tools Pod 未运行，请启动它。

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

检查集群运行状况。

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

确认 OSD 已启动，并已分配到预期的主机。

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

若要检查本地 BlueStore 元数据布局，请从 OSD Pod 中或准备 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 OSD 具有指向预期元数据设备或分区的 `block.db`，以及（当由 Ceph 单独配置时）`block.wal` 条目。

Ceph 是 Red Hat, Inc. 或其子公司在美国和其他国家/地区的商标或注册商标。 Rook® 是 The Linux Foundation 的注册商标。 Alauda 是一家独立供应商。本产品与 LF Projects, LLC 或 The Linux Foundation 无关联，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

配置 Multus 多网卡网络

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** 流量分配到不同的网络接口（NIC）。

目录

概述

两个管理界面

限制和前提条件

在创建集群时配置多网卡网络

创建 Multus 网络

打开创建集群向导

切换到 YAML 模式

编辑网络部分

在 YAML 模式下提交

验证

（新控制台）检查 StorageCluster 网络设置

检查生效的 CephCluster 网络设置

检查 Ceph Pod 是否已连接到网络

检查工作负载挂载

故障排除建议

概述

默认情况下，ACP Ceph 存储集群使用主机网络（`provider: host`）。从 Web 控制台创建存储集群时，也可以选择容器网络，将 Ceph 流量置于单个容器网络上（分配给 `rook-ceph` 命名空间的单个 `NetworkAttachmentDefinition`）。有关基本的单网络选项，请参阅创建标准存储服务。

Multus 多网卡网络扩展了容器网络模型，使你可以将两个 Ceph 网络角色映射到不同的网络：

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

将这两个角色分配到不同的网卡可以增强隔离性，并避免大量复制/恢复流量与客户端 I/O 争用资源。

> **NOTE**
>
> 所有流入和流出 **OSD** 的流量都使用 **Multus** 网络——这是主要的存储数据路径：RBD 客户端 I/O、CephFS 数据和元数据（MDS）、对象数据（RGW 在 OSD 上读取/写入对象），以及 OSD 复制流量。通过 **Kubernetes Service** 访问的端点——**monitors**（客户端使用 mon Service 获取集群映射）、**manager** 和 **RGW S3** 端点——通过默认网络提供服务；即使这些守护进程的 Pod 连接了 Multus 接口，它们面向 Service 的监听器仍使用默认网络。这是 Rook 的行为，并非配置错误。因此，Multus 隔离的是存储数据路径，而不是由 Service 提供服务的控制和入口端点。

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`（由存储 operator 管理）	`StorageCluster.spec.ceph.network` — operator 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`

要确定哪个资源管理特定集群，请检查是否存在 `StorageCluster`：

```
kubectl get storagecluster -A
```

- 如果集群存在 `StorageCluster`，则由新版控制台管理 — 请在 **StorageCluster** 上配置网络（operator 会还原对底层 `CephCluster` 的手动修改）。
- 如果不存在 `StorageCluster`，则集群由旧版控制台作为 **CephCluster** 直接管理 — 请在 `CephCluster` 上配置网络。

这两种资源都公开相同的网络字段（`provider`、`selectors`、`addressRanges`）。以下各节提供了每种管理界面对应的清单。

网络角色与网络之间的映射通过 `selectors` 声明：

选择器键	**Ceph** 角色	后端网络
`public`	Ceph public network	作为 `<namespace>/<nad-name>` 引用的 `NetworkAttachmentDefinition`（NAD）
`cluster`	Ceph cluster（复制）网络	作为 `<namespace>/<nad-name>` 引用的 `NetworkAttachmentDefinition`（NAD）

你可以仅指定 `public`，也可以同时指定 `public` 和 `cluster`。当 `provider` 为 `multus` 时，至少需要一个选择器。

限制和前提条件

配置 Multus 多网卡网络之前，请注意以下限制：

- 仅限创建时配置

  - 必须在创建存储集群时配置 Multus 多网卡网络。不支持更改运行中集群的网络提供程序：应用新的网络模型需要 Ceph 将所有监视器故障转移，而且 API 不允许直接从一个非空提供程序切换到另一个非空提供程序（必须先将提供程序恢复为空值）。请将网络模型视为集群的固定属性。

- 已安装 **Multus CNI** 插件
  - 集群必须使用 Kube-OVN 作为网络插件，并且必须安装 Multus CNI 插件。这是一次性集群范围设置。若要安装，请导航至管理员 > **Marketplace** > 集群插件，搜索 `multus`，然后安装 **Alauda Container Platform Multi-Network**。详情请参阅配置 Kube-OVN 网络以支持 Pod 多网络接口。

- 单一 **IP** 地址族
  - Multus 网络必须使用单一 IP 地址族。IPv4 和 IPv6 均受支持，但 Multus 不支持双栈（同时使用 IPv4 和 IPv6）。若要使用 IPv6，请在网络块中设置 `ipFamily: IPv6`。
  - 这与创建标准存储服务中的基本容器网络选项不同；该选项不支持 IPv6。此限制仅适用于该单网络表单选项；此处介绍的 Multus 多网卡操作步骤支持 IPv6。

> **WARNING**
>
> Multus 会将存储数据路径（OSD 流量）移至专用网卡，但不会改变工作负载使用存储的方式：应用 **Pod** 无需加入 **Multus** 网络。客户端通过默认网络（经由 mon Service）访问监视器，而 Ceph CSI 驱动程序在每个节点上以主机网络模式运行，负责处理到 OSD 的数据路径。因此，真正的要求是存储节点能够路由到底层网络，而不是工作负载 Pod 已连接到该网络。请在部署前规划网络角色、NAD 和节点路由。

在创建集群时配置多网卡网络

向导的表单字段无法表达多网卡设置，但创建集群向导提供了 **YAML** 模式。建议的操作流程是：照常完成向导，让它自动填充清单；然后切换到 YAML 模式，仅编辑网络部分，并在 YAML 模式下提交。这样就无需手动编写整个清单。

YAML 模式中显示的清单取决于所使用的控制台（请参阅两个管理界面）：新控制台显示 `StorageCluster`，旧版控制台显示 `CephCluster`。请在对应路径编辑网络块。

1 创建 Multus 网络

若要以接近主机的速度通过专用物理网卡传输存储流量，请使用 **Kube-OVN Underlay** 网络，而非 Overlay 网络。Underlay 会将专用物理网卡直接桥接到存储网络，不产生封装开销。

Underlay 要求预先准备物理网络：每个存储节点都需要专用网卡（不能承载 SSH 等其他流量），并且必须配置上游交换机端口/VLAN 和网关。然后创建一个桥接网络（`ProviderNetwork`）和一个 **VLAN**（`Vlan`），供存储子网接入。有关物理环境的前提条件以及如何创建桥接网络和 VLAN，请参阅准备 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，请在 `rook-ceph` 命名空间中创建一个 `NetworkAttachmentDefinition`，并创建一个后端 Underlay 子网。以下示例创建公共网络；如果还要隔离复制流量，请使用不同的名称和不同的 CIDR/VLAN，为集群网络重复此操作。

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

```yaml
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"
    }
```

创建提供该网络所需地址的 Underlay Kube-OVN 子网。将 `vlan` 设置为已准备好的 `Vlan`，使用与物理网络匹配的 `cidrBlock` / `gateway`，并使 `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
```

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

2 打开创建集群向导

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。在创建集群步骤中，照常配置设备类别、存储设备和组件放置方式，让向导填充清单的其余部分。

3 切换到 **YAML** 模式

将向导从表单模式切换为 **YAML** 模式。清单已根据你的选择填充完毕。

4 编辑网络部分

在清单中添加或替换网络块。设置 `provider: multus` 和 `selectors`，将 Ceph 公共/集群角色映射到你的 NAD。

- 新 **Web** 控制台 — 清单为 `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 — 清单为 `CephCluster`；编辑 `spec.network`（字段相同，但不包含 `spec.ceph` 封装）：

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

如果只需要将客户端流量迁移到专用网络，请仅保留 `public` 选择器，并省略 `cluster`。

在与 `selectors` 同级的位置添加 `addressRanges` 块，明确指定 CIDR 范围。Rook 会尝试自动检测所接入网络的 CIDR，但在 Kube-OVN Underlay 网络上，此检测并不可靠，可能会失败，导致集群卡在“Configuring Ceph Mons”状态。指定 `addressRanges` 可避免此问题：

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

使用已创建的 Underlay 子网的 CIDR（如果没有单独隔离集群网络，则只需 `public`）。

TIP

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

5 在 YAML 模式下提交

保持在 YAML 模式下提交向导（点击创建集群）。提交前不要切回表单模式，否则网络字段可能无法保留。

> **WARNING**
>
> 在新控制台中，请在此处的 `StorageCluster` 中配置网络，不要事后编辑 `CephCluster`。当集群由 `StorageCluster` 管理时，`spec.network` 是受管理的字段，对 `CephCluster` 的手动编辑会被 operator 还原。

验证

创建存储集群后，从 ACP 和 Ceph 两侧验证网络配置。

（新控制台）检查 StorageCluster 网络设置

如果集群由 `StorageCluster` 管理，请确认所需的配置。Ceph `StorageCluster` 在 `rook-ceph` 命名空间中始终命名为 `ceph-cluster`（两者均由准入机制强制执行）：

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

输出应包含预期的 provider 和选择器：

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

检查生效的 CephCluster 网络设置

此检查适用于两个控制台，因为 `CephCluster` 是 Rook 实际操作的资源：

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

`provider` 和 `selectors` 应与配置的内容一致。在新控制台中，它们还应与管理该资源的 `StorageCluster` 一致。

检查 Ceph Pod 是否已连接到网络

确认 Ceph 守护进程 Pod（例如 MON 和 OSD）带有 Multus 网络连接注解，并报告预期的接口：

```
kubectl get pod -n rook-ceph -l app=rook-ceph-osd \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.annotati
ons.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

创建或重启一个同时挂载 CephFS PVC 和 RBD PVC 的测试工作负载，然后验证：

- Pod 是否成功启动
- 文件系统是否可读写

- CSI 或工作负载日志中是否未出现与挂载相关的错误

故障排除建议

如果启用 Multus 后存储集群无法变为健康状态，请先检查以下项目：

1. **NAD** 不存在 — 选择器引用的 `NetworkAttachmentDefinition` 在 `rook-ceph` 命名空间中缺失，或者 `<namespace>/<nad-name>` 引用拼写错误。
2. 缺少选择器 — 已设置 `provider: multus`，但未定义 `public` 或 `cluster` 选择器。至少需要一个选择器。
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** 耗尽 — 为 NAD 提供支持的 IPAM 池（例如配置的 CIDR 或地址范围）没有可供新 Ceph Pod 使用的空闲地址。
5. 节点无法访问 **Underlay** — Ceph CSI driver 使用主机网络运行，因此每个节点（而不是工作负载 Pod）都必须能够通过 Underlay 网络路由到 OSD 地址。如果挂载卡住，请验证节点的 Kube-OVN Underlay 路由，并确认节点可以访问 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集群。由于运行中的集群无法切换网络模型，因此应通过重新创建存储集群来解决网络配置错误，而不是在原集群上直接修补。

MinIO 到 Rook Ceph RGW 数据迁移指南

目录

1. 概述和架构
   1.1 架构说明
   1.2 同步策略
2. 前提条件和准备工作
   2.1 验证 Alauda Storage Volume Replication 安装并提取版本（关键前提条件）
   2.2 收集 MinIO 源信息
   2.3 创建 Ceph RGW 用户（目标端）
3. 部署配置（Manifest）
   3.1 创建 Rclone Config Secret
   3.2 定义迁移 Job
4. 执行流程
   阶段 1：初始全量同步
   阶段 2：增量同步
   阶段 3：最终切换
5. 故障排查与调优建议

1. 概述和架构

本文介绍如何使用已安装的 **Alauda Storage Volume Replication** Operator 镜像中内置的 Rclone 组件，将 Kubernetes 中高可用（HA）MinIO 部署的所有数据迁移到 Rook Ceph RGW。

1.1 架构说明

- 安全地复用镜像：此方法会调度一个 Kubernetes Job，复用经过安全修复的 `build-harbor.alauda.cn/acp/volsync` Operator 镜像，覆盖默认入口点，并直接调用内部 Rclone binary，以执行标准的 **S3** 到 **S3** API 级对象同步。

1.2 同步策略

采用三阶段策略：**“全量 -> 增量 -> 切换”**：

1. 初始同步（全量同步）：保持服务在线，同时迁移历史数据。
2. 增量同步：保持服务在线并多次运行，以追赶新添加或已修改的数据。
3. 切换同步：停止写入，执行严格的一致性检查，并完成最终切换。

---

2. 前提条件和准备工作

2.1 验证 Alauda Storage Volume Replication 安装并提取版本（关键前提条件）

执行此方案之前，请确保 **“Alauda Storage Volume Replication”** 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 Storage Volume Replication** 的详细安装说明，请参阅配置 Persistent Volume Claims 的灾难恢复。

获取版本的方法： 登录集群管理控制台，导航至 **Marketplace / OperatorHub**，打开 **Installed Operators**，找到 Alauda Storage Volume Replication Operator，并记录显示的版本（例如

`v0.8.0` 或 `v0.9.0`）。将此值保存为 `<OPERATOR_VERSION>`，供后续配置使用。

2.2 收集 MinIO 源信息

- **Endpoint**：MinIO 内部 Service 地址（例如：`http://minio.tenant-ns.svc:9000`）。
- 凭据：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，以创建专用迁移用户并授予 bucket 创建权限。

```
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: "*"
```

为遵循最小权限原则，仅为此迁移账户保留 bucket 级别的权限，不要授予 `user` 管理能力。

提取并解码 Ceph 凭据，供后续使用：

```
# 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（Manifest）

建议在目标端命名空间（Ceph RGW）或专用运维命名空间中部署迁移任务。

在应用两个 Manifest 前，先创建它们共用的命名空间：

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

> 部署位置建议（重要）
> 在迁移期间，Rclone 会在本地读取数据进行处理，然后将其上传到目标 S3 集群。因此，运行迁移任务的集群所在网络位置会直接影响带宽和完成时间。建议将 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** 值两侧添加引号。

```
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，调用经过安全修复的 Alauda Storage Volume Replication 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`，这也是定义范围最清晰、最安全的方式。
> - 本文有意保留 `source-minio:` -> `dest-ceph:`，以支持完整的实例级迁移（源端所有可见 bucket）。

> - 风险提示：`remote:` 的范围更广。如果凭据权限过大，预期范围之外的 bucket 也可能被迁移。务必先在测试环境中验证，并考虑在生产切换时改用显式的 `remote:bucket[/prefix]` allowlist 模式。

务必将以下 **YAML** 中的 `<OPERATOR_VERSION>` 替换为在第 **2.1** 节中获取的实际版本。

```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 operator image. Version must exactly mat
ch 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 Imag
e / 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 R
GW fragmentation
            - "--fast-list"         # use ListObjectsV2 to reduce API req
uests
            - "--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

为追赶全量同步期间新生成的数据，请根据需要重复此步骤。

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 会比较 Size 和 ModTime，以判断是否需要传输。现有的未修改文件会被跳过。

阶段 3：最终切换

1. 停止写入：在应用层停止向 MinIO 写入，或通过负载均衡器/Ingress 阻断写入流量。
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"`，然后将 `- "--checksum"` 追加到 `args`。这样可以避免掩盖失败对象，并优先使用 Size+Checksum（如果可用）进行差异检测。
- 一致性建议（必须）：不要仅依赖对象数量或 ETag。切换前，运行 `rclone check source-minio: dest-ceph: --download --checksum`（或对关键对象执行抽样下载并计算 SHA256），以减少 multipart/ETag 差异导致的误报。
- 重新应用 Job：`kubectl apply -f rclone-job.yaml`

3. 验证并切换：Job 达到 `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>`，验证生效的 `Image`（是否已重写）和 `ImageID`。然后验证 `<OPERATOR_VERSION>`，并检查生效 registry（`imagePullSecrets`）的网络和身份验证情况。
Pod OOMKilled	在大量小对象场景中，`--fast-list` 会批量加载元数据，消耗大量内存。	将 Pod 内存限制提高到 8Gi 以上，或移除 `--fast-list`（这会增加 API 请求并减慢遍历速度）。

症状	技术诊断	解决方案
403 Forbidden	目标 Ceph 用户缺少创建存储桶的权限。	检查 `CephObjectStoreUser` 配置，并确保 `capabilities` 包含 `bucket: "*"`。
dial tcp: lookup "http: no such host"	Secret 配置格式无效；Endpoint URL 带有引号。	编辑 Secret 并移除 **Endpoint URL** 两侧的引号，确保格式严格符合要求：`endpoint = http://...`。
503 Service Unavailable	迁移并发度过高，导致 Ceph RGW 写入压力过大。	减少 Job `args` 中的 `--transfers`，或添加 `--bwlimit` 以限制传输带宽。

使用 Ceph RGW Exporter 监控 Ceph RGW

Alauda Storage Exporter 是一个 Operator，可从 Ceph RGW（RADOS Gateway）Admin Ops API 收集使用情况、存储桶和用户指标，并以 Prometheus 格式公开这些指标。Rook 的内置监控涵盖 Ceph 集群自身的运行状况，但不涵盖对象存储租户：每个用户存储了多少数据、一个存储桶包含多少对象，或配额是否即将耗尽。此 Operator 填补了这一空白。

安装后，Operator 会监视集群中的每个 `CephObjectStore`，并自动为每个对象存储部署一个 exporter，同时创建一个只读 RGW 用户和一个 `ServiceMonitor`。无需手动处理任何凭据。

目录

- 术语
- 前提条件
- 工作原理
- 操作步骤
  - 安装 Operator
  - 验证自动发现的 exporter
  - 验证指标
- 自定义 exporter
- 指标参考
- 故障排除
- 卸载

术语

术语	描述
CephRGWExporter	声明一个 exporter 实例的自定义资源（`ceph.alauda.io/v1alpha1`）。每个实例面向一个 `CephObjectStore`。
Exporter	查询 RGW Admin Ops API 并提供 `/metrics` 的 Pod。Operator 与 exporter 共享同一个容器镜像，区别仅在于启动参数不同。
CephObjectStoreUser	用于配置 RGW 用户的 Rook 资源。exporter 从 Rook 为该用户创建的 Secret 中读取 RGW 端点、访问密钥和密钥。
Admin Ops API	RGW 管理 REST API（`/admin/usage`、`/admin/bucket`、`/admin/user`）。exporter 发出的所有请求均使用 AWS Signature V4 签名。

前提条件

- 已安装 **Alauda Storage**，且至少有一个 `CephObjectStore` 处于 `Ready` 阶段。请参阅 Ceph Distributed Storage。
- **Alauda Storage Exporter** Operator 软件包已在 **OperatorHub** 中可用。在 air-gapped 集群上，请先按照 Upload Packages 中的说明上架该软件包。
- 若要在平台中查看指标，集群上必须安装监控插件（Prometheus 或 VictoriaMetrics）。请参阅 Monitoring Installation。不安装该插件 exporter 也可以正常工作，但不会有任何组件抓取这些指标。

工作原理

1. Operator 监视 `CephObjectStore` 资源。对于它创建的每个对象存储 `<store>`，Operator 会在同一命名空间中创建：
   - 一个名为 `rgw-exporter-user-<store>` 的 `CephObjectStoreUser`，具有只读的 `bucket`、`user`、`metadata` 和 `usage` 功能；

- 一个名为 `rgw-exporter-<store>` 的 `CephRGWExporter`，并添加 `ceph.alauda.io/auto-created: "true"` 注解。

2. 对于每个 `CephRGWExporter`，Operator 会协调一个 Deployment、一个 Service（端口为 `9242`，名称为 `http-metrics`）、一个带有命名空间范围 Role 的 ServiceAccount，以及一个 `ServiceMonitor`，所有资源均使用 CR 的名称。
3. exporter 启动时会解析所引用的 `CephObjectStoreUser`，从其 Secret 中读取 `Endpoint`、`AccessKey` 和 `SecretKey`，然后开始轮询 RGW Admin Ops API。Operator 本身不会读取这些密钥。
4. 删除 `CephObjectStore` 后，自动创建的 `CephRGWExporter` 和 `CephObjectStoreUser` 也会随之删除。手动创建的资源不会受到影响。

操作步骤

1 安装 Operator

1. 转到 **Administrator**。
2. 在左侧导航栏中，单击 **Marketplace** > **OperatorHub**。
3. 找到 **Alauda Storage Exporter**，然后单击 **Install**。选择 `alpha` channel，将其安装到 `rook-ceph` 命名空间（Rook-Ceph 运行所在的命名空间），并将升级审批设置为 **Manual**。有关安装对话框的详细说明，请参阅 Operator。
   或者，也可以在集群的 **control plane node** 上从命令行应用 `Subscription`。`rook-ceph` 命名空间中已包含由 Rook-Ceph 创建的 `OperatorGroup`，因此无需创建新的资源。

```
cat << EOF | kubectl apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ceph-rgw-exporter
  namespace: rook-ceph
spec:
  channel: alpha
  installPlanApproval: Manual
  name: ceph-rgw-exporter
  source: platform
  sourceNamespace: cpaas-system
EOF
```

审批生成的 `InstallPlan`：

```
ip=$(kubectl -n rook-ceph get subscription ceph-rgw-exporter -o js
onpath='{.status.installPlanRef.name}')
kubectl -n rook-ceph patch installplan $ip --type=merge -p '{"spe
c":{"approved":true}}'
```

4. 验证 Operator 正在运行。

```
kubectl -n rook-ceph get csv | grep ceph-rgw-exporter
kubectl -n rook-ceph get deploy ceph-rgw-exporter-operator
```

预期输出：

```
ceph-rgw-exporter.v1.0.3   Alauda Build of Ceph RGW Exporter   1.
0.3   Succeeded
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
ceph-rgw-exporter-operator   1/1     1            1           3m
```

2 验证自动发现的 exporter

Operator 启动后约一分钟内，每个 `CephObjectStore` 都会有一个 exporter。

1. 列出 exporter。

```
kubectl -n rook-ceph get cephrgwexporters
```

预期输出（此示例中的 `ceph-cluster-object` 是对象存储名称）：

```
NAME                                PHASE     READY   ENDPOINT
AGE
rgw-exporter-ceph-cluster-object    Running   1       http://rook-c
eph-rgw-ceph-cluster-object.rook-ceph.svc:7480   2m
```

exporter Pod 启动期间，`PHASE` 为 `Pending`；Pod 就绪后为 `Running`；如果无法解析所引用的 `CephObjectStoreUser` 或其 Secret，则为 `Failed`。详细信息会记录在 `status.conditions` 中。

2. 确认已为 exporter 配置的只读 RGW 用户。

```
kubectl -n rook-ceph get cephobjectstoreuser rgw-exporter-user-cep
h-cluster-object -o jsonpath='{.spec.capabilities}{"\n"}{.status.p
hase}{"\n"}'
```

预期输出：

```
{"bucket":"read","metadata":"read","usage":"read","user":"read"}
Ready
```

3. 确认子资源。

```
kubectl -n rook-ceph get deploy,svc,servicemonitor -l app.kubernet
es.io/name=ceph-rgw-exporter
```

3 验证指标

1. 将 exporter Service 转发到本地端口，并请求指标端点。

```
kubectl -n rook-ceph port-forward svc/rgw-exporter-ceph-cluster-ob
ject 9242:9242 &
curl -s http://localhost:9242/-/ready
curl -s http://localhost:9242/metrics | grep -v '^#' | head
```

预期输出：

```
ok
radosgw_usage_ops_total{bucket="bucket_root",category="list_bucket
s",owner="anonymous",store="ceph-cluster-object"} 142126
radosgw_usage_scrape_duration_seconds 0.271208467
radosgw_usage_user_quota_enabled{store="ceph-cluster-object",user
="rgw-exporter-user-ceph-cluster-object"} 0
radosgw_usage_user_total_bytes{store="ceph-cluster-object",user="r
gw-exporter-user-ceph-cluster-object"} 0
...
```

只有对象存储中至少存在一个存储桶后，存储桶指标（`radosgw_usage_bucket_*`）才会显示。

2. 如果已安装监控插件，`ServiceMonitor` 会被自动发现：它带有 `prometheus: kube-prometheus` 标签，平台使用该标签选择抓取目标，并且默认每隔 `300s` 抓取 `http-metrics` 端口。经过一个抓取间隔后，在平台的监控面板中或通过 PromQL 查询 `radosgw_usage_scrape_duration_seconds`，以确认数据已被摄取。有关如何使用自定义指标，请参阅 Management of Metrics。

自定义 exporter

自动创建的 `CephRGWExporter` 使用默认设置。当需要不同的配置时，你可以直接编辑其 `spec`，也可以创建自己的 `CephRGWExporter`（使用除 `rgw-exporter-<store>` 之外的任意名称）。自动发现逻辑绝不会修改或删除手动创建的资源。

```
cat << EOF | kubectl apply -f -
apiVersion: ceph.alauda.io/v1alpha1
kind: CephRGWExporter
metadata:
  name: <name>
  namespace: rook-ceph
spec:
  cephObjectStoreUserRef:
    name: <cephObjectStoreUser>
    namespace: rook-ceph
  store: <store>
  port: 9242
  timeout: 60
  logLevel: INFO
  tagList:
    - team
  replicas: 1
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 500m
      memory: 512Mi
  serviceMonitor:
    enabled: true
    interval: 300s
EOF
```

参数

参数	说明
cephObjectStoreUserRef	必填。exporter 使用其凭证的 `CephObjectStoreUser`。该用户必须至少对 `bucket`、`user`、`metadata` 和 `usage` 具有 `read` 权限；请参阅创建 Ceph 对象存储用户。 `namespace` 默认为 `CephRGWExporter` 所在的命名空间。当用户位于其他命名空间时，Operator 会创建从该命名空间读取其 Secret 所需的 Role 和 RoleBinding。

参数	说明
store	每个指标上的 `store` 标签值，也是用于 AWS Signature V4 签名的地域。请使用 `CephObjectStore` 名称。默认为 `us-east-1`。
adminEntry	RGW Admin Ops API 的路径前缀。默认为 `admin`。仅当已在网关上自定义 `rgw_admin_entry` 时才需更改。
port	exporter 监听的端口。Service 和探针也使用此端口。默认为 `9242`。
timeout	向 RGW 发出的每个请求的超时时间，以秒为单位。默认为 `60`。对于包含大量用户或存储桶的对象存储，请增大此值。
insecure	当 RGW endpoint 使用自签名证书时，跳过 TLS 证书验证。默认为 `false`。
logLevel	可设置为 `DEBUG`、`INFO`、`WARNING` 或 `ERROR`。默认为 `INFO`。`DEBUG` 会记录发送到 RGW 的每个请求 URL。
tagList	作为附加标签公开在存储桶指标上的存储桶标签键。缺失的标签会以空字符串形式发出，以确保标签集保持稳定。默认不设置。
replicas, **resources**, **nodeSelector**, **tolerations**, **image**, **imagePullSecrets**	exporter Deployment 的标准 Pod 模板设置。`image` 默认为 Operator 自身的镜像。
serviceMonitor	`enabled`（默认为 `true`）控制是否创建 `ServiceMonitor`；将其设置为 `false` 会删除现有资源。`interval`（默认为 `300s`）表示抓取间隔。`labels` 用于向 `ServiceMonitor` 添加额外标签。

指标参考

所有指标都带有 `store` 标签。使用情况指标和存储桶指标还带有 `bucket` 和 `owner` 标签，并且会为 `tagList` 中的每个键额外添加一个标签。

指标	说明
`radosgw_usage_ops_total`	按所有者、存储桶和类别统计的操作总数。
`radosgw_usage_successful_ops_total`	成功操作的数量。
`radosgw_usage_sent_bytes_total`	RGW 发送到客户端的字节数。
`radosgw_usage_received_bytes_total`	RGW 从客户端接收的字节数。
`radosgw_usage_bucket_bytes`	存储桶的大小，以字节为单位。
`radosgw_usage_bucket_utilized_bytes`	存储桶实际使用的字节数。
`radosgw_usage_bucket_objects`	存储桶中的对象数量。
`radosgw_usage_bucket_shards`	存储桶的索引分片数量。
`radosgw_usage_bucket_quota_enabled`, `radosgw_usage_bucket_quota_size`, `radosgw_usage_bucket_quota_size_bytes`, `radosgw_usage_bucket_quota_size_objects`	存储桶配额设置。
`radosgw_usage_user_total_objects`	用户拥有的对象数量。
`radosgw_usage_user_total_bytes`	用户存储的字节数。
`radosgw_usage_user_quota_enabled`, `radosgw_usage_user_quota_size`, `radosgw_usage_user_quota_size_bytes`, `radosgw_usage_user_quota_size_objects`	用户配额设置。

指标	说明
`radosgw_usage_user_bucket_quota_enabled`, `radosgw_usage_user_bucket_quota_size`, `radosgw_usage_user_bucket_quota_size_bytes`, `radosgw_usage_user_bucket_quota_size_objects`	每用户的默认存储桶配额设置。
`radosgw_usage_scrape_duration_seconds`	上次抓取期间从 RGW 收集所有指标所用的时间。

配额大小为 `-1` 表示配额不受限制。

查询示例：

```
# Storage used per RGW user
sum by (user) (radosgw_usage_user_total_bytes{store="ceph-cluster-objec
t"})

# Ten largest buckets
topk(10, radosgw_usage_bucket_bytes)

# Request rate by category over the last 5 minutes
sum by (category) (rate(radosgw_usage_ops_total[5m]))
```

故障排除

症状	原因和处理措施
`CephRGWExporter` 一直为 `Pending`，且 Pod 日志中显示 `must contain Endpoint, AccessKey, and SecretKey`	引用的 `CephObjectStoreUser` 尚未处于 `Ready` 状态，或者其 Secret 缺失。检查 `kubectl -n rook-ceph get cephobjectstoreuser`，并等待 Rook 完成用户预配。
指标端点仅返回 `radosgw_usage_scrape_duration_seconds`	RGW 用户缺少 Admin Ops 权限，或者 `store` 的值与网关的地域约定不匹配，导致请求被拒绝。设置 `logLevel: DEBUG` 并

症状	原因和处理措施
	检查 exporter 日志。请注意，Rook 仅在创建用户时应用 `capabilities`；如需更改这些设置，请重新创建用户。
在大型对象存储上抓取超时	exporter 会为每个 RGW 用户发出一个请求。调高 `timeout`，并增大 `serviceMonitor.interval`。
尽管 `curl /metrics` 正常工作，平台仪表板中仍无数据	此集群上未安装监控插件，或者 `serviceMonitor.enabled` 已设置为 `false`。

卸载

在 **Marketplace** > **OperatorHub** 中删除 Operator，或从 `rook-ceph` 命名空间中删除 `Subscription` 和 `ClusterServiceVersion`。Exporter Deployment、Service 和 `ServiceMonitor` 对象由其 `CephRGWExporter` 所有，并会在删除自定义资源时被垃圾回收。自动创建的 RGW 用户不会自动删除；如果不再需要，请显式删除：

```
kubectl -n rook-ceph delete cephrgwexporters --all
kubectl -n rook-ceph get cephobjectstoreusers | grep rgw-exporter-user-
kubectl -n rook-ceph delete cephobjectstoreuser rgw-exporter-user-<store>
```

Ceph 是 Red Hat, Inc. 或其子公司在美国及其他国家/地区的商标或注册商标。 Rook® 是 Linux Foundation 的注册商标。 Alauda 是独立供应商。本产品与 LF Projects, LLC 或 Linux Foundation 无关联，也未获得其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于标识目的。

LVM 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 TopoLVM

- 前提条件
- 限制
- 操作步骤

从 LVM 存储中移除磁盘、设备类和

- 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

本章介绍 **LVM Storage**。Marketplace 软件包名称为 **Alauda Container Platform LVM Volume Provisioner**。有关静态配置的整盘或分区 PersistentVolumes，请参阅 Local Storage。有关如何在 Local Storage、LVM Storage 和 Ceph Distributed Storage 之间进行选择，请参阅 Overview。

TopoLVM 是专为 Kubernetes 设计的 Container Storage Interface（CSI）插件，旨在提供高效、便捷的节点本地 LVM 卷管理。

主要功能和优势：

- 本地卷管理：TopoLVM 专注于管理 Kubernetes 节点上的本地存储设备（例如磁盘和 SSD）。与传统网络存储相比，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），从而根据 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，无需手动干预，从而显著简化运维并降低复杂性。
- 与 **Kubernetes** 深度集成：作为 CSI 插件，TopoLVM 可无缝集成 Kubernetes 存储管理 API，使用户能够直接通过 PersistentVolumeClaims 等标准 Kubernetes 资源对象管理本地卷。

总之，TopoLVM 解决了在 Kubernetes 中使用本地存储时常见的挑战，例如手动管理、缺乏拓扑感知以及动态分配能力不足。它为数据库和缓存等需要高性能本地存储的应用提供了更高效、更易用的解决方案。

TopoLVM 是 topolvm Community 的一个项目。

Alauda 是独立供应商。本产品与 topolvm 项目无关联，也未获得其认可或赞助。所有商标均归其各自所有者所有，此处仅用于标识。

安装

LVM Storage 是一种基于 Community TopoLVM CSI driver 的节点本地存储系统。它可以根据集群节点上的 LVM 卷组动态配置持久卷。Marketplace 软件包名称为 **Alauda Container Platform LVM Volume Provisioner**。

目录

前提条件

操作步骤

部署 Alauda Container Platform Storage Essentials

部署存储

前提条件

- 存储集群中的每个节点都必须安装 lvm2 软件包。如果尚未安装，请在节点上执行 `yum install -y lvm2` 命令。
- 下载与平台架构对应的 **Alauda Container Platform Storage Essentials** 安装包。
- 使用上架软件包机制上架 **Alauda Container Platform Storage Essentials** 安装包。
- 下载与平台架构对应的 **Alauda Container Platform LVM Volume Provisioner** 安装包。
- 使用上架软件包机制上架 **Alauda Container Platform LVM Volume Provisioner** 安装包。

- 可选：如果要使用 **Selection Device**，请安装 **Alauda Container Platform Local Disk Provisioner**。请参阅安装本地存储。

操作步骤

1 部署 Alauda Container Platform Storage Essentials

1. 登录并进入 **Administrator** 页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Storage Essentials**，点击 **Install**，然后进入 **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`：集群中的所有命名空间共享一个 Operator 实例，用于创建和管理，资源使用量更低。
Installation Place	选择 `Recommended`，Namespace 仅支持 **acp-storage**。
Upgrade Strategy	`Manual`：Operator Hub 中有新版本时，需要手动确认才能将 Operator 升级到最新版本。

2 部署存储

1. 前往 **Administrator**。
2. 在左侧导航栏中，点击 **Storage Management** > **Local Storage**。
3. 点击 **Configure Now**。
4. 在 **Install Operator** 向导页面中，点击 **Start Deployment**。
   - 页面自动进入下一步时，表示 Operator 部署成功。

- 如果部署失败，请根据界面提示解决问题，然后点击 **Clean Up** 并重新部署 Operator。

5. 在 **Create Cluster** 向导页面中，添加设备。
   若要使用 **Selection Device**，请先安装 **Alauda Container Platform Local Disk Provisioner**。该组件会提供 LVM Storage 向导所需的设备发现结果。请参阅安装本地存储。
   安装该 operator 前，**Selection Device** 处于禁用状态。**Legacy Selection Device** 无需安装 Alauda Container Platform Local Disk Provisioner；它会列出节点上的磁盘，但不提供发现结果。

参数	说明
Select Node	至少有 1 个裸盘的节点。
Device Class	每个设备类别对应一组特征相同的存储设备。建议根据磁盘类型填写名称，例如 *hdd*、*ssd*。
Device Type	仅支持磁盘类型。
Storage Device	例如，*/dev/sda*。如果有多个磁盘，可以逐个添加。
Snapshot	启用后，支持创建 PVC 快照，并使用快照配置新的 PVC，以便快速备份和恢复业务数据。 如果创建存储时未启用快照，之后仍可在存储集群详情页的 **Operations** 部分按需启用。 注意：使用前，请确保已为当前集群部署 Volume Snapshot Plugin。

- 点击 Next。页面自动进入下一步时，表示集群部署成功。
- 如果创建失败，请根据界面提示操作，并及时清理资源。

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

参数	说明
Name	存储类的名称，在当前集群中必须唯一。

参数	说明
Display Name	便于识别或筛选的名称，例如存储类的中文说明。
Device Class	在 TopoLVM 中，设备类别用于对存储设备进行分类。每个设备类别对应一组特征相同的存储设备。如果没有特殊要求，可以使用集群中的 **Auto-Allocated** 设备类别。
File System	- **XFS** 是一种高性能日志文件系统，擅长处理并行 I/O 工作负载，支持处理大型文件并提供流畅的数据传输。 - **EXT4** 是 Linux 中的日志文件系统，提供 extent 文件存储方式并支持处理大型文件。文件系统容量最高可达 1 EiB，支持的最大文件大小为 16 TiB。
Recycling Policy	持久卷的回收策略。 - Delete：删除持久卷声明时，绑定的持久卷也会被删除。 - Retain：即使删除持久卷声明，绑定的持久卷仍会保留。
Access Mode	ReadWriteOnce (RWO)：可由单个节点以读写模式挂载。
Allocation Project	此类持久卷声明只能在特定项目中创建。 如果暂时未分配项目，之后也可以进行 **Updated**。

7. 点击 **Next**，等待资源创建完成。

TopoLVM 是 topolvm community 的一个项目。 Alauda 是独立供应商。本产品与 topolvm 项目无关联，也未获其认可或赞助。所有商标均为其各自所有者的财产，此处仅用于识别目的。

操作指南

设备管理

前提条件

添加设备

监控与告警

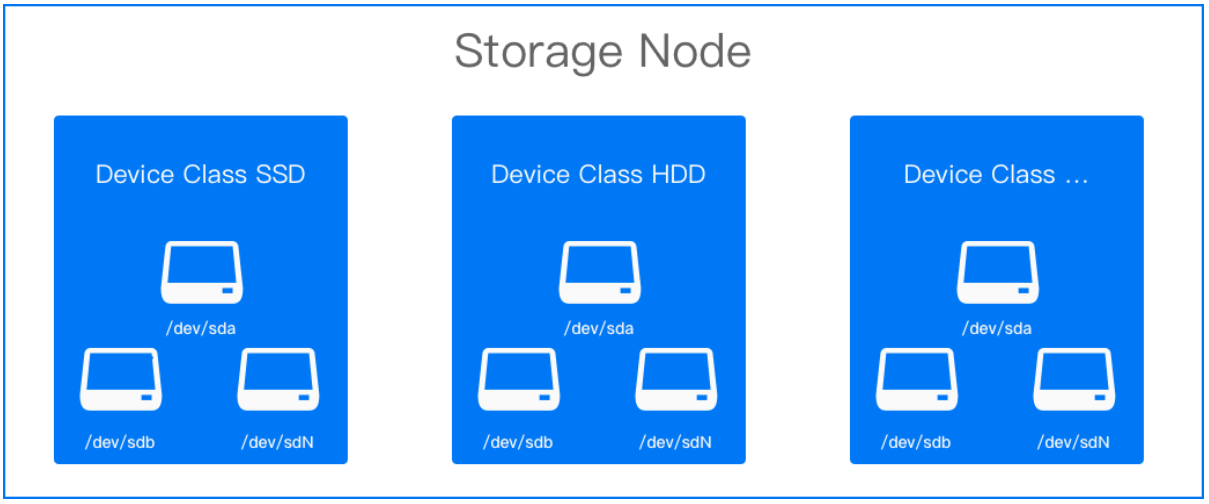
监控

告警

设备管理

无论是初始部署还是资源扩容，都需要将节点上的可用磁盘映射为存储设备，以供使用和管理。

具有相似特征的存储设备通常会集中使用，这些设备在 LVM Storage 中归类为设备类。使用设备类等同于直接使用磁盘，可确保零损耗和高性能，同时降低应用感知以及对特定设备的依赖。



目录

前提条件

添加设备

前提条件

- 创建 LVM Storage 集群时，至少必须添加 1 个设备类（deviceClasses.classes），其中包括设备类中的设备。
- 节点上必须至少存在 1 个裸磁盘。

添加设备

要使用选择设备，请先安装 **Alauda Container Platform Local Disk Provisioner**。请参阅安装本地存储。旧版选择设备无需该 operator。

1. 前往管理员。
2. 在左侧导航栏中，单击存储管理 > 本地存储。
3. 在详情页签中，单击添加存储节点。
4. 根据以下说明配置相关参数。

参数	描述
存储节点	至少具有 1 个裸磁盘的节点。
设备类	每个设备类对应一组具有相同特征的存储设备；建议根据磁盘的性质为其命名，例如 *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，而不是硬盘上的分区，否则会导致错误。

5. 单击添加。
   注意：如果设备类因缺少已添加的设备而处于 `Unavailable` 状态，可以执行以下操作。
6. 切换到存储设备页签，然后单击添加存储设备。
7. 根据界面提示添加设备。
8. 单击添加。

监控与告警

LVM Storage 提供开箱即用的监控指标采集和告警功能。启用平台监控组件后，可以根据存储集群、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决策，完善的告警机制则有助于确保存储系统稳定运行。

目录

监控
- 性能监控
- 容量监控

告警
- 配置通知
- 处理告警
- 事后分析

监控

性能监控

默认情况下，平台会为 LVM Storage 采集常用的性能监控指标，例如读写带宽、IOPS 和延迟。可以在存储管理下本地存储页面的监控选项卡中查看这些指标的实时监控数据。平台通过图形和图表直观展示这些指标，使管理员能够清晰观察当前存储性能并快速识别潜在问题。

容量监控

由于 LVM Storage 只能使用节点上本地可用的存储资源，因此用户必须在声明 LVM Storage 卷之前确保节点具有足够的可用容量，以避免因过度声明而导致问题。

为此，平台在 LVM Storage 的详情部分提供了按设备类型分类的详细容量监控。用户可以查看以数值和图形格式清晰显示的可用存储空间。如果任何设备类型显示可用容量不足，则应先清理空间或添加其他磁盘设备，然后再使用 LVM Storage。

告警

平台包含一组默认告警策略。当资源出现异常或监控数据达到告警阈值时，系统会自动触发告警。预配置的告警策略可以有效覆盖常见的运维需求，包括集群健康状态和设备类型容量告警。

配置通知

为确保及时接收告警，应在运维中心配置通知策略。通知可以通过电子邮件、短信或其他方式发送给相关人员，提醒其立即关注并解决问题或防止故障发生。用户可以直接从运维中心界面访问通知策略设置。有关配置告警的详细说明，请参阅[创建告警策略]文档。

处理告警

- 如果存储集群的健康状态变为 `Alert`，管理员应立即进行调查。详情部分提供了用于排查和解决这些问题的信息。常见原因包括节点服务异常或特定设备类型出现问题。

检查项	对应状态	原因
健康状态	告警	由节点服务异常或设备类型问题引起。

检查项	对应状态	原因
服务状态	未知	节点处于 `notready` 状态，可能是由于网络故障或电源故障导致。
设备类型状态	不可用	正在使用的磁盘可能不是裸盘，也可能已缺失。

- 在告警选项卡上触发的实时告警需要及时处理，即使存储集群当前状态显示为 `Healthy`。快速响应可以防止问题升级为更严重的故障。下表说明了告警级别及其含义：

告警级别	含义
严重	表示导致平台服务中断或数据丢失的重大问题，影响严重。
主要	表示可能影响平台功能和正常业务运行的已知问题。
警告	表示存在运维问题风险；需要及时干预，以避免影响正常业务运行。

事后分析

告警历史记录了之前触发且不再需要立即处理的所有告警。进行事后分析时，请考虑以下问题：

- 事件发生时观察到了哪些具体异常？
- 是否有特定告警反复出现？未来如何主动防止这些告警发生？
- 是否有特定时段告警数量激增，并且与外部因素或运维事件相关？是否应据此调整运维策略？

实用指南

使用 Velero 备份和恢复 TopoLVM

前提条件

限制

操作步骤

从 LVM 存储中移除磁盘、设备类

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

使用 Velero 备份和恢复 TopoLVM 文件系统 PVC

Velero 支持备份和恢复 TopoLVM 文件系统的 Persistent Volume Claims（PVC）和 Persistent Volumes（PV）。此功能已集成到平台中。

本指南专门适用于 TopoLVM 文件系统 PVC。

目录

前提条件

限制

操作步骤

步骤 1：配置备份存储库

步骤 2：执行备份

步骤 3：恢复集群

前提条件

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

限制

1. S3 存储必须有足够的可用空间，以存储目标集群中的所有 PV 数据。
2. 在恢复过程中，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

步骤 1：配置备份存储库

1. 确保兼容 S3 的存储可用。
2. 导航到 管理员 > 集群管理 > 备份和恢复 > 备份存储库。
3. 使用对象存储凭据创建备份存储库。

步骤 2：执行备份

1. 为要备份的 PVC 及其关联 Pod 添加标签：
   Velero 需要一个 Pod 来恢复文件系统 PVC。该 Pod 会挂载 PVC，以便 Velero 导入数据；如果没有 Pod，PVC 将保持 Pending 状态。对于复杂应用，请暂停应用，并将 PVC 挂载到轻量级 Pod（例如 Nginx）以进行备份/恢复，然后在恢复完成后恢复原始应用配置。

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

2. 转到 备份和恢复，创建新的备份：

   - 选择 备份 **Kubernetes** 资源和 **PVC** 数据卷。
   - 选择包含要备份数据的命名空间。
   - 使用以下设置配置备份：

```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 bucket（例如 MinIO）中的数据：

```
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/
```

步骤 **3**：恢复集群

1. 在目标集群中，将同一个 S3 bucket 配置为备份所使用的 bucket。Velero 将自动检测现有备份。
2. 导航到 备份和恢复，创建恢复任务：
   - 选择要恢复的命名空间。
   - 在高级配置中，根据需要将原始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 名称与原始集群一致。
   - PVC 中的应用数据完整且可访问。

从 LVM 存储中移除磁盘、设备类卷组或节点

本文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 LVM Storage 中的存储节点。

根据故障范围，本文涵盖以下场景：

- 从 `device-class volume group` 中移除 `volume group device`
- 从 `device class` 中移除 `device-class volume group`
- 移除 TopoLVM 存储节点

> 风险警告
>
> - 本文中的操作步骤会直接删除 PVC、PV 和 `logicalvolumes.topolvm.cybozu.com` 等存储资源。删除这些资源后，受影响卷中的数据通常无法恢复。请提前备份数据。
> - 继续操作之前，请确认已识别出正确的磁盘、设备类卷组或节点，并为受影响的工作负载安排维护窗口。

目录

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

用户场景

前提条件

检查设备类卷组是否仍可恢复

操作步骤

步骤 1：停止 topolvm-operator

步骤 2：查找受影响的 LVM 逻辑卷

步骤 3：查找受影响的 PVC 和 PV

步骤 4：停止使用受影响 PVC 的工作负载

步骤 5：删除受影响的 Kubernetes 存储资源

步骤 6：清理残留的 LVM 逻辑卷

步骤 7：从 LVM 卷组中移除缺失的物理卷

步骤 8：更新 TopolvmCluster 资源

步骤 9：更新 lvmdconfig ConfigMap

步骤 10：启动 topolvm-operator

步骤 11：验证卷组设备已移除

场景 2：从设备类中移除设备类卷组

用户场景

前提条件

检查设备类卷组是否已完全损坏

操作步骤

步骤 1：停止 topolvm-operator

步骤 2：查找受影响的 PVC 和 PV

步骤 3：停止使用受影响 PVC 的工作负载

步骤 4：删除受影响的 Kubernetes 存储资源

步骤 5：更新 TopolvmCluster 资源

步骤 6：更新 lvmdconfig ConfigMap

步骤 7：启动 topolvm-operator

步骤 8：验证设备类卷组已移除

场景 3：移除 TopoLVM 存储节点

用户场景

前提条件

操作步骤

步骤 1：停止 topolvm-operator

步骤 2：查找受影响的 PVC 和 PV

步骤 3：停止使用受影响 PVC 的工作负载

步骤 4：删除受影响的 Kubernetes 存储资源

步骤 5：更新 TopolvmCluster 资源

步骤 6：更新 lvmdconfig ConfigMap

步骤 7：启动 topolvm-operator

步骤 8：验证节点已移除

术语

术语	描述
设备类	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。
设备类卷组	节点上的一个 `LVM volume group`，表示设备类在该节点上的存储资源。
卷组设备	节点上的磁盘。在 LVM 中，它对应于物理卷。

选择正确的场景

使用下表确定适用于您环境的场景。

条件	场景 1：移除卷组设备	场景 2：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。	节点已无法恢复，或者您已决定将其从 TopolvmCluster 中永久移除。
卷组状态	目标 `LVM volume group` 仍然存在且仍可识别。	目标 `LVM volume group` 已不存在或已无法识别，但节点仍保留。	该节点及其上的所有设备类卷组将一起移除。

条件	场景 1：移除卷组设备	场景 2：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。	既不保留节点，也不保留该节点上的任何设备类卷组。

场景 1：从设备类卷组中移除卷组设备

用户场景

- 当 `device-class volume group` 未完全损坏，但卷组中的一个或多个 `volume group devices` 发生故障且必须移除时，请使用此操作步骤。

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，并且卷组中至少保留一个正常的卷组设备。

检查设备类卷组是否仍可恢复

在目标节点上运行以下命令：

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

参数：

- `<vg-name>`：目标 LVM 卷组的名称。

如果命令返回正常输出，或者仅报告部分 PV 缺失但卷组仍然存在，则设备类卷组未完全损坏，您可以按照此操作步骤执行。

示例输出：

```
WARNING: Couldn't find device with uuid VJes6j-2a8V-8Cxf-eW84-yEJK-K24A-yBc9OD.
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-da94c6185c33/b6b331f0-5242-420f-a531-df90628bef80 while checking used and assumed devices.
```

操作步骤

步骤 1：停止 **topolvm-operator**

在控制平面节点上运行以下命令，以防止 operator 在清理期间协调资源：

```
kubectl -n nativestor-system scale --replicas 0 deployment topolvm-operator
```

步骤 2：查找受影响的 **LVM** 逻辑卷

在目标节点上运行以下命令，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 卷组的名称。
- `<path-to-disks>`：故障磁盘的设备路径，例如 `/dev/vdb`。如果需要匹配多个磁盘，请使用 `|` 分隔它们。

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
```

步骤 3：查找受影响的 PVC 和 PV

在控制平面节点上运行以下命令，通过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 逻辑卷的名称。

示例输出：

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

在输出中：

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

步骤 4：停止使用受影响 PVC 的工作负载

确定受影响的 PVC 后，停止使用这些 PVC 的工作负载，并确认所有相关 Pod 均已停止，然后再继续。

步骤 5：删除受影响的 Kubernetes 存储资源

在控制平面节点上运行以下命令：

```
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` 资源的名称。它与 `<pv-name>` 相同。

如果查询结果包含多个资源，请按照映射关系逐一删除。

如果某个资源无法正常删除，请根据需要添加 `--force`。

步骤 6：清理残留的 LVM 逻辑卷

在目标节点上运行以下命令，检查是否仍有逻辑卷存在：

```
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 卷组的名称。
- `<path-to-disks>`：故障磁盘的设备路径，例如 `/dev/vdb`。如果需要匹配多个磁盘，请使用 `|` 分隔它们。

如果命令仍返回输出，请逐一删除剩余的逻辑卷：

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

参数：

- `<vg-name>`：目标 LVM 卷组的名称。
- `<lv-name>`：要移除的逻辑卷的名称。

如有需要，添加 `--force`。

步骤 7：从 LVM 卷组中移除缺失的物理卷

在目标节点上运行以下命令：

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

参数：

- `<vg-name>`：目标 LVM 卷组的名称。

如有需要，添加 `--force`。

步骤 8：更新 TopolvmCluster 资源

在控制平面节点上运行以下命令：

```
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`。

修改前：

```
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
```

修改后：

```
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
```

步骤 9：更新 **lvmdconfig ConfigMap**

在控制平面节点上运行以下命令：

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

参数：

- `<node-name>`：目标节点的名称。

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。

修改前：

```
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":[]}'
```

修改后：

```
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":[]}'
```

步骤 10：启动 topolvm-operator

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-op
erator
```

步骤 11：验证卷组设备已移除

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com t
opolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认已移除的卷组设备不再显示在目标节点的 `deviceStates` 中。

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进行恢复时，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，节点上至少保留一个其他设备类卷组。

检查设备类卷组是否已完全损坏

在目标节点上运行以下命令：

```
vgs
```

如果输出中不再显示目标 `LVM volume group`，则表示设备类卷组已完全损坏，您可以按照此操作步骤执行。

> **注意**
>
> 此场景会从节点中移除整个设备类卷组，而不仅仅是移除该卷组中的单个卷组设备。

操作步骤

步骤 1：停止 **topolvm-operator**

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system scale --replicas 0 deployment topolvm-operator
```

步骤 2：查找受影响的 PVC 和 PV

在控制平面节点上运行以下命令，以查找目标节点上与指定存储类关联的 PV、PVC 和 `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>`：目标节点的名称。
- `<storageclass-name>`：与目标设备类卷组关联的存储类名称。如果涉及多个存储类，请针对每个存储类分别运行查询。

示例输出：

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

在输出中：

- 第 1 列是 `PersistentVolume` 和 `logicalvolumes.topolvm.cybozu.com` 资源的名称。
- 第 2 列和第 3 列是 `PersistentVolumeClaim` 的命名空间和名称。

如果目标设备类卷组关联了多个存储类，请针对每个存储类重复此查询以及后续清理步骤。

步骤 3：停止使用受影响 PVC 的工作负载

确定受影响的 PVC 后，停止使用这些 PVC 的工作负载，并确认所有相关 Pod 均已停止，然后再继续。

步骤 4：删除受影响的 Kubernetes 存储资源

在控制平面节点上运行以下命令：

```
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` 资源的名称。它与 `<pv-name>` 相同。

如果查询结果包含多个资源，请根据映射关系逐个删除。

如果某个资源无法正常删除，请根据需要添加 `--force`。

步骤 5：更新 TopolvmCluster 资源

在控制平面节点上运行以下命令：

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

在编辑器中，从目标节点移除设备类卷组。例如，在 `nodeName: 192.168.140.13` 的配置中，移除 `className: hdd` 条目及其关联的 `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
```

如果移除的 `className` 条目包含 `default: true`，请将另一个保留的类指定为默认类。

步骤 6：更新 lvmdconfig ConfigMap

在控制平面节点上运行以下命令：

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

参数：

- `<node-name>`：目标节点的名称。

在编辑器中，从 `lvmd.yaml` 和 `status.json` 中移除与目标设备类卷组对应的配置。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":[]}'
```

如果移除的 `className` 条目包含 `default: true`，请将另一个保留的类指定为默认类。

步骤 7：启动 **topolvm-operator**

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-op
erator
```

步骤 8：验证设备类卷组已移除

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com t
opolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认目标节点上不再显示已移除的设备类卷组，并确认剩余的设备类卷组运行状况正常。

场景 3：移除 **TopoLVM** 存储节点

用户场景

- 目标节点已无法恢复，或者您已决定将其从 TopolvmCluster 中永久移除。

前提条件

- 您已决定不保留目标节点上的任何设备类卷组。
- 使用目标节点上本地卷的工作负载已停止，或已迁移到其他节点。
- 移除目标节点后，剩余节点仍可继续承载所需的工作负载。

操作步骤

步骤 1：停止 **topolvm-operator**

在控制平面节点上运行以下命令：

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

步骤 2：查找受影响的 **PVC** 和 **PV**

在控制平面节点上运行以下命令，查找与目标节点关联的 PV、PVC 和 `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>`：目标节点的名称。

步骤 3：停止使用受影响 **PVC** 的工作负载

确定受影响的 PVC 后，停止使用这些 PVC 的工作负载，并确认所有相关 Pod 均已停止，然后再继续。

步骤 4：删除受影响的 Kubernetes 存储资源

在控制平面节点上运行以下命令：

```
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` 资源的名称。它与 `<pv-name>` 相同。

如果查询结果包含多个资源，请根据映射关系逐个删除这些资源。

如果资源无法正常删除，请根据需要添加 `--force`。

步骤 5：更新 TopolvmCluster 资源

在控制平面节点上运行以下命令：

```
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。

删除前：

```
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
```

删除后：

```
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
```

步骤 6：更新 **lvmdconfig ConfigMap**

在控制平面节点上运行以下命令：

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

参数：

- `<node-name>`：目标节点的名称。

在编辑器中，删除整个 `lvmd.yaml` 和 `status.json` 部分。不要保留已移除节点的任何配置或状态。

步骤 7：启动 topolvm-operator

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-operator
```

步骤 8：验证节点已移除

在控制平面节点上运行以下命令：

```
kubectl -n nativestor-system get topolvmclusters.topolvm.cybozu.com topolvm -o jsonpath="{.status.nodeStorageState}" | jq
```

确认目标节点不再显示在 `status.nodeStorageState` 中。例如，`192.168.140.13` 不应再出现在输出中。

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

操作步骤

使用默认的 `lvcreate-option-class`

创建自定义 lvcreate-option-class（可选）

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。

本地存储

介绍

介绍

设备发现不会创建卷

相关信息

安装

安装

先决条件

操作步骤

后续步骤

操作指南

发现本地磁盘

使用 **LocalVolume** 进行配置

使用 **LocalV**

前提条件
检查是否已存在发现对象
创建 LocalVolumeDiscovery
检查发现结果
后续步骤

前提条件
配置 Filesystem 卷
配置 Block 卷
绑定 PersistentVolumeClaim
添加磁盘或节点

前提条件
配置 Block 卷
绑定 Persistent
添加磁盘或节点

实用指南

移除卷并卸载

删除 LocalVolume 或 LocalVolumeSet
卸载 Alauda Container Platform Local Disk Provisioner

介绍

Local Storage 根据已连接到集群节点的磁盘和分区静态配置 PersistentVolumes。它由 **Alauda Container Platform Local Disk Provisioner** operator 提供。

当工作负载必须通过标准 PersistentVolumeClaim 使用特定的本地磁盘或分区时，请使用 Local Storage。该 operator 会创建一个 StorageClass，并为每个匹配的设备创建一个 PersistentVolume。随后，工作负载通过 PersistentVolumeClaims 绑定这些卷。

Local Storage 不会跨节点复制数据。节点故障会导致卷不可用。磁盘故障可能导致数据丢失。

如需在 Local Storage、LVM Storage 和 Ceph Distributed Storage 之间进行选择，请参阅 概览。

目录

设备发现不会创建卷

相关信息

设备发现不会创建卷

该 operator 可以发现未使用的块设备，并为每个节点将结果写入一个 `LocalVolumeDiscoveryResult` 对象。发现功能仅用于清点设备，不会创建 PersistentVolumes、StorageClasses 或文件系统。

只有在应用选择设备的 `LocalVolume` 或 `LocalVolumeSet` 后，才会创建 PersistentVolumes。相关过程请参阅 指南。

相关信息

- 概览
- 安装
- LVM Storage
- Ceph Distributed Storage

安装

安装 **Alauda Container Platform Local Disk Provisioner**，以便集群清点本地磁盘，并在之后从这些磁盘中配置 PersistentVolumes。

本页面仅介绍 operator。安装完成后，使用 指南 来发现设备并创建 PersistentVolumes。`LocalVolume` 和 `LocalVolumeSet` provisioning 不属于 operator 安装的一部分。

目录

先决条件

操作步骤

安装 Alauda Container Platform Local Disk Provisioner

后续步骤

先决条件

- 下载与平台架构对应的 **Alauda Container Platform Local Disk Provisioner** 安装包。
- 使用 Upload Packages 机制上架 **Alauda Container Platform Local Disk Provisioner** 安装包。

操作步骤

1 安装 Alauda Container Platform Local Disk Provisioner

1. 登录，进入管理员页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **Alauda Container Platform Local Disk Provisioner**，点击安装，进入安装 **Alauda Container Platform Local Disk Provisioner** 页面。
   配置参数：

参数	推荐配置
Channel	默认 channel 为 `stable`。
Installation Mode	`Cluster`：集群中的所有 namespace 共享单个 Operator 实例进行创建和管理，从而降低资源使用量。
Installation Place	选择 `Recommended`，Namespace 仅支持 **acp-storage**。
Upgrade Strategy	`Manual`：Operator Hub 中有新版本时，需要手动确认才能将 Operator 升级到最新版本。

后续步骤

- 在指南中发现可用磁盘并置备 PersistentVolumes。
- LVM Storage 和 Ceph Internal Mode 的选择设备使用此 operator 进行发现。请参阅安装 LVM Storage 和以内部模式部署。
- 若要使用 Local Storage Block 卷为 Ceph OSD 提供后端存储，请在指南中置备卷，然后参阅使用 Local Storage 以内部模式部署。

操作指南

安装 **Alauda Container Platform Local Disk Provisioner** 后，使用这些操作指南盘点磁盘并配置 PersistentVolumes。

设备发现功能会列出未使用的块设备。它不会创建 PersistentVolumes、StorageClasses 或文件系统。只有在应用 `LocalVolume` 或 `LocalVolumeSet` 时，才会创建 PersistentVolumes。

发现本地磁盘

- 前提条件
- 检查是否已存在发现对象
- 创建 LocalVolumeDiscovery
- 检查发现结果
- 后续步骤

使用 **LocalVolume** 进行配置

- 前提条件
- 配置 Filesystem 卷
- 配置 Block 卷
- 绑定 PersistentVolumeClaim
- 添加磁盘或节点

使用 **LocalV**

- 前提条件
- 配置 Block 卷
- 绑定 Persistent
- 添加磁盘或节点

发现本地磁盘

`LocalVolumeDiscovery` 会清点集群节点上未使用的块设备，并为每个节点写入一个 `LocalVolumeDiscoveryResult` 对象。使用结果为 `LocalVolume` 选择设备路径，或为 `LocalVolumeSet` 选择过滤器。

发现操作不会创建 PersistentVolumes、StorageClasses 或文件系统。具有 `status.state: Available` 的设备可用于配置。它不是 PersistentVolume。

目录

前提条件

检查是否已存在发现对象

创建 LocalVolumeDiscovery

检查发现结果

后续步骤

前提条件

- 已安装 **Alauda Container Platform Local Disk Provisioner**。请参阅安装。
- 您可以对集群运行 `kubectl`。

检查是否已存在发现对象

安装 **Alauda Container Platform Local Disk Provisioner** 后，如果同时安装了 **Alauda Container Platform Storage Essentials** 且尚不存在 `LocalVolumeDiscovery`，平台会在 `acp-storage` 中创建一个名为 `auto-discover-devices` 的 `LocalVolumeDiscovery`。

在控制平面节点上：

```
kubectl -n acp-storage get localvolumediscovery
```

如果列出了 `auto-discover-devices`，请勿创建其他对象。在下一节中检查结果。若要限制扫描的节点，请编辑该对象上的 `spec.nodeSelector`。

创建 LocalVolumeDiscovery

仅当 `acp-storage` 中不存在 `LocalVolumeDiscovery` 时，才创建一个该对象。

```
cat << EOF | kubectl apply -f -
apiVersion: local.storage.openshift.io/v1alpha1
kind: LocalVolumeDiscovery
metadata:
  name: auto-discover-devices
  namespace: acp-storage
spec:
  nodeSelector: 1
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-01
              - worker-02
              - worker-03
  tolerations: 2
    - operator: Exists
EOF
```

1 可选。运行发现操作的节点。请从 `kubectl get node` 获取主机名。如果省略，发现操作将在每个节点上运行。

2 可选。传递给发现 DaemonSet 的容忍度。`operator: Exists` 匹配每个污点。

检查发现结果

在控制平面节点上：

```
kubectl -n acp-storage get localvolumediscoveryresult
```

示例输出：

```
NAME                          AGE
discovery-result-worker-01    21m
discovery-result-worker-02    21m
discovery-result-worker-03    21m
```

每个选定的节点都会获得一个 `LocalVolumeDiscoveryResult`。在一个节点上检查设备：

```
kubectl -n acp-storage get localvolumediscoveryresult discovery-result-wo
rker-01 -o yaml
```

在 `LocalVolume` 中使用 `status.discoveredDevices[].deviceID`（一个 `/dev/disk/by-id/...` 路径）作为稳定路径。不要使用 `path` 值（例如 `/dev/sdb` 或 `/dev/disk/by-path/...`）。节点重启后，这些名称可能会发生变化。

`status.discoveredDevices[].status.state` 的值为以下之一：

状态	含义
`Available`	设备未使用，可以进行配置。尚不存在 PersistentVolume。
`NotAvailable`	设备已在使用中，不得进行配置。
`Unknown`	operator 无法确定设备是否空闲。

列表中的设备不可移除、不是只读设备、未挂载、不是启动磁盘，并且没有子分区。发现操作仍然不会创建卷。

后续步骤

- 使用 LocalVolume 为列出的路径配置资源。
- 使用 LocalVolumeSet 为匹配的磁盘配置资源。

使用 LocalVolume 进行配置

`LocalVolume` 会为列出的每个设备路径静态配置一个 PersistentVolume。如果不存在在 `spec.storageClassDevices[].storageClassName` 中指定名称的 StorageClass，operator 还会创建该 StorageClass。

当你知道确切的磁盘时，请使用 `LocalVolume`。对于基于过滤器的配置，请参阅使用 LocalVolumeSet 进行配置。

目录

前提条件

配置 Filesystem 卷

配置 Block 卷

绑定 PersistentVolumeClaim

添加磁盘或节点

前提条件

- 已安装 **Alauda Container Platform Local Disk Provisioner**。请参阅安装。
- 每个目标设备都已连接到节点、未挂载且没有分区。
- 优先使用发现本地磁盘中介绍的稳定路径，例如 `/dev/disk/by-id/...`。不要使用简短的 `/dev/sdX` 名称或 `/dev/disk/by-path/...` 路径。这些名称可能会在节点重启后发生变

化。

配置 **Filesystem** 卷

LocalVolume 是静态的。provisioner 会在你列出的路径中查找设备。

> **NOTE**
>
> 不要对同一设备使用不同的 StorageClass 名称。这样会为一个磁盘创建多个 PersistentVolume。

在控制平面节点上：

```
cat << EOF | kubectl apply -f -
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: local-disks
  namespace: acp-storage (1)
spec:
  nodeSelector: (2)
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-01
              - worker-02
              - worker-03
  storageClassDevices:
    - storageClassName: local-sc (3)
      forceWipeDevicesAndDestroyAllData: false (4)
      volumeMode: Filesystem (5)
      fsType: xfs (6)
      devicePaths: (7)
        - /dev/disk/by-id/<wwn-id>
EOF
```

1 安装 **Alauda Container Platform Local Disk Provisioner** 的命名空间。默认值为 `acp-storage`。

2 可选。具有所列磁盘的节点。请从 `kubectl get node` 获取主机名。如果省略，operator 会在每个节点上查找这些路径。

3 PersistentVolume 的 StorageClass。如果不存在，operator 会创建它。请使用能够唯一标识这组磁盘的名称。

4 当 `true` 时，operator 会在使用前擦除所列设备上的分区表、文件系统签名和数据。默认值为 `false`。仅当可以销毁这些设备时，才设置 `true`。

5 `Filesystem` 或 `Block`。

6 首次挂载时创建的文件系统。仅当 `volumeMode` 为 `Filesystem` 时设置此项。

7 设备路径。使用来自 `LocalVolumeDiscoveryResult.status.discoveredDevices[].deviceID` 的 `/dev/disk/by-id/...`。不要使用 `/dev/sdX` 或 `/dev/disk/by-path/...`。

> **WARNING**
>
> `forceWipeDevicesAndDestroyAllData: true` 会销毁所列设备上的现有数据。

验证 PersistentVolume：

```
kubectl get pv
```

示例输出：

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS   REASON   AGE
local-pv-n8xlrd2a   100Gi      RWO            Delete           Available
local-sc                88m
local-pv-tc78vc73   100Gi      RWO            Delete           Available
local-sc                82m
```

> **NOTE**

编辑 `LocalVolume` 不会更改已存在的 PersistentVolume 上的 `fsType` 或 `volumeMode`。此更改会造成破坏性影响。

配置 Block 卷

要配置原始 Block 设备，请设置 `volumeMode: Block`，并省略 `fsType`：

```
      volumeMode: Block
      devicePaths:
        - /dev/disk/by-id/<wwn-id>
```

使用可标识此 Block 集的 StorageClass 名称，例如 `localblock`。PersistentVolumeClaim 还必须设置 `volumeMode: Block`。

绑定 PersistentVolumeClaim

在将使用该卷的项目中创建声明。请求不得超过设备容量。

Filesystem 声明：

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem 1
  resources:
    requests:
      storage: 100Gi 2
  storageClassName: local-sc 3
EOF
```

1. 必须匹配 `LocalVolume` `volumeMode`。对于 Block 卷，请使用 `Block`。
2. 必须小于或等于 PersistentVolume 容量。
3. 必须匹配 `spec.storageClassDevices[].storageClassName`。

确认声明已绑定：

```
kubectl get pvc local-pvc
```

添加磁盘或节点

添加磁盘或节点需要编辑同一个 `LocalVolume`，而不是执行单独的操作步骤。

- 添加磁盘：将 `/dev/disk/by-id/...` 路径追加到 `spec.storageClassDevices[].devicePaths`。
- 添加节点：将节点主机名添加到 `spec.nodeSelector.nodeSelectorTerms`，然后列出该节点的设备路径。

应用编辑后，等待处于 `Available` 状态的新 PersistentVolume：

```
kubectl get pv -l storage.openshift.com/owner-name=local-disks
```

使用 LocalVolumeSet 进行配置

`LocalVolumeSet` 会根据匹配筛选条件的设备配置 PersistentVolume。如果不存在由 `spec.storageClassName` 指定名称的 StorageClass，operator 会创建该 StorageClass。

当磁盘应根据大小、类型或类似属性声明，而不是通过显式路径列表声明时，请使用 `LocalVolumeSet`。有关精确路径，请参阅使用 LocalVolume 进行配置。

WARNING

请谨慎使用 `LocalVolumeSet`。当 operator 根据本地磁盘配置 PersistentVolume 时，它可能会声明所有匹配 `deviceInclusionSpec` 的设备。**Alauda Container Platform Local Disk Provisioner** 必须是节点上唯一管理这些磁盘的组件。不要创建多个选择同一节点的 `LocalVolumeSet`。如果必须限制每个节点配置的设备数量，请设置 `maxDeviceCount`。

`LocalVolumeSet` 不支持逻辑卷管理（LVM）设备。不要将其指向 LVM 卷。对于由 LVM 支持的动态卷，请使用 LVM Storage。

目录

前提条件

配置 Block 卷

绑定 PersistentVolumeClaim

添加磁盘或节点

前提条件

- 已安装 **Alauda Container Platform Local Disk Provisioner**。请参阅安装。
- 设备发现正在运行，以便确认哪些磁盘是 `Available`。请参阅发现本地磁盘。
- 匹配的设备已连接、未挂载，并且在选择整个磁盘时没有子分区。

配置 **Block** 卷

此示例会创建名为 `localblock` 的 Block StorageClass，并根据匹配的整个磁盘创建 PersistentVolume。

在控制平面节点上：

```
cat << EOF | kubectl apply -f -
apiVersion: local.storage.openshift.io/v1alpha1
kind: LocalVolumeSet
metadata:
  name: local-block-disks
  namespace: acp-storage 1
spec:
  storageClassName: localblock 2
  volumeMode: Block 3
  maxDeviceCount: 2 4
  nodeSelector: 5
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-01
              - worker-02
              - worker-03
  deviceInclusionSpec:
    deviceTypes: 6
      - disk
    minSize: 10Gi
    maxSize: 2Ti
EOF
```

1 安装 **Alauda Container Platform Local Disk Provisioner** 的命名空间。默认值为 `acp-storage`。

2 用于配置 PersistentVolume 的 StorageClass。如果不存在，operator 会创建该 StorageClass。请使用能够唯一标识此集合的名称。

3 `Block` 或 `Filesystem`。对于 Ceph OSD 消费和其他原始设备用户，请使用 `Block`。

4 可选。每个节点配置的匹配设备的最大数量。如果省略，则会声明所选节点上的每个匹配设备。

5 可选。运行匹配的节点。请从 `kubectl get node` 获取主机名。如果省略，则会在每个节点上运行匹配。

6 要包含的设备类型：`disk`、`part`、`loop` 或 `mpath`。如果列表为空，则只选择 `disk`。不支持 LVM 设备。

可选的 `deviceInclusionSpec` 筛选条件：

字段	值来源
`deviceTypes`	`disk`、`part`、`loop` 或 `mpath`。为空表示仅使用 `disk`。
`deviceMechanicalProperties`	`Rotational` 或 `NonRotational`。默认使用两者。
`minSize`	设备大小下限。为空时，默认为 `1Gi`。
`maxSize`	设备大小上限。
`models`	磁盘 `MODEL` 字符串中不区分大小写的子字符串。从 `LocalVolumeDiscoveryResult.status.discoveredDevices[].m` 或节点上 `lsblk -d -o NAME,TYPE,SIZE,MODEL,VENDOR,SERIAL` `MODEL` 列复制值。类似 `SAMSUNG` 的值会匹配 `SAMSUNG MZVLB5`
`vendors`	磁盘 `VENDOR` 字符串中不区分大小写的子字符串。从 `LocalVolumeDiscoveryResult.status.discoveredDevices[].v` 或同一 `lsblk` 命令的 `VENDOR` 列复制值。此列通常是诸如 `ATA` `NVMe` 的总线名称，而不是磁盘品牌。品牌通常位于 `MODEL` 中。

如果 `MODEL` 或 `VENDOR` 为空（Virtio 和许多云磁盘中很常见），请不要设置 `models` 或 `vendors`。这些筛选条件不会匹配空字符串。

验证 PersistentVolume：

```
kubectl get pv
```

示例输出：

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM        STORAGECLASS   REASON   AGE
local-pv-1cec77cf   100Gi      RWO            Delete           Available
localblock              88m
local-pv-2ef7cd2a   100Gi      RWO            Delete           Available
localblock              82m
```

如果之后插入的磁盘匹配筛选条件，并且 `maxDeviceCount` 仍允许配置，则会自动为其配置。

绑定 PersistentVolumeClaim

在将使用该卷的项目中创建声明。对于 Block 卷，请设置 `volumeMode: Block`。请求不得超过设备容量。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block (1)
  resources:
    requests:
      storage: 100Gi (2)
  storageClassName: localblock (3)
EOF
```

(1) 必须与 `LocalVolumeSet` `volumeMode` 匹配。

(2) 必须小于或等于 PersistentVolume 容量。

3. 必须与 `spec.storageClassName` 匹配。

确认声明已绑定：

```
kubectl get pvc local-block-pvc
```

添加磁盘或节点

- 添加磁盘：在选定的节点上连接与 `deviceInclusionSpec` 匹配的磁盘。当磁盘为 `Available` 且未超过 `maxDeviceCount` 时，operator 会配置 PersistentVolume。无需编辑路径列表。
- 添加节点：在 `LocalVolumeSet` 以及清点该节点的 `LocalVolumeDiscovery` 上，将主机名添加到 `spec.nodeSelector` 中。请参阅发现本地磁盘。

实用指南

删除已配置本地卷或卸载 **Alauda Container Platform Local Disk Provisioner** 的操作步骤。若要盘点磁盘并创建 PersistentVolumes，请参阅指南。

移除卷并卸载

删除 LocalVolume 或 LocalVolumeSet

卸载 Alauda Container Platform Local Disk Provisioner

移除卷并卸载

卸载 **Alauda Container Platform Local Disk Provisioner** 之前，请先删除已置备的本地卷。卸载 operator 不会删除你的 `LocalVolume`、`LocalVolumeSet`、StorageClass 或 PersistentVolume 对象。不要通过卸载 operator 来清除磁盘。

置备和绑定声明的操作请参阅 指南。

目录

删除 LocalVolume 或 LocalVolumeSet

卸载 Alauda Container Platform Local Disk Provisioner

删除 LocalVolume 或 LocalVolumeSet

只要仍有其管理的 PersistentVolume，operator 就会在 `LocalVolume` 和 `LocalVolumeSet` 上保留 finalizer。已绑定的卷以及具有 `Retain` 回收策略的卷会阻止删除。

前提条件

- 已安装 **Alauda Container Platform Local Disk Provisioner**。
- 知道 `acp-storage` 中的对象名称。

操作步骤

1. 列出该对象管理的 PersistentVolume：

```
kubectl get pv -l storage.openshift.com/owner-name=<local-volume-name>
```

2. 删除绑定到这些卷的所有 PersistentVolumeClaim。等待声明全部删除。
3. 删除该对象：

```
kubectl -n acp-storage delete localvolume <local-volume-name>
```

对于 `LocalVolumeSet`：

```
kubectl -n acp-storage delete localvolumeset <local-volume-set-name>
```

4. 如果对象仍处于 `Terminating`，请再次列出其管理的剩余 PersistentVolume。删除仍存在的卷，或更改其回收策略。只有在这些卷全部删除后，才能移除 finalizer。

具有 `Delete` 回收策略的可用卷会由 operator 释放，然后删除。具有 `Retain` 回收策略的卷不会自动删除；只有在不再需要其中的数据后，才删除这些卷。

> **WARNING**
>
> 如果 PersistentVolume 使用 `Delete` 回收策略，删除 `LocalVolume` 或 `LocalVolumeSet` 可能会销毁底层磁盘上的数据。请确认没有工作负载仍需要这些卷。

卸载 Alauda Container Platform Local Disk Provisioner

仅当您已删除不再需要的 `LocalVolume` 和 `LocalVolumeSet` 对象，或已决定保留这些对象后，才能卸载 operator。

删除 operator subscription 时，平台创建的名为 `auto-discover-devices` 的 `LocalVolumeDiscovery` 可能会被移除。卸载不会删除您的 `LocalVolume` 和 `LocalVolumeSet` 对象。

操作步骤

1. 登录并转到管理员页面。
2. 单击 **Marketplace** > **OperatorHub**。
3. 找到 **Alauda Container Platform Local Disk Provisioner**，然后打开已安装的 operator。
4. 卸载 operator。如果控制台提供删除 operands 或 custom resources 的选项，请勿选择这些选项。
5. 确认 `acp-storage` 中的 operator Pod 已消失：

```
kubectl -n acp-storage get csv | grep local-storage
```