存储

Ceph 分布式存储

介绍

功能概览

存储方案对比

核心概念

实用指南

部署规划

部署架构

安全考虑

基础设施要求

架构

技术架构

MinIO 对象存储

介绍

核心概念

操作指南

安装

前提条件

实用指南

架构

核心组件：

结论：

TopoLVM 本地存储

介绍

安装

前提条件

操作指南

实用指南

Ceph 分布式存储

介绍

部署规划

架构

架构

技术架构

安装

创建标准型集群

前提条件

注意事项

操作步骤

相关操作

创建 Stretch 类型集群

术语

典型部署方案

约束与限制

前提条件

操作步骤

相关操作

核心概念

核心概念

Rook Operator

Ceph CSI

Ceph 模块功能

操作指南

访问存储服务

前提条件

操作步骤

后续操作

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

更新组件部署配

重启存储组件

添加设备/设

监控与告警

监控

告警

实用指南

替换或移除设备

前提条件

约束与限制

操作步骤

参考资料

替换或移除存储节点

前提条件

约束与限制

操作步骤

References

配置专用集群

架构

基础设施要求

操作步骤

后续操作

清理分布式存储

注意事项

操作步骤

灾难恢复

更新优化参数

创建 Ceph 对象存储用户

前提条件

池配

为 Ceph RGW 启用 D3N 缓存

背景

前提条件

操作概览

准备本地文件系统作为缓存

在 CephObjectStore 中启用 D3N 缓存

配置传输加密

概述

限制与前提条件

介绍

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

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

目录

功能概览

- 简易部署：提供存储集群的图形化自动部署和管理服务；支持计算与存储的集成部署和解耦部署两种模式。
- 专业运维：提供持久卷快照备份和克隆新卷功能；容量、性能和组件级别的可视化监控；内置告警策略，满足大多数存储运维场景需求。

- 安全可靠：分布式多副本机制保障数据安全和可靠；简单可靠的自动化管理支持存储资源的在线扩容。
- 卓越性能：提供弹性高性能存储服务；支持混合磁盘设备部署，提升存储系统性能和效率。

存储方案对比

平台支持以下两种存储方案，您可以任选其一。

创建存储集群

需求	优势
您可以选择创建标准型集群或扩展型集群	无需额外准备存储方案，配置可在业务集群上完成，节省成本。

接入外部存储

方案一：接入平台内其他业务集群的分布式存储资源，确保存储与业务隔离，便于管理和维护。

方案二：将外部 Ceph 存储资源作为分布式存储接入。

需求（任选其一）	优势
方案一：分布式存储已在其他业务集群部署。	可充分利用跨集群存储资源，避免业务变更干扰。确保存储数据安全稳定，降低运维复杂度。 注意：若接入的存储为不同平台的分布式存储，如灾备环境中的主备平台，请使用外部 Ceph 集成方式。
方案二：平台外部 **Ceph** 存储，版本 **≥ 14.2.3**。	相较于直接创建存储类，使用该方式更便于利用平台界面进行卷快照、扩容等功能操作。

注意：若需维护外部存储的存储池、存储设备等配置，必须在存储集群的管理界面中进行操作。

部署规划

本主题提供了在 Alauda Container Platform (ACP) 上部署 Ceph 分布式存储的规划清单。它总结了架构选择、安全选项、基础设施规模、网络限制和灾难恢复考虑，帮助您在实际安装之前确定部署模型。

有关产品背景，请参见简介和架构。有关部署流程，请参见安装和操作指南下的文档。

目录

部署架构

ACP 分布式存储基于 Ceph 和 Rook。从高层来看，平台结合了以下层次：

- Ceph 守护进程，如 MON、MGR、OSD、MDS 和 RGW，提供块存储、文件存储和对象存储能力
- Rook 和 CSI 组件，用于自动化部署、配置、扩容和生命周期管理
- ACP 平台集成，暴露存储池、可观测性和运维入口

部署前，需决定环境是使用本地集群的存储服务，还是消费外部 Ceph 环境的存储。

内部与外部部署模型

您可以按以下方式规划 ACP 分布式存储：

部署模式	存储服务运行位置	存储集群管理方	适用场景	关键权衡
内部，共存部署	Ceph 组件运行在与业务工作负载相同的 ACP 工作节点上	ACP 平台团队或集群管理员	初期环境、裸金属集群或存储需求尚不明确的情况	部署简单，但应用与存储间资源争用可能性较大
内部，专用节点部署	Ceph 组件运行在同一 ACP 集群内的专用存储或基础设施节点上	ACP 平台团队或集群管理员	生产环境，存储需求可预测且隔离要求较高	运维隔离和容量控制更好，但需要预留更多节点和容量规划

部署模式	存储服务运行位置	存储集群管理方	适用场景	关键权衡
外部部署	ACP 从外部 Ceph 环境消费存储类	独立存储团队、SRE 团队或已有的外部存储所有者	大规模环境、多消费集群或已有独立 Ceph 集群的组织	所有权边界清晰，但跨集群网络、认证和依赖管理更复杂

内部部署更易于推广和管理，因为存储服务和消费工作负载都规划在同一 ACP 环境内。内部部署的首个设计选择是存储是否与业务工作负载共享节点，还是使用专用节点。外部部署适用于需要更强隔离的存储与应用集群，或多个业务集群共享同一存储后端的场景。

主要规划决策点：

- 需要快速部署且能容忍存储与应用共享同一工作节点池时，选择共存部署。
- 存储需求明确，且希望有更清晰的容量控制、故障隔离和维护边界时，选择专用节点部署。
- 存储已由其他团队管理，或单个外部集群需服务多个 ACP 集群时，选择外部部署。

节点角色

规划节点布局时，应区分控制平面节点、基础设施节点和工作节点的职责：

- 控制平面节点负责集群管理功能，除非部署模型明确支持，否则不应作为通用存储节点。
- 基础设施节点适合隔离存储平台组件与业务工作负载。
- 工作节点可在共存部署中承载存储服务，但会增加应用与存储守护进程间的资源争用。

生产环境建议规划至少三个故障域以实现高可用存储服务。尽可能将存储节点分布在机架、区域或主机组中。

安全考虑

部署前确认存储设计是否需要传输加密，并在启用前验证其对运维的影响。

传输加密

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

启用传输加密前，请确认：

- 存储和客户端节点的内核及操作系统支持情况
- 繁忙存储节点的预期 CPU 开销
- 目标硬件上的吞吐量和延迟影响

有关实现细节，请参见配置传输加密。

基础设施要求

最低与推荐配置

创建集群前规划节点数量、存储设备和可用资源。

项目	最低配置	推荐配置
存储节点	3 个节点	3 个或更多节点，分布于多个故障域
存储设备	每节点至少 1 个可用存储设备	每节点多个专用设备，类型和容量保持一致
节点分布	3 个节点可承载 Ceph 服务	3 个故障域，如机架或区域
设备使用	系统盘与存储盘分离	Ceph 数据使用专用裸盘，预留未来扩容空间

集群至少应有三节点，每节点至少一个可用存储设备。生产环境建议跨至少三个故障域部署，并预留足够资源以应对重平衡、修复和未来增长。

资源规模规划

Ceph 存储服务持续消耗 CPU、内存和设备容量。先规划存储守护进程资源，再预留恢复、重平衡、升级和后台任务的额外空间。

基线建议：

- 至少三台存储节点以保证高可用集群
- 预留 MON、MGR、OSD 及启用的 MDS 或 RGW 服务的 CPU 和内存
- 保留新存储池、额外设备和集群恢复的增长空间
- 避免初期部署即接近资源饱和

若设计采用专用存储节点，资源规划更可预测。若存储与业务工作负载共存，应预留额外空间以缓冲峰值负载和节点故障时的争用。

集群总体规划预算

早期规模规划应从集群总体预算出发，而非仅从单组件值开始。下表为三节点高可用集群的规划参考：

部署模式	存储预留总 **CPU**	存储预留总内存	备注
内部，最低基线	24 逻辑 CPU	72 GiB	满足最低部署目标的入门级三节点规划基线
内部，标准基线	30 逻辑 CPU	72 GiB	更适合一般生产规划和未来扩容
内部，性能导向基线	45 逻辑 CPU	96 GiB	适用于从一开始就需要更高吞吐量或更低延迟的场景
外部消费集群	仅按连接和客户端访问需求规划	仅按连接和客户端访问需求规划	存储守护进程运行于 ACP 集群外，ACP 集群主要需网络可达性、凭据和客户端容量

以上数值为集群级规划目标，非精确调度保留。三节点集群的单节点预算可将总数均分至参与存储节点。

以下推荐适合早期规划：

组件	推荐 **CPU**	推荐内存
MON	2 核	3 GiB

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

以上为规划参考，非硬性调度保证。实际需求取决于设备数量、启用服务和负载强度。

如何估算集群规模

规划集群规模时，按以下顺序进行：

1. 选择部署模式：共存、专用节点或外部。
2. 确定最小节点数和故障域布局。
3. 决定是否需要块存储、文件存储、对象存储或混合服务。
4. 从集群总体规划预算开始。
5. 预留额外空间以支持新增设备集、恢复、监控和预期增长。

若同时需要文件和对象服务，或集群将承载重业务负载，应高于最低基线规划。

Pod 调度

Pod 调度规则直接影响弹性。规划集群时应确保：

- 高可用组件分布于不同故障域
- 每个故障域均有可访问存储设备和足够可分配资源
- 新设备集或未来扩容仍可遵循相同调度模式

实际中，仅有三节点不足，节点还需分布合理，避免单一机架、主机组或区域成为单点故障。

存储设备规划

选型时尽量统一设备容量和类型。混合设备增加性能调优和容量规划难度。

遵循以下原则：

- 预留一块系统盘用于操作系统，Ceph 数据使用独立存储设备
- 优先使用裸盘或专用设备，避免共享盘分区
- 控制每节点设备数量，确保恢复和维护可行
- 关注可用容量而非原始容量，因复制机制降低有效存储空间

容量规划还应包含告警阈值和扩容策略。应在集群接近满载前规划扩容，避免满载带来的重平衡压力和恢复难度。

相关运维指导请参见存储池管理和添加设备/设备类。

容量规划

规划集群容量时，应计算可用容量而非原始磁盘容量。复制 Ceph 部署中，部分原始存储用于数据保护。

规划原则：

- 保持可用容量领先于预期业务增长，避免仅在接近满载时扩容
- 预留恢复、重平衡、快照和临时数据峰值的额外空间
- 跨节点和故障域均衡扩容，避免新容量造成利用率偏斜
- 添加新工作负载前，审查当前利用率和预测增长

以下示例为三节点集群、每节点一设备、3 副本数据保护策略的早期规划参考：

每节点设备容量	原始集群容量	**3** 副本近似可用容量
0.5 TiB	1.5 TiB	0.5 TiB
2 TiB	6 TiB	2 TiB
4 TiB	12 TiB	4 TiB

以上仅为示例。可用容量随实际数据保护策略变化，不应作为所有集群设计的通用规则。

二次运维时，应在集群达到告警级别前复核容量。若增长可预测，应提前扩容，避免满载或接近满载状态。

网络要求

Ceph 对网络质量敏感。部署前验证：

- 集群网络能稳定提供复制和恢复流量的吞吐量
- 故障域间延迟符合所选部署模型支持范围
- 存储节点与消费集群间所需端口已开放
- 任何专用网络设计（如基于 Multus 的隔离）已提前确定

若计划隔离存储流量与普通应用流量，部署前确认网络接口、路由策略和运维归属。网络隔离提升安全和性能，但增加设计复杂度。

IPv6 支持

ACP 分布式存储规划需遵循平台选定的集群网络栈。

- 支持单栈 IPv6 环境。
- 双栈规划需在存储部署前验证与 ACP 集群网络设计的兼容性。
- 存储节点和客户端节点应使用相同地址族策略，避免连接和服务发现问题。

若环境使用 IPv6，安装前确认：

- ACP 集群网络已配置支持 IPv6
- 所有存储节点可通过所需 IPv6 路由通信
- 访问存储端点的监控、告警和外部集成也支持 IPv6

IPv6 应作为安装时的架构决策，不应假设现有 IPv4 设计可无须重新验证直接转换。

灾难恢复规划

ACP 分布式存储可根据恢复点目标 (RPO)、恢复时间目标 (RTO) 和站点拓扑规划不同恢复模型。

区域灾备 (Regional-DR)

ACP 支持区域灾备，适用于跨地域或跨站点的异步复制场景，允许少量潜在数据丢失。

规划区域灾备时，提前确认：

- 源集群和目标集群存储及网络设计兼容
- 复制延迟和故障切换预期符合业务恢复目标
- 受保护的工作负载类型明确，如块存储、文件系统或对象数据

实现细节请参见灾难恢复。

Stretch 集群

Stretch 集群仅适用于站点间延迟严格受控且拓扑专门设计的场景。一般规划要求：

- 两个数据站点和一个仲裁或仲裁站点
- 至少五个节点，分布于三个区域
- 集群创建前手动明确故障域标签
- 每个数据站点有足够节点保障存储服务可用性
- 站点间延迟保持低延迟设计范围，通常数据站点间 RTT 不超过 10 毫秒

> **WARNING**
>
> Stretch 集群不应作为长距离、高延迟、多数据中心部署的通用方案。若站点间延迟不可严格控制，应采用专用灾难恢复架构。

ACP 相关 Stretch 集群部署指导请参见创建 Stretch 类型集群。

性能规划

性能规划应基于工作负载特性，而非仅凭设备数量。部署前识别：

- 主要工作负载是块存储、文件存储还是对象存储
- 工作负载是延迟敏感、吞吐量敏感还是容量密集
- 热数据、备份流量或分析任务是否主导集群

还需确认是否需要特殊调优或特定功能设计。例如，对象工作负载可能需单独规划网关容量，部分环境可能需缓存导向或专用集群设计。

后续步骤

完成规划后，进入匹配所选部署模型的部署指南：

内部部署

- 共存部署，请参见创建标准类型集群。
- Stretch 集群部署，请参见创建 Stretch 类型集群。
- 专用节点部署，请参见为分布式存储配置专用集群。

外部部署

- 从其他集群或外部 Ceph 环境消费存储服务，请参见访问存储服务。

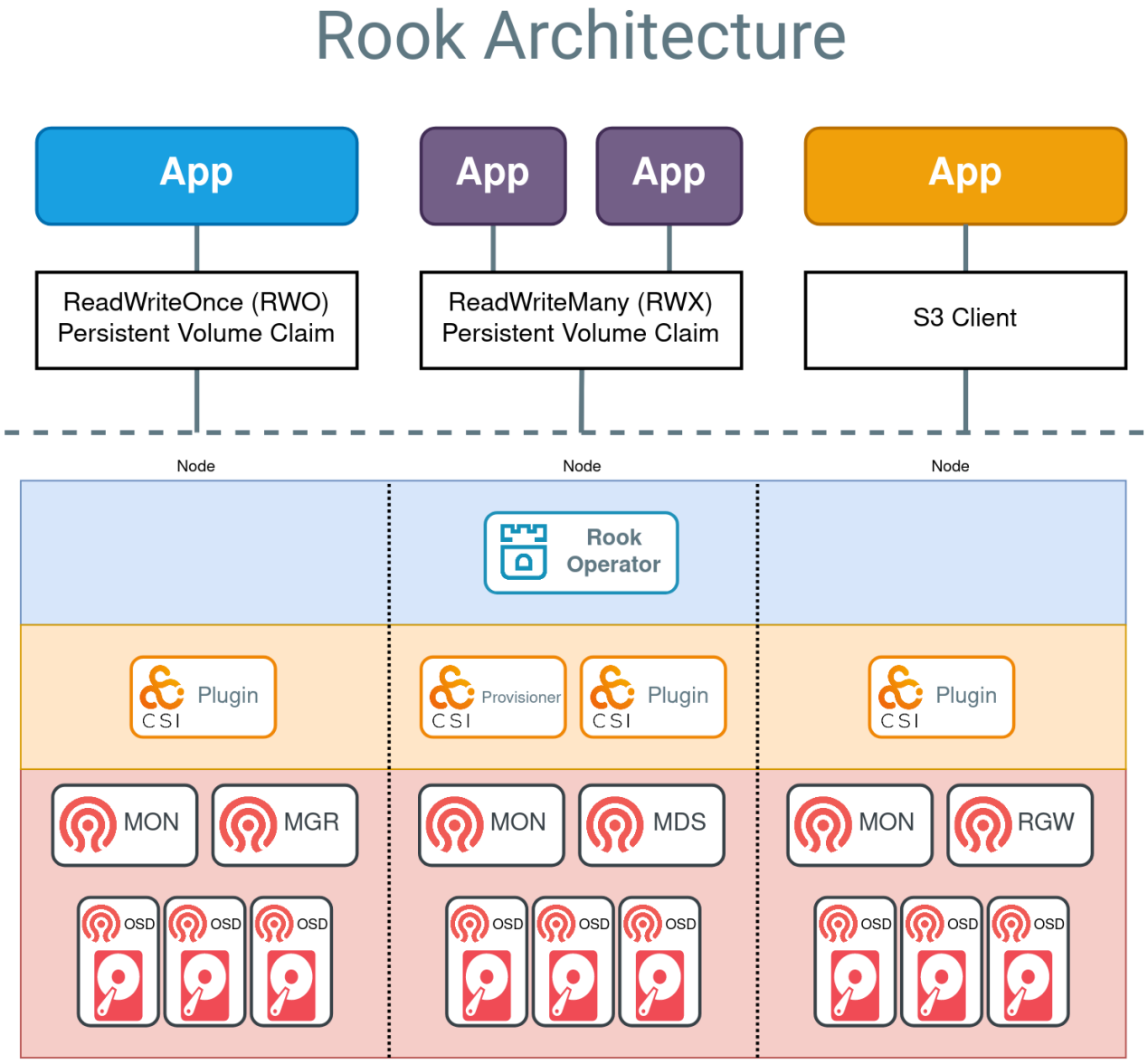
相关后续配置

- 启用已部署存储服务的网络流量加密，请参见配置传输加密。
- 部署后配置灾难恢复，请参见灾难恢复。

架构

目录

技术架构



上图展示了三种支持的存储类型的示例应用：

- 块存储由蓝色应用表示，该应用挂载了一个 ReadWriteOnce (RWO) 卷。应用可以对该 RWO 卷进行读写操作，而 Ceph 负责管理 IO。
- 共享文件系统由两个紫色应用表示，它们共享一个 ReadWriteMany (RWX) 卷。两个应用可以同时主动读写该卷。Ceph 通过 MDS 守护进程确保数据对多个写入者安全保护。
- 对象存储由一个橙色应用表示，该应用可以使用标准 S3 客户端对存储桶进行读写。

在上图虚线以下，组件分为三类：

- Rook operator（蓝色层）：operator 自动化配置 Ceph
- CSI plugins and provisioners（橙色层）：Ceph-CSI 驱动提供卷的供应和挂载

- Ceph daemons（红色层）：Ceph 守护进程运行核心存储架构。详见术语表了解各守护进程。

块存储

在上图中，创建带有 RWO 卷的应用的流程为：

- （蓝色）应用创建 PVC 来请求存储。
- PVC 定义 Ceph RBD 存储类（sc）以供应存储。
- K8s 调用 Ceph-CSI RBD provisioner 创建 Ceph RBD 镜像。
- kubelet 调用 CSI RBD 卷插件将卷挂载到应用中。
- 卷现在可用于读写。
- ReadWriteOnce 卷一次只能挂载在一个节点上。

共享文件系统

在上图中，创建带有 RWX 卷的应用的流程为：

- （紫色）应用创建 PVC 来请求存储。
- PVC 定义 CephFS 存储类（sc）以供应存储。
- K8s 调用 Ceph-CSI CephFS provisioner 创建 CephFS 子卷。
- kubelet 调用 CSI CephFS 卷插件将卷挂载到应用中。
- 卷现在可用于读写。
- ReadWriteMany 卷可以挂载在多个节点上供应用使用。

对象存储 **S3**

在上图中，创建一个可以访问 S3 存储桶的应用的流程为：

- （橙色）应用创建 BucketClaim 来请求存储桶。
- Ceph COSI Driver 创建 Ceph RGW 存储桶。
- Ceph COSI Driver 创建包含访问存储桶凭据的 secret。
- 应用从 secret 中获取凭据。
- 应用现在可以使用 S3 客户端对存储桶进行读写。

安装

创建标准型集群

标准型集群是 Ceph 存储最典型的部署方式。它将数据副本分布在不同主机的硬盘上，确保单个主机故障时，其他主机上的数据副本仍能保持服务可用性。

目录

前提条件

准备软件包

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

准备基础设施

- 存储集群至少需要 3 个节点。
- 每个节点必须至少有 1 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。
- 如果您使用的是以 Containerd 作为运行时组件的附加 Kubernetes 集群，请确保集群所有节点的 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值配置为 `1048576`，以确保分布式存储部署成功。配置说明请参考 修改 Containerd 配置信息。注意：从 v3.10.2 之前版本升级到当前版本时，如果需要在自定义 Kubernetes 集群中部署以 Containerd 作为运行时组件的 Ceph 分布式存储，也必须将集群所有节点的 `/etc/systemd/system/containerd.service` 文件中的 `LimitNOFILE` 参数值设置为 `1048576`。

注意事项

创建存储服务 和 访问存储服务 仅支持选择一种方式。

操作步骤

1 部署 Alauda Container Platform Storage Essentials

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 Storage Essentials**，点击 安装，进入 安装 **Alauda Container Platform Storage Essentials** 页面。
   配置参数：

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

2 （可选）部署 Alauda Build of LocalStorage

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

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

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

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

3 部署 Operator

1. 进入 管理员。
2. 在左侧边栏点击 存储管理 > 分布式存储。
3. 点击 立即配置。
4. 在 部署 **Operator** 向导页面，点击右下角的 部署 **Operator** 按钮。
   - 页面自动跳转到下一步，表示 Operator 部署成功。
   - 若部署失败，请参考界面提示的 清理已部署信息并重试，重新部署 Operator；若需返回分布式存储选择页面，点击 应用商店，先卸载已部署的 **rook-operator** 资源，再卸载 **rook-operator**。

4 创建集群

1. 在 创建集群 向导页面，配置相关参数，点击右下角的 创建集群 按钮。

参数	说明
集群类型	选择 标准型。
设备类类型	设备类是硬盘的分组；您可以根据存储需求自定义设备类，将不同性能的硬盘分配存储不同内容。 • 默认设备类：平台会自动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。 • 自定义设备类：自定义节点中特定组合硬盘的设备类名称，支持添加多个设备类。同一硬盘只能属于一个设备类。

参数	说明
设备类 - 名称	设备类名称。选择 自定义设备类 时，设备类名称不能使用以下名称：`hdd`、`ssd`、`nvme`。
设备类 - 存储设备	添加存储设备到设备类，可选择“选择设备”或“输入设备”方式： • 选择设备： 从可用存储设备中选择。设备可用条件如下： ◦ 设备类型为 disk 或 mpath ◦ 未检测到文件系统（fsType 为空） ◦ 容量大于 10 GiB rbd、nbd、dm-* 等设备不会显示在可选设备列表中。 注意：需先部署 Alauda Build of LocalStorage Operator。 • 输入设备： 手动输入节点下空白设备名称，如 `sda`。 注意：建议使用裸盘作为存储设备，避免使用硬盘上的单个分区，以获得更佳性能和管理效果。
快照	启用后支持创建 PVC 快照，并使用快照配置新 PVC，实现业务数据快速备份恢复。 若创建存储时未启用快照，仍可在存储集群详情页的 操作 中按需启用。 注意：使用前请确保已为当前集群部署卷快照插件。
监控告警	启用后将提供开箱即用的监控指标采集和告警能力，详见监控与告警。 注意：若此时未启用，需自行寻找存储监控和告警方案，例如在运维中心手动配置监控面板和告警策略。

2. 点击 高级配置 进行组件高级配置。

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

- 页面自动跳转到下一步，表示 Ceph 集群部署成功。
- 若创建失败，可点击清理 已创建信息或重试，自动清理资源并重新创建集群，或根据文档 分布式存储服务资源清理 手动清理资源。

5 创建存储池

1. 在 创建存储池 向导页面，配置相关参数，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、可扩展的共享文件存储服务，适用于文件共享、数据备份等。

参数	说明
	• 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。 • 对象存储：提供标准 S3 接口存储服务，适用于大数据、备份归档、云存储等。
副本数	副本数越多，冗余和数据安全性越高，但存储利用率降低。通常设置为 3，满足大多数需求。
设备类	对同类型设备或同一业务逻辑的硬盘进行统一分类，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。 • 未选择设备类时，数据将在存储池所有设备中随机存储。

对象存储还需配置以下参数：

参数	说明
地域	指定存储池所在的地域。
网关类型	默认是 S3，无法修改。
内网端口	指定集群内访问端口。
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。

- 页面自动跳转到下一步，表示存储池部署成功。
- 若部署失败，请根据界面提示检查核心组件，然后点击 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 标签页查看已创建存储池信息。

相关操作

创建 **Stretch** 型集群

详情请参考 创建 Stretch 型集群。

清理分布式存储

详情请参考 清理分布式存储。

创建 Stretch 类型集群

Stretch 集群可以跨越两个地理位置不同的站点，提供存储基础设施的灾难恢复能力。当发生灾难时，如果两个可用区中的一个完全不可用，Ceph 仍能保持可用性。

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

目录

术语

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

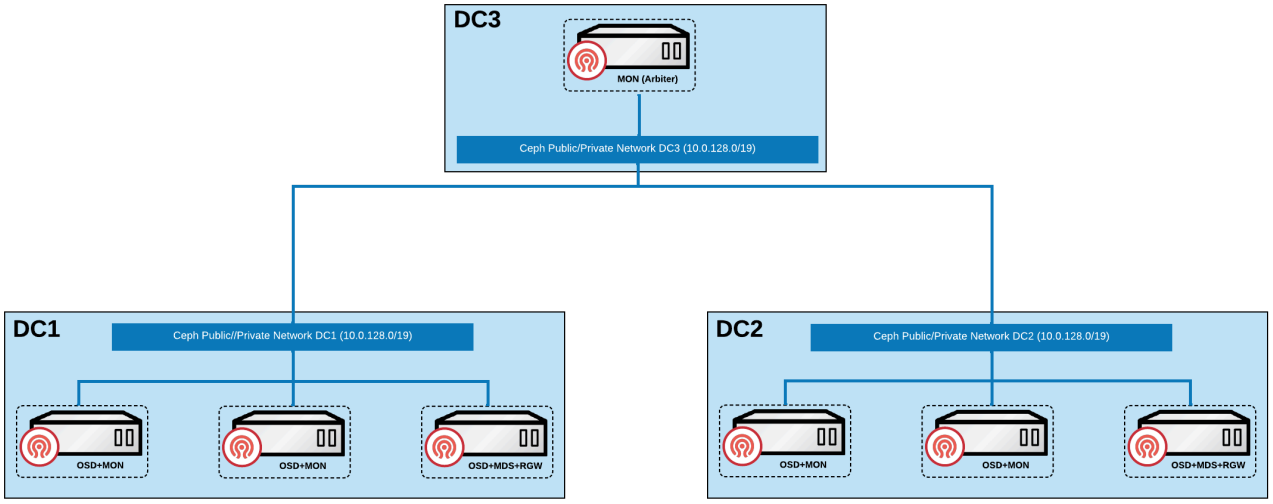
典型部署方案

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

组件说明

节点需分布在三个可用区，包括两个数据可用区和一个仲裁可用区。

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



灾难恢复说明

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

约束与限制

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

前提条件

请提前将集群中全部或部分节点划分为三个可用区，具体如下：

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

操作步骤

1 标记节点

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

2 创建存储服务

本文档仅描述与标准类型集群不同的参数，其他参数请参考创建标准类型集群。

创建集群

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

创建存储池

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

相关操作

创建标准类型集群

详情请参考创建标准类型集群。

清理分布式存储

详情请参考清理分布式存储。

核心概念

核心概念

目录

Rook Operator

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

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

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

Ceph CSI

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

Ceph 模块功能

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

操作指南

访问存储服务

前提条件

操作步骤

后续操作

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

更新组件部署配

重启存储组件

监控与告警

监控

告警

访问存储服务

访问存储服务支持两种集成方式：一是集成平台内其他业务集群的分布式存储资源，实现存储与业务隔离，便于管理和维护；二是接入外部 Ceph 存储资源进行分布式存储使用。

目录

前提条件

准备软件包

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

- 通过上传软件包机制上传该安装包。

准备存储

选择以下之一：

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

开放端口

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

获取认证信息（外部 **Ceph**）

若准备的存储为外部 Ceph 存储，需通过以下命令获取认证信息。

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

参数	获取方式
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：以下步骤以接入外部 **Ceph** 存储为例，接入分布式存储的操作类似。

1 部署 Alauda Container Platform Storage Essentials

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

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

2 访问存储

1. 在左侧导航栏点击存储管理 > 分布式存储。
2. 点击访问存储。
3. 在访问配置向导页面，选择外部 **Ceph**。

参数	说明
Snapshot	启用后支持创建 PVC 快照，并使用快照配置新的 PVC，实现业务数据的快速备份与恢复。 若访问存储时未启用快照，后续仍可在存储集群详情页的操作中根据需要启用。 注意：请确保已为当前集群部署卷快照插件后方可使用。
网络配置	• **Host Network**：本集群计算组件通过主机网络访问存储集群。 • **Container Network**：本集群计算组件通过容器网络访问存储集群。可在网络管理中创建子网，并将子网分配给 `rook-ceph` 命名空间。留空则使用默认子网。
其他参数	请填写前提条件中获取的外部 Ceph 认证参数。

4. 在创建存储类向导页面，完成配置后点击访问。

参数	说明
类型	根据上述创建的存储池类型，默认对应的存储类为： • 文件存储：CephFS 文件存储 • 块存储：CephRBD 块存储
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，绑定的持久卷仍会被保留。
项目分配	可使用该类型存储的项目。 若当前无项目需要该类型存储，可暂不分配，后续再更新。

5. 等待约 1-5 分钟，完成集成。

后续操作

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

注意：若需维护外部存储的存储池、存储设备配置等，需在存储集群的管理平台中进行操作。

存储池管理

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

目录

创建存储池

除了在分布式存储配置过程中创建的存储池外，还可以创建其他类型的存储池。

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

操作步骤

1. 进入 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。

3. 在 集群信息 标签页，向下滚动至 存储池 区域，点击 创建存储池。
4. 按照以下说明配置相关参数。

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

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

如果是对象存储类型，还可以配置以下参数：

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

参数	说明
实例数	对象存储的资源实例数量。

5. 点击 创建。

删除存储池

当某种类型的存储不再需要时，可在解除与存储类的关联后删除该存储池。

操作步骤

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

查看对象存储池地址

创建对象存储池后，可以查看该存储池的内外网访问地址。

操作步骤

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

节点特定组件部署

创建分布式存储后，您仍然可以查看和修改组件的部署位置，方便存储扩容和维护。

目录

更新组件部署配置

注意事项

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

操作步骤

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 分布式存储。

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

重启存储组件

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

操作步骤

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

添加设备/设备类

目录

添加设备类

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

注意事项

集群类型为 **Extend** 时，不支持添加设备类。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏中，点击 存储管理 > 分布式存储。

3. 点击 设备类 标签页。
4. 点击 添加设备类，并根据以下说明配置相关参数。

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

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

添加设备

将可用硬盘映射为存储设备以供使用和管理。

注意：硬盘添加为存储设备后，不支持通过界面进行更新或删除操作。

操作步骤

1. 进入 管理员。

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

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

5. 点击 添加。

硬盘状态

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

监控与告警

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

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

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

目录

监控

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

存储概览

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

性能监控

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

组件监控

监控 MON、OSD 等组件的 CPU 使用率和内存使用情况。

告警

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

配置通知

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

处理告警

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

下表说明了预设策略使用的告警级别含义，可作为您建立告警处理原则的参考。

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

故障复盘

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

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

实用指南

替换或移除设备

替换或移除存储节点

配置专用集群用于分布式存储

配置专用集群用于分布式存储

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

文件存储灾备

术语

备份配置

故障切换

块存储灾难恢复

术语

备份配置

计划迁移

灾难恢复

对象存储灾备

术语

前提条件

架构

操作步骤

故障切换

故障恢复

更新优化参数

创建 Ceph 对象存储用户

设置存储池配额

为 Ceph RGW 启用 D3N 缓存

配置传输加密

配置传输加密

替换或移除设备

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

目录

前提条件

- 所有集群组件均正常运行。

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

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

- 适用于平台版本 3.8 及以上。

约束与限制

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

操作步骤

1 检查集群状态和容量

1. 验证集群整体健康状态。

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

2. 确认待移除磁盘对应的 OSD ID 及使用情况。

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

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

2 添加替换磁盘（如有需要）

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

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

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

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

3 缩减 Rook Operator

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

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

4 标记 OSD 为 Out 并等待数据迁移

1. 若 rook-ceph-tools pod 未运行，启用它。

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

2. 进入 tools pod。

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

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

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

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

```
ceph -s
```

数据迁移未完成前请勿继续操作。 过早移除 OSD 会导致数据丢失。

5 移除 OSD

1. 编辑 CephCluster 资源，删除对应磁盘条目。

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

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

2. 删除 OSD 部署。

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

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

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

在 tools pod 内执行：

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

6 清理磁盘

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

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

# 清除分区表
sgdisk --zap-all /dev/vdb

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

7 恢复 Rook Operator

集群健康后，恢复 Rook operator。

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

8 验证集群健康

1. 确认已移除的 OSD 不再出现在集群中。

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

2. 验证集群已恢复健康状态。

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

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

参考资料

- Rook Ceph OSD Management ↗

替换或移除存储节点

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

目录

前提条件

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

约束与限制

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

操作步骤

1 检查集群状态和容量

1. 确认待移除节点上所有运行的 OSD ID 及其磁盘使用情况。

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

2. 验证集群整体健康状态。

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

3. 查看所有 OSD 的容量信息。

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

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

2 添加替换节点（如有需要）

如果剩余 OSD 空闲容量不足，需先添加替换节点再移除旧节点。

1. 进入 **Container Platform**。

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

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

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

3 调整组件部署配置

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

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

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

4 标记所有 OSD 为 Out 并等待数据迁移

1. 如果 rook-ceph-tools Pod 未运行，启用它。

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

2. 进入 tools Pod。

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

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

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

对节点上每个 OSD ID 重复此操作。

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

```
ceph -s
```

数据迁移完成前请勿继续操作。提前移除 **OSD** 会导致数据丢失。

5 移除旧节点的 OSD

1. 编辑 CephCluster 资源，删除旧节点条目。

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

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

2. 删除旧节点上每个 OSD 的部署。

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

对每个 OSD ID 重复此操作。

3. 进入 tools Pod，永久从集群中移除每个 OSD。

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

在 tools Pod 内，对每个 OSD 执行：

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

6 验证集群健康状态

1. 确认已移除的 OSD 不再出现在集群中。

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

2. 验证集群已恢复健康状态。

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

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

References

- Rook Ceph OSD Management ↗

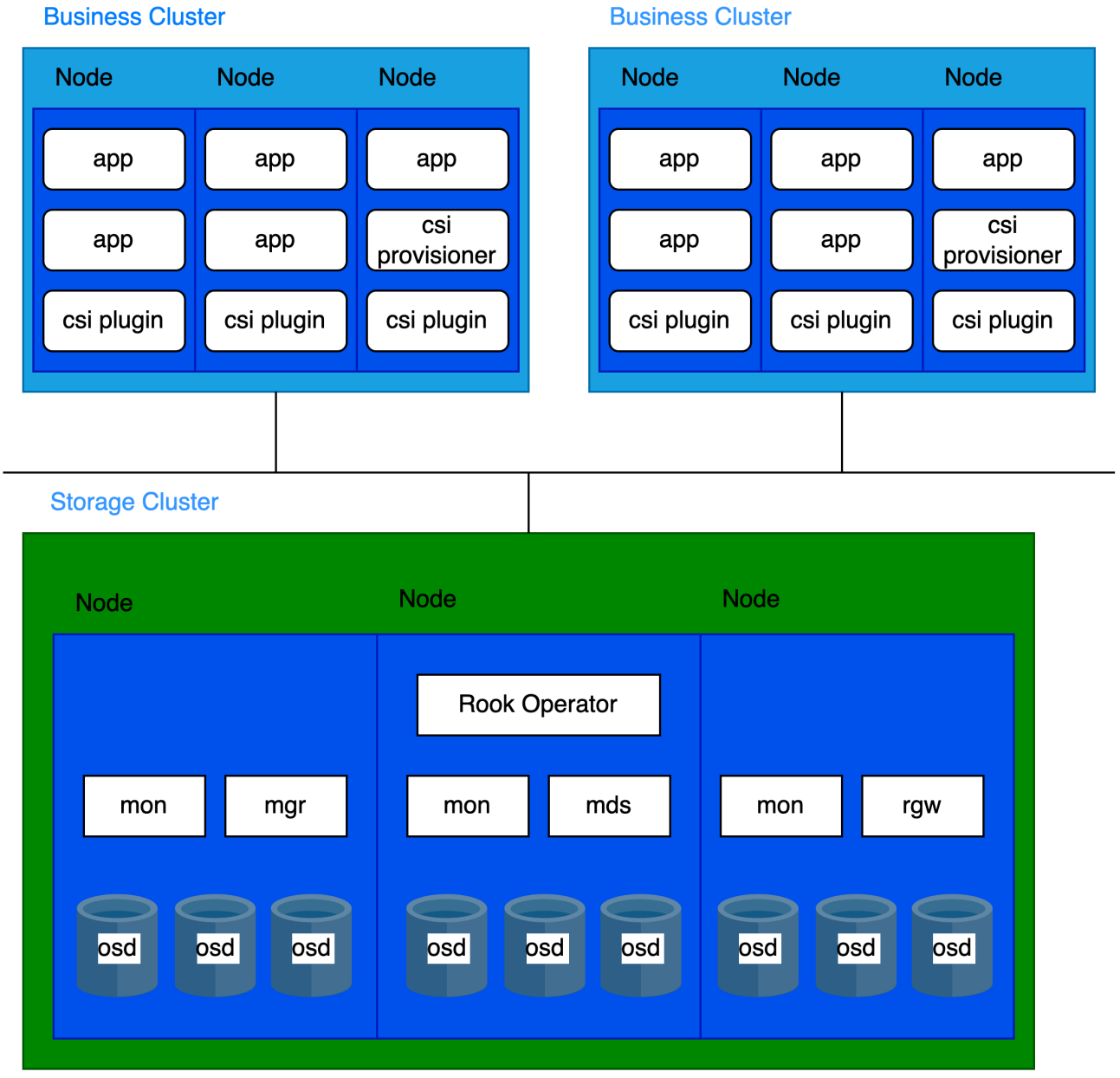
配置专用集群用于分布式存储

专用集群部署是指使用独立集群部署平台的分布式存储，平台内其他业务集群通过集成访问并使用该存储提供的服务。
为保证平台分布式存储的性能和稳定性，专用存储集群仅部署平台核心组件和分布式存储组件，避免与其他业务负载混合部署。此种分离部署方式是平台分布式存储的推荐最佳实践。

目录

架构

存储计算分离架构



基础设施要求

平台要求

支持版本为 3.18 及以后版本。

集群要求

建议使用裸金属集群作为专用存储集群。

资源要求

分布式存储部署的组件请参考核心概念。

各组件有不同的 CPU 和内存需求，推荐配置如下：

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

一个集群通常运行：

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

根据组件分布，单节点资源推荐如下：

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

存储设备要求

建议每节点部署不超过 12 个存储设备，有助于限制节点故障后的恢复时间。

存储设备类型要求

建议使用企业级 SSD，单个设备容量不超过 10TiB，且所有磁盘大小和类型保持一致。

容量规划

部署前需根据具体业务需求规划存储容量。默认分布式存储系统采用 3 副本冗余策略，因此可用容量为所有存储设备总原始容量除以 3。

以 30（N）节点（副本数 = 3）为例，可用容量示例如下：

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

容量监控与扩容

1. 主动容量规划
   始终确保可用存储容量大于消耗容量。存储空间耗尽后，恢复需人工干预，无法通过删除或迁移数据自动解决。
2. 容量告警
   集群在两个阈值触发告警：

   - **80%** 利用率（“接近满”）：主动释放空间或扩容集群。

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

请务必及时处理告警，定期监控存储使用，避免服务中断。

3. 扩容建议

- 避免：向现有节点添加存储设备。
- 推荐：通过新增存储节点进行横向扩展。
- 要求：新增节点存储设备大小、类型和数量需与现有节点保持一致。

网络要求

分布式存储必须使用 **HostNetwork**。

网络隔离

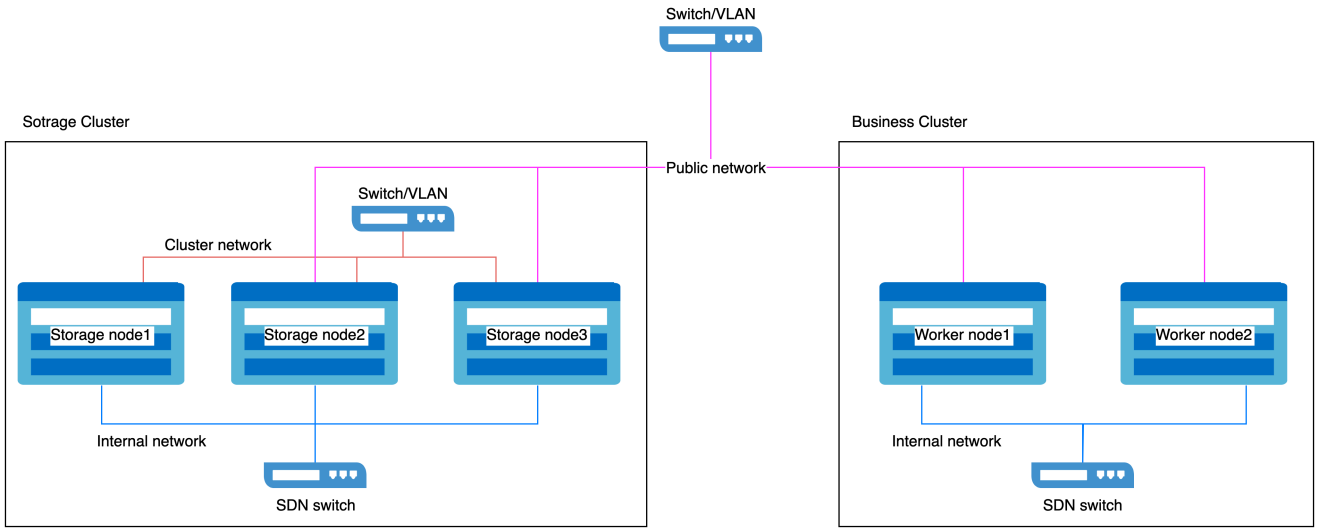
网络分为两类：

- 公网网络：用于客户端与存储组件交互（如 I/O 请求）。
- 集群网络：专用于副本间数据复制及数据均衡（如恢复）。

为保障服务质量和性能稳定：

1. 对于专用存储集群：
   每台主机预留两个网络接口：
   - 公网网络：用于客户端和组件通信。
   - 集群网络：用于内部复制和均衡流量。
2. 对于业务集群：
   每台主机预留一个网络接口访问存储公网网络。

示例网络隔离配置



网络接口速率要求

1. 存储节点

   - 公网网络和集群网络均需 10GbE 或更高速率网络接口。

2. 业务集群节点

   - 用于访问存储公网网络的网络接口需为 10GbE 或更高。

操作步骤

1 部署 **Operator**

1. 进入 管理员。
2. 在左侧边栏点击 存储管理 > 分布式存储。
3. 点击 立即创建。
4. 在 部署 **Operator** 向导页面，点击右下角 部署 **Operator** 按钮。
   - 页面自动跳转下一步表示 Operator 部署成功。
   - 若部署失败，请根据界面提示选择 清理已部署信息并重试，重新部署 Operator；若需返回分布式存储选择页，点击 应用商店，先卸载已部署的 **rook-operator** 资源，再卸载 **rook-operator**。

2 创建 **ceph** 集群

在存储集群的 控制节点 执行命令。

▶ 点击查看

参数说明：

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

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

> 提示
>
> 使用磁盘的全球唯一名称（WWN）进行稳定命名，避免依赖如 `sdb` 这类重启后可能变化的设备路径。

3 创建存储池

提供三种存储池类型，根据业务需求选择创建。

创建文件池

在存储集群的 控制节点 执行命令。

▶ 点击查看

创建块池

在存储集群的 控制节点 执行命令。

▶ 点击查看

创建对象池

在存储集群的 控制节点 执行命令。

▶ 点击查看

后续操作

当其他集群需要使用分布式存储服务时，请参考以下指南。
访问存储服务

清理分布式存储

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

目录

注意事项

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

操作步骤

1 删除 VolumeSnapshotClasses

1. 删除 VolumeSnapshotClasses。

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

2. 验证 VolumeSnapshotClasses 是否已清理。

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

当这些命令无输出时，表示清理完成。

2 删除 StorageClasses

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

3 删除存储池

此步骤应在上一步完成后执行。

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

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

响应：

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

4 删除 **ceph-cluster**

此步骤应在上一步完成后执行。

1. 更新 ceph-cluster 并启用清理策略。

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

2. 删除 ceph-cluster。

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

3. 删除执行清理的 jobs。

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

4. 验证 ceph-cluster 清理是否完成。

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

当该命令无输出时，表示清理完成。

5 删除 **rook-operator**

此步骤应在上一步完成后执行。

1. 删除 rook-operator。

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

2. 验证 rook-operator 清理是否完成。

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

当该命令无输出时，表示清理完成。

3. 验证所有 ConfigMaps 是否已清理。

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

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

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

4. 验证所有 Secrets 是否已清理。

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

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

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

5. 验证 rook-ceph 清理是否完成。

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

当该命令无输出时，表示清理完成。

6 执行清理脚本

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

清理脚本

清理脚本 clean-rook.sh 内容如下：

▶ 点击查看

注意事项

清理脚本依赖 sgdisk 命令，请确保执行清理脚本前已安装。

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

操作步骤

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

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

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

灾难恢复

文件存储灾备

CephFS Mirror 是 Ceph 文件系统的一个功能，旨在实现不同 Ceph 集群之间的异步数据复制，从而提供跨集群的灾难恢复能力。其核心功能是以主备模式同步数据，确保当主集群发生故障时，备份集群能够快速接管服务。

WARNING

- CephFS Mirror 基于快照进行增量同步，默认快照间隔为每小时一次（可配置）。主备集群之间的差异数据通常是一个快照周期内写入的数据量。
- CephFS Mirror 仅提供底层存储数据的备份，无法处理 Kubernetes 资源的备份。请结合平台的备份与恢复 功能对 PVC 和 PV 资源进行备份。

目录

术语

术语	说明
主集群	当前提供存储服务的集群。
备集群	用于备份的集群。

备份配置

前提条件

- 准备两个适合部署 Alauda Build 版本的 Rook-Ceph 的集群，分别为主集群和备集群，确保集群间网络互通。
- 两个集群使用的平台版本（v3.12 及以上）需保持一致。
- 在主集群和备集群中分别创建分布式存储服务。
- 在主集群和备集群中创建同名的文件存储池。

操作步骤

1 在备集群启用文件存储池的 **Mirror** 功能

在备集群的控制节点执行以下命令：

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
```

参数说明：

- `<fs-name>`：文件存储池名称。

2 获取 Peer Token

该令牌是建立两个集群镜像连接的关键凭证。

在备集群的控制节点执行以下命令：

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

```
# 由于涉及敏感信息，输出已截断。
eyJmc2lkIjogImMyYjAyNmMzLTA3ZGQtNDA3Z...
```

参数说明：

- `<fs-name>`：文件存储池名称。

3 在主集群创建 Peer Secret

获取备集群的 Peer Token 后，需要在主集群创建 Peer Secret。

在主集群的控制节点执行以下命令：

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
```

参数说明：

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

4 在主集群启用文件存储池的 Mirror 功能

在主集群的控制节点执行以下命令：

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
```

参数说明：

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

5 在主集群部署 Mirror Daemon

Mirror Daemon 持续监控文件存储池（启用 Mirror）的数据变化，定期创建快照并将快照差异通过网络推送到备集群。

在主集群的控制节点执行以下命令：

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: NoSchedule
      operator: Exists
  resources:
    limits:
      cpu: "500m"
      memory: "1Gi"
    requests:
      cpu: "500m"
      memory: "1Gi"
  priorityClassName: system-node-critical
EOF
```

输出

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

故障切换

当主集群发生故障时，可以直接继续使用备集群中的 CephFS。

前提条件

主集群的 Kubernetes 资源已备份并恢复到备集群，包括 PVC、PV 以及应用的工作负载。

块存储灾难恢复

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

WARNING

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

目录

术语

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

备份配置

前提条件

- 准备两个能够部署 Alauda Build 版本 Rook-Ceph 的集群：Primary 集群和 Secondary 集群。
- 两个集群必须运行相同的平台版本（v3.12 或更高）。
- 在 Primary 和 Secondary 集群中分别创建分布式存储服务。
- 在 Primary 和 Secondary 集群中创建同名的块存储池。
- 请确保以下两个镜像已上传至平台私有镜像仓库：
  - `quay.io/csiaddons/k8s-controller:v0.12.0` -> `<registry>/csiaddons/k8s-controller:v0.12.0`

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

网络前提条件

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

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

操作步骤

引导对等节点（`Primary <-> Secondary`）

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

   命令

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

   输出

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

   参数说明：

- `<block-pool-name>`：块存储池名称。

2. 获取对等令牌

   该令牌是建立集群镜像连接的关键凭证。

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

   命令

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

   输出

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

3. 在对等集群中创建对等令牌 **Secret**

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

   命令

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

   输出

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

   参数说明：

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

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

命令

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

输出

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

参数说明：

- `<block-pool-name>`：块存储池名称。

5. 部署 **Mirror** 守护进程
   该守护进程负责监控和管理 RBD 镜像同步进程，包括数据同步和错误处理。
   在 Primary 和 Secondary 集群的 Control 节点上执行以下命令：

命令

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

输出

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

6. 验证镜像状态

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

命令

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

输出

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

参数说明：

- `<block-pool-name>`：块存储池名称。

配置卷复制环境

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

1. 部署 **CsiAddons Controller**

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

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

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

   参数说明：

   - `<registry>`：平台镜像仓库地址。

2. 启用 **CsiAddons sidecar**

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

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

等待所有 csi pod 成功重启

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

3. 创建 **VolumeReplicationClass**

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

   命令

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

输出

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

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

为 PVC 启用镜像功能

在 Primary 集群的 Control 节点上执行以下命令：

命令

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

输出

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

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

注意

启用后，Secondary 集群中的 RBD 镜像将变为只读。

计划迁移

使用场景：数据中心维护、技术更新、灾难规避等。

迁移切换

迁移切换是指将生产环境切换到备份设施（通常是恢复站点）或反向切换的过程。

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

前提条件

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

操作步骤

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

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

示例

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

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

3 `<pvc-name>`：需要启用镜像的 PVC 名称。

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

灾难恢复

使用场景：自然灾害、电力故障、系统故障及崩溃等。

故障切换（突发停机）

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

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

前提条件

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

操作步骤

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

示例

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

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

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

故障恢复后的回切

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

前提条件

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

操作步骤

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

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

对象存储灾备

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

目录

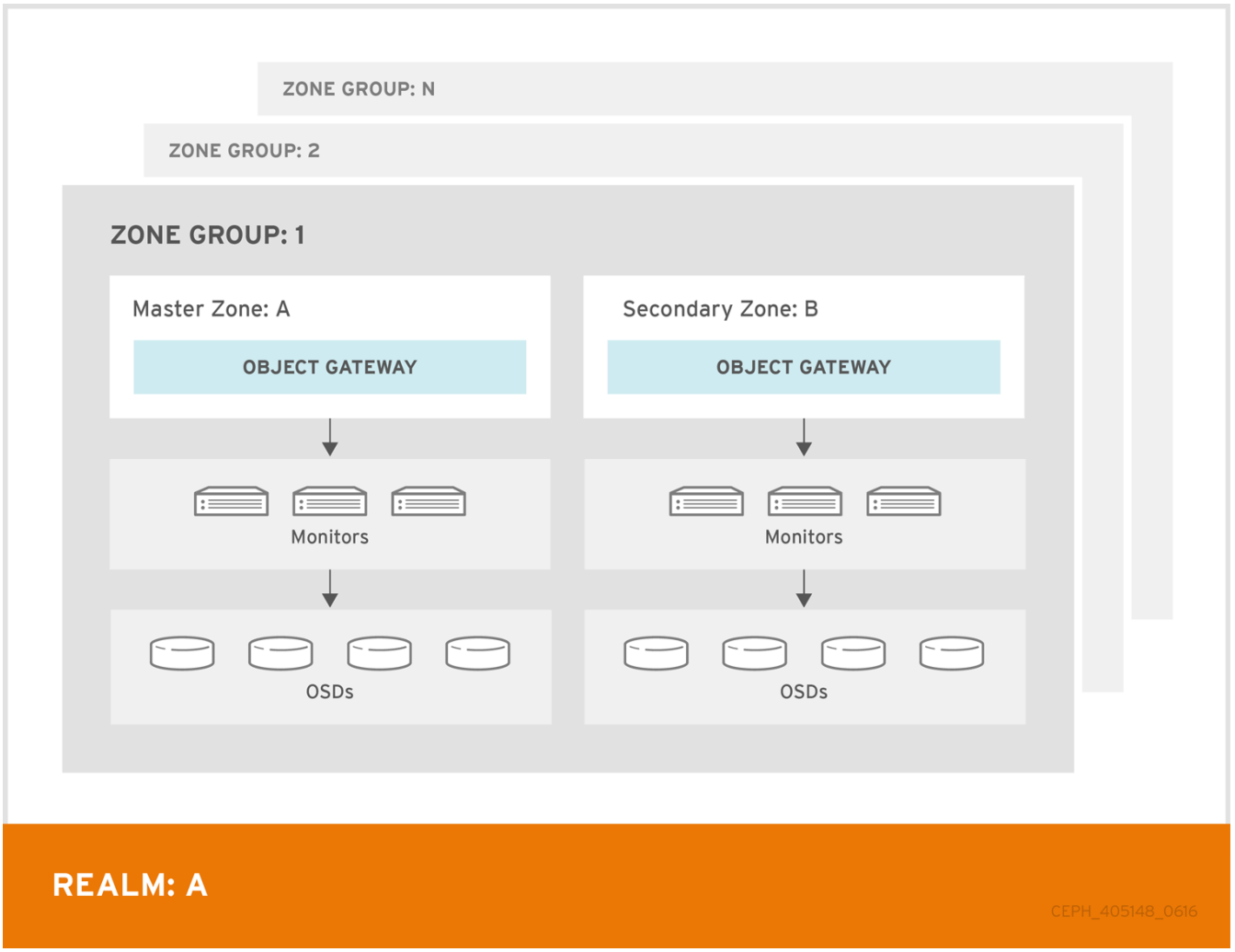
术语

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

前提条件

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

架构



操作步骤

本指南提供同一 ZoneGroup 中两个 Zone 之间的同步方案。

1 在 **Primary** 集群创建对象存储

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

在 Primary 集群的 Control 节点执行以下命令：

1. 设置参数

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

参数说明：

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

2. 创建对象存储

命令

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

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

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

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

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

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

输出

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

2 配置 **Primary Zone** 的外部访问

1. 获取 ObjectStore 的 UID

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

2. 创建外部访问 Service

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

3. 向 CephObjectZone 添加外部端点

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

3 获取 `access-key` 和 `secret-key`

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

4 创建 Secondary Zone 并配置 Realm 同步

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

在 Secondary 集群的 Control 节点执行以下命令：

1. 设置参数

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

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

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

参数说明：

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

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

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

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

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

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

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

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

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

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

5 配置 **Secondary Zone** 的外部访问

1. 获取 Secondary Gateway 的 UID

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

2. 创建外部访问 Service

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

3. 向 Secondary CephObjectZone 添加外部端点

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

6 检查 Ceph 对象存储同步状态

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

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

radosgw-admin sync status
```

输出示例

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

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

故障切换

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

操作步骤

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

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

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

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

参数

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

故障恢复

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

操作步骤

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

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

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

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

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

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

参数

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

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

更新优化参数

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

目录

操作步骤

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

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

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

创建 Ceph 对象存储用户

我们允许通过自定义资源定义（CRDs）创建和自定义对象存储用户。

目录

前提条件

- 对象存储池已创建

操作步骤

1 创建用户

在集群的控制节点上执行命令。

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

参数说明

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

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

2 允许在其他命名空间创建用户

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

在其自身命名空间中创建对象存储凭据的应用程序非常有用。

在集群的控制节点上执行命令。

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

3 获取用户信息

在集群的控制节点上执行命令。

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

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

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

设置存储池配额

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

目录

前提条件

- 已安装 Ceph 集群
- 已创建 Ceph 存储池

为文件存储池设置池配额

在集群的控制节点上执行命令。

Command

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

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

Example

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

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

参数说明

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

为块存储池设置池配额

在集群的控制节点上执行命令。

Command

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

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

Example

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

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

参数说明

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

为对象存储池设置池配额

在集群的控制节点上执行命令。

Command

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

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

Example

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

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

参数说明

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

通过 **Ceph** 终端验证池配额

```
ceph osd pool ls detail | grep max_bytes

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

为 Ceph RGW 启用 D3N 缓存

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

目录

背景

D3N 的核心思想简单而有效：
RGW 不再反复从 Ceph 后端存储获取对象数据，而是直接从位于 RGW 节点上快速磁盘（NVMe/SSD）上的 本地持久缓存 提供热点数据。

该设计特别适用于：

- 大对象下载
- 热点对象的重复读取
- 对带宽敏感的工作负载

前提条件

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

操作概览

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

准备本地文件系统作为缓存

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

在 **CephObjectStore** 中启用 **D3N** 缓存

编辑 `CephObjectStore` 资源：

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

在 `gateway` 下添加如下配置：

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

参数说明

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

验证 **D3N** 配置

通过 **Ceph** 工具检查 **RGW** 配置

在 `rook-ceph-tools` pod 中运行：

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

示例输出：

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

验证 **RGW** 日志

RGW pod 重启后，应出现 D3N 初始化日志：

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

示例输出：

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

验证缓存行为

通过 RGW 下载对象时，缓存文件会出现在宿主机的缓存目录中：

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

示例输出：

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

这些文件的存在证明 RGW 正在从本地 D3N 缓存提供数据。

配置传输加密

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

目录

概述

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

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

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

限制与前提条件

启用此功能前，请注意以下限制：

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

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

为新集群启用传输加密

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

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

集群创建完成后，验证：

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

部署后启用传输加密

如果集群已运行，仅修改加密开关是风险最低的方式。

步骤 1. 确认节点内核版本

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

```
uname -r
```

如果部分工作节点不满足要求，升级这些节点之前请勿在生产集群启用传输加密。

步骤 2. 在 CephCluster 中启用加密

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

步骤 3. 等待配置生效

配置更新后：

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

禁用传输加密

在现有集群中禁用加密

仅禁用传输加密，同时保持 `msgr2` 可用：

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

验证

启用功能后，从 Kubernetes 和 Ceph 两方面验证集群状态。

检查 **CephCluster** 设置

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

确认输出包含：

```
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

MinIO 对象存储

介绍

介绍

安装

安装

前提条件

操作步骤

相关信息

架构

架构

核心组件：

部署架构：

多池扩展：

结论：

核心概念

核心概念

操作指南

添加存储池

注意事项

操作步骤

Monitoring & Alerts

Monitoring

Alerts

实用指南

数据灾难恢复

适用场景

术语

前提条件

操作步骤

相关操作

MinIO 到 Rook Ceph RGW 数据迁移指南

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

介绍

Alauda Container Platform (ACP) Object Storage with MinIO 是一款基于 Apache License v2.0 许可的对象存储服务。它兼容 Amazon S3 云存储服务接口，非常适合存储大量非结构化数据，如图片、视频、日志文件、备份数据以及容器/虚拟机镜像。单个对象文件的大小范围从几 KB 到最大 5T。

主要优势如下：

- 简洁性：极简主义是 MinIO 的设计指导原则，支持开箱即用的功能。简洁性降低了出错概率，提高了正常运行时间和可靠性，同时也提升了性能。
- 高性能：MinIO 是全球领先的对象存储解决方案。在标准硬件上，读写速度可达 183 GB/sec 和 171 GB/sec。
- 可扩展性：可以建立多个小型到中型、易于管理的集群，支持将多个集群聚合成跨数据中心的超大资源池，而非直接采用大型集中管理的分布式集群。
- 云原生：符合所有原生云计算架构和构建流程，融合了云计算领域的最新技术和理念，使对象存储对 Kubernetes 更加友好。

安装

Alauda Container Platform (ACP) Object Storage with MinIO 是基于 Apache License v2.0 开源协议的对象存储服务。它兼容 Amazon S3 云存储服务接口，非常适合存储大量非结构化数据，如图片、视频、日志文件、备份数据以及容器/虚拟机镜像。一个对象文件的大小可以是任意的，从几千字节到最大 5 TB 不等。

目录

前提条件

- MinIO 构建于底层存储之上，请确保当前集群已创建存储类，推荐使用 TopoLVM。
- 下载对应您平台架构的**Alauda Container Platform Storage Essentials**安装包。

- 使用 Upload Packages 机制上传该安装包。
- 下载对应您平台架构的**Alauda Container Platform (ACP) Object Storage with MinIO**安装包。
- 使用 Upload Packages 机制上传该安装包。

操作步骤

1 部署 Alauda Container Platform Storage Essentials

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

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

2 部署 Operator

1. 在左侧导航栏点击 **Storage** > **Object Storage**。
2. 点击 **Configure Now**。
3. 在 **Deploy MinIO Operator** 向导页面，点击右下角的 **Deploy Operator**。

- 页面自动跳转到下一步表示 Operator 部署成功。
- 若部署失败，请根据界面提示执行 **Clean Up Deployed Information and Retry**，然后重新部署 Operator。

3 创建集群

1. 在 **Create Cluster** 向导页面，配置基本信息。

参数	说明
Access Key	访问密钥 ID。与私有访问密钥关联的唯一标识符，用于与访问密钥 ID 一起加密和签名请求。
Secret Key	私有访问密钥，与访问密钥 ID 配合使用，用于加密和签名请求，识别发送者并防止请求篡改。

2. 在 **Resource Configuration** 区域，根据以下说明配置规格。

参数	说明
Small scale	适用于处理最多 100,000 个对象，支持测试环境或数据备份场景中不超过 50 个并发访问。CPU 资源请求和限制默认设置为 2 核，内存资源请求和限制默认设置为 4 Gi。
Medium scale	面向企业级应用，需存储 1,000,000 个对象，支持最多 200 个并发请求。CPU 资源请求和限制默认设置为 4 核，内存资源请求和限制默认设置为 8 Gi。
Large scale	面向拥有 10,000,000 个对象存储需求和支持最多 500 个并发请求的集团用户，适合高负载场景。CPU 资源请求和限制默认设置为 8 核，内存资源请求和限制默认设置为 16 Gi。
Custom	为专业用户提供灵活配置选项，满足特定需求，确保服务规模和性能要求精准匹配。注意：配置自定义规格时，需确保： • CPU 资源请求大于 100 m。 • 内存资源请求大于等于 2 Gi。

参数	说明
	• CPU 和内存资源限制大于等于资源请求。

3. 在 **Storage Pool** 区域，根据以下说明配置相关信息。

参数	说明
Instance Number	增加 MinIO 集群实例数量可显著提升系统性能和可靠性，确保数据高可用。但实例过多可能导致： • 资源消耗增加。 • 若单节点承载多个实例，节点故障可能导致多个实例同时离线，降低集群整体可靠性。 注意： • 最小可输入实例数为 4。 • 实例数大于 16 时，输入值必须为 8 的倍数。 • 新增存储池时，实例数不得少于第一个存储池的实例数。
Single Storage Volume	单个存储卷 PVC 容量。每个存储服务管理一个存储卷。输入单个存储卷容量后，平台会自动计算存储池容量及其他信息，可在 **Storage Pool Overview** 中查看。
Underlying Storage	MinIO 集群使用的底层存储。请选择当前集群已创建的存储类，推荐使用 TopoLVM。
Storage Nodes	选择 MinIO 集群所需的存储节点，建议使用 4-16 个存储节点。平台会为每个选中的存储节点部署一个存储服务。
Storage Pool Overview	具体参数及计算公式，请参考 Storage Pool Overview。

4. 在 **Access Configuration** 区域，根据以下说明配置相关信息。

参数	说明
External Access	启用时支持跨集群访问 MinIO；禁用时仅支持集群内访问。
Protocol	支持 HTTP 和 HTTPS；选择 HTTPS 时需填写 **Domain** 并导入域名证书的 **Public Key** 和 **Private Key**。 注意： • 访问协议为 HTTP 时，集群内 pod 可通过获取的 IP 或域名直接访问 MinIO，无需配置 IP 与域名映射；集群内节点可通过获取的 IP 直接访问 MinIO，若需域名访问，则需手动配置 IP 与域名映射；外部访问可直接通过获取的 IP 访问。 • 访问协议为 HTTPS 时，集群内外均无法通过 IP 访问 MinIO，需手动配置获取的 IP 与集群创建时填写的域名映射，才能通过域名正常访问。
Access Method	• **NodePort**：在每个计算节点主机上开放固定端口，将服务暴露到外部。配置域名访问时，建议使用 VIP 进行域名解析以保证高可用。 • **LoadBalancer**：使用负载均衡器转发流量到后端服务。使用前请确保当前集群已部署 MetalLB 插件且外部地址池有可用 IP。

5. 点击右下角 **Create Cluster**。

   - 页面自动跳转到 **Cluster Details** 表示集群创建成功。
   - 若集群仍处于创建中，可点击 **Cancel**。取消后会清理已部署的集群信息，可返回集群创建页面重新创建集群。

4 创建 **Bucket**

登录集群控制节点，使用命令创建 bucket。

1. 在集群详情页面，点击 **Access Method** 标签查看 MinIO 访问地址，或使用以下命令查询。

```
kubectl get svc -n <tenant ns> minio | grep -w minio | awk '{print $3}'
```

注意：

- 将 `tenant ns` 替换为实际命名空间 `minio-system`。
- 示例：`kubectl get svc -n minio-system minio | grep -w minio | awk '{print $3}'`

2. 获取 mc 命令。

```
wget https://dl.min.io/client/mc/release/linux-amd64/mc -O /bin/mc && chmod a+x /bin/mc
```

3. 配置 MinIO 集群别名。

- IPv4：

```
mc --insecure alias set <minio cluster alias> http://<minio endpoint>:<port> <accessKey> <secretKey>
```

- IPv6：

```
mc --insecure alias set <minio cluster alias> http://[<minio endpoint>]:<port> <accessKey> <secretKey>
```

- 域名：

```
mc --insecure alias set <minio cluster alias> http://<domain name>:<port> <accessKey> <secretKey>
mc --insecure alias set <minio cluster alias> https://<domain name>:<port> <accessKey> <secretKey>
```

注意：

- `<minio endpoint>` 填写步骤 1 中获取的 IP 地址。
- `<accessKey>` 和 `<secretKey>` 填写集群创建时配置的 **Access Key** 和 **Secret Key**。

- 配置示例：
  - IPv4: `mc --insecure alias set myminio http://12.4.121.250:80 07Apples@ 07Apples@`
  - IPv6: `mc --insecure alias set myminio http://[2004::192:168:143:117]:80 07Apples@ 07Apples@`
  - 域名: `mc --insecure alias set myminio http://test.minio.alauda:80 07Apples@ 07Apples@` 或 `mc --insecure alias set myminio https://test.minio.alauda:443 07Apples@ 07Apples@`

4. 创建 bucket。

```
mc --insecure mb <minio cluster alias>/<bucket name>
```

5 上传/下载文件

创建 bucket 后，可使用命令行上传文件到 bucket，或从 bucket 下载已有文件。

1. 创建测试上传文件。若上传已有文件，此步骤可跳过。

```
touch <file name>
```

2. 上传文件到 bucket。

```
mc --insecure cp <file name> <minio cluster alias>/<bucket name>
```

3. 查看 bucket 中的文件，确认上传成功。

```
mc --insecure ls <minio cluster alias>/<bucket name>
```

4. 删除已上传文件。

```
mc --insecure rm <minio cluster alias>/<bucket name>/<file name>
```

相关信息

冗余因子映射表

注意：新增存储池时，冗余因子需根据第一个存储池的实例数计算。

实例数	冗余因子
4 - 5	2
6 - 7	3
>= 8	4

Storage Pool Overview

Storage Pool Overview 参数	计算公式
可用容量	当实例数 ≤ 16 时，可用容量 = 单个存储卷容量 × (实例数 - 冗余因子)。
当实例数 > 16 时，可用容量 = 单个存储卷容量 × (实例数 - 4 × (实例数 + 15) / 16)。其中“4 × (实例数 + 15) / 16”的结果向下取整。	
总容量	总容量 = 实例数 × 单个存储卷容量
可容忍故障存储服务数量	当实例数 > 2 × 冗余因子时，可容忍故障存储服务数量 = 冗余因子。
当实例数 = 2 × 冗余因子时，可容忍故障存储服务数量 = 冗余因子 - 1	

架构

Alauda Container Platform (ACP) 结合 MinIO 的对象存储是一种高性能、分布式的对象存储系统，专为云原生环境设计。它利用纠删码、分布式存储池和高可用机制，确保 Kubernetes 中的数据持久性和可扩展性。

目录

核心组件：

- **MinIO Operator**：管理 MinIO 集群的部署和升级。
- **MinIO Peer**：配置和管理 MinIO 的站点复制功能。
- **MinIO Pool**：MinIO 的核心组件，负责处理对象存储请求。每个 pool 对应一个 StatefulSet，提供存储资源。

部署架构：

在 Kubernetes 中部署 MinIO 需要定义一个 MinIO tenant，指定服务器实例（pod）数量及每个实例的卷（驱动器）数量。每个 MinIO 服务器通过 StatefulSet 管理，确保稳定的身份和持久存

储。MinIO 将所有驱动器聚合成一个或多个纠删集，并应用纠删码以实现容错。

多池扩展：

MinIO 集群可以通过添加额外的服务器池来扩展，每个池拥有自己的纠删集。虽然这提供了更大的存储容量，但也增加了集群维护的复杂性，并降低了整体集群的可靠性。任何服务器池的故障都可能导致整个 MinIO 集群不可用，即使其他池仍在运行。

结论：

MinIO 是一个高度可扩展的云原生对象存储解决方案，兼顾性能与可靠性。在设计 MinIO 集群架构时，必须仔细规划存储池、配置纠删码设置，并实施高可用策略，以确保 Kubernetes 环境中的数据完整性和运行稳定性。

核心概念

核心概念

- **Erasure Coding (EC)**：MinIO 采用 Reed-Solomon 擦除编码将对象拆分为数据和校验分片，分布在多个驱动器上以确保容错。例如，在 16 驱动器的配置中，数据可以拆分为 12 个数据分片和 4 个校验分片，使系统即使在最多 4 个驱动器故障的情况下也能重建数据。
- **Server Pools & Erasure Sets**：MinIO Server Pools 是存储资源的逻辑分组，每个池由多个节点组成，共享存储和计算能力。在一个池内，驱动器会自动组织成一个或多个 **Erasure Sets**。
  - 数据分布：当存储对象时，它会被拆分为数据和校验分片，并分布在擦除集内的不同驱动器上。
  - 冗余模型：擦除集构成数据冗余的基本单元，基于配置的数据与校验分片比例确保系统的弹性。
  - 可扩展性：单个 MinIO 存储池可以包含多个擦除集，新数据总是写入可用容量最大的擦除集。

操作指南

添加存储池

存储池是指用于存储数据的逻辑分区。在同一个存储集群中，可以同时使用不同类型的底层存储，以满足多样化的业务需求。

除了在配置对象存储时创建的存储池外，您还可以添加额外的存储池。

目录

注意事项

添加存储池会导致 MinIO 服务短暂中断，但之后会自动恢复到正常状态。

操作步骤

1. 进入 管理员。
2. 点击左侧导航栏中的 存储管理 > 对象存储。
3. 在 集群信息 标签页下，向下滚动至 存储池 区域，点击 添加存储池。
4. 按照以下说明配置相关参数。

参数	说明
底层存储	MinIO 集群使用的底层存储。请选择当前集群中已创建的存储类，推荐使用 TopoLVM。
存储节点	选择 MinIO 集群所需的存储节点，建议使用 4-16 个存储节点；平台会为每个选中的存储节点部署 1 个存储服务。 注意：当使用 3 个存储节点时，为保证可靠性，每个节点会部署 2 个存储服务。
单个存储卷容量	单个存储卷 PVC 的容量。每个存储服务管理 1 个存储卷，输入单个存储卷容量后，平台会自动计算存储池容量及其他信息，可在 存储池概览 中查看。

5. 点击 确认。

Monitoring & Alerts

对象存储系统内置了监控和告警功能，涵盖存储集群、服务健康状况和资源利用率。同时支持可配置的通知策略，确保运维团队及时获知系统状态。实时监控数据有助于性能调优和运维决策，自动告警则保障存储系统的稳定性和可靠性。

目录

Monitoring

平台默认采集存储集群和服务状态的关键指标。您可以在 **Storage Management** > **Object Storage** > **Monitoring** 中查看实时监控数据。

Storage Overview

本节提供存储系统健康状况、服务状态及原始容量利用率的整体视图。如果存储状态异常，告警详情将指明根本原因，帮助您高效诊断和解决问题。

Cluster Monitoring

跟踪存储集群的原始容量使用情况和 I/O 性能趋势，有助于识别存储瓶颈、优化资源分配，确保数据操作顺畅。

Object Monitoring

监控访问模式，包括总请求数和失败请求数。这些洞察有助于分析存储负载，发现可能导致服务中断或安全风险的异常情况。

Alerts

平台内置了预配置的告警策略，用于检测异常并在达到预设阈值时触发通知。这些规则覆盖关键领域，如组件健康、容量使用和用户数据完整性。

Configuring Notifications

为确保及时响应，请在 **Operations Center** 中配置通知策略。告警可通过邮件、短信或其他渠道发送，通知相关人员。根据组织的事件响应流程，细化设置以匹配实际需求。

Handling Alerts

- **Cluster in "Alert" state**：已触发警告，系统稳定性可能受影响。请查看 **Live Alerts** 部分获取详情，定位根因并采取纠正措施。
- **Cluster in "Failure" state**：存储集群已无法正常运行。需立即介入以恢复服务可用性。

平台将告警分为不同严重级别，帮助团队优先处理事件：

Severity	Description
Critical	影响业务运营或导致数据丢失的系统故障，需立即处理。

Severity	Description
Major	可能导致功能中断的已知问题，可能影响业务流程。
Warning	潜在风险，若不处理可能影响性能或可用性。

Post-Incident Analysis

Alert History 记录所有历史事件，为事后分析和系统改进提供宝贵数据。回顾告警时，请考虑：

1. 事件发生时的具体症状是什么？
2. 是否有某些告警反复出现？是否可以采取主动措施防止复发？
3. 是否存在某个时间段告警激增？是由运维问题还是外部因素引起？是否需要调整响应策略？

通过持续分析告警模式和优化监控策略，团队可提升系统韧性，减少停机时间，确保存储系统平稳运行。

实用指南

数据灾难恢复

MinIO 到 Rook Ceph RGW 数据迁移指南

数据灾难恢复

MinIO 支持通过远程数据备份或主动-主动部署建立灾难恢复中心，以确保在灾难发生时原始数据不丢失或损坏，从而保障数据的安全性和可靠性。

目录

适用场景

- 热备份：在同一城市或不同地点有两个数据中心，一个为主，一个为备。数据实时从主集群复制到备集群，确保数据一致性。当主集群发生灾难时，业务流量可以无缝切换到备集群，保证业务连续性。
- 城市级主动-主动：在城市级主动-主动（多集群）架构中，有两个位于不同集群的数据中心。两个数据中心均处于活动状态，可以同时接收业务流量。当其中一个数据中心遇到灾难时，业务可以在另一个数据中心不间断运行。

术语

- 主集群：指当前处于活动状态并处理业务请求的集群。它是数据的源头或操作的发起方。在主集群中创建、修改或更新数据，业务流量首先发送到该集群进行处理。
- 目标集群：指接收数据复制、迁移或故障切换的集群。通常处于备份或待命状态，等待接收来自主集群的数据或接管业务流量。当主集群发生故障或需要切换时，目标集群将接收来自主集群的数据副本或接管业务流量，以确保业务连续性。在主动-主动场景中，两个集群可以互为目标集群。

前提条件

- 主集群和目标集群均需开启外网访问。具体配置方法请参见 Create Object Storage。
- 主集群必须使用 **LoadBalancer** 访问方式，目标集群建议支持 负载均衡器 功能。
- 主集群和目标集群必须使用相同的访问协议，即均使用 HTTP 或均使用 HTTPS。
- 使用 HTTPS 协议时，主集群和目标集群均需配置自身及对方的 DNS 解析。
- 使用 HTTPS 协议时，建议主集群和目标集群均使用 CA 签发的证书以确保通信安全可信；若使用自签证书，双方必须导入并信任对方的自签证书，才能成功建立安全的 HTTPS 连接。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏点击 存储管理 > 对象存储。
3. 在 数据灾难恢复 标签页，点击 添加目标集群。
4. 按照以下说明配置目标集群相关参数。

参数	说明
访问地址	目标集群的外部访问地址，需以 http:// 或 https:// 开头。
Access Key	目标集群的 Access Key ID。与私有访问密钥关联的唯一标识符；用于配合私有访问密钥对请求进行加密。

参数	说明
Secret Key	用于配合 Access Key ID 对请求进行加密、识别发送方并防止请求被篡改的私有访问密钥。

5. 点击 添加。

- 添加成功后，可查看目标集群状态及集群间同步状态。

参数	说明
集群状态	目标集群的状态，包括 **Healthy**（健康）、**Abnormal**（异常）或 **Unknown**（未知）。
Buckets	待同步和已同步的桶数量。 • 在热备份场景中，待同步指主集群需要同步到目标集群的桶数量。 • 在城市级主动-主动场景中，待同步指主集群与目标集群之间需要同步的桶总数。
Objects	桶中同步失败的对象数量。 注意：该数字仅供参考，因 MinIO 在同步过程中会同步相关文件配置。
网络流量速率	主集群的网络进出速率。 • 在热备份场景中，网络入口速率始终为 0。 • 在城市级主动-主动场景中，入口和出口速率均有数据。

- 若添加目标集群失败，可点击 重新添加 清除集群信息并返回添加目标集群页面，重新添加目标集群。

相关操作

当不再需要灾难恢复时，可以点击 移除目标集群。移除目标集群不会删除已同步的数据；若有数据正在同步，将会被中断。

MinIO 到 Rook Ceph RGW 数据迁移指南

目录

1. 概述与架构

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

1.1 架构说明

- 安全镜像复用：该方案通过调度 Kubernetes Job，复用安全加固的 `build-harbor.alauda.cn/acp/volsync` Operator 镜像，覆盖默认入口，直接调用镜像内置的 Rclone 二进制文件，执行标准的 **S3-to-S3** API 级对象同步。

1.2 同步策略

采用三阶段策略：**“全量 -> 增量 -> 切换”**：

1. 初始同步（全量同步）：保持服务在线，迁移历史数据。
2. 增量同步：保持服务在线，多次运行以追赶新增或修改的数据。
3. 切换同步：停止写入，强制严格一致性校验，完成最终切换。

---

2. 前提条件与准备工作

2.1 验证 Alauda VolSync 安装及提取版本（关键前提）

执行本方案前，确保在 `volsync-system` 命名空间已安装 **Alauda Build of VolSync** Operator。迁移 Job 镜像版本必须严格匹配当前运行的 Operator 版本。

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

获取版本方法：
登录集群管理控制台，进入 **Marketplace / OperatorHub**，打开 **Installed Operators**，找到 VolSync Operator，记录显示的版本号（例如 `v0.8.0` 或 `v0.9.0`）。将此值保存为 `<OPERATOR_VERSION>`，后续配置时使用。

2.2 收集 MinIO 源信息

- **Endpoint**：MinIO 内部 Service 地址（例如：`http://minio.tenant-ns.svc:9000`）。
- 凭证：具备所有桶的 `Read` 和 `List` 权限的 Access Key / Secret Key。

2.3 创建 Ceph RGW 用户（目标端）

在 Rook Ceph 集群中应用以下 YAML，创建专用迁移用户并授予桶创建权限。

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

为最小权限原则，该迁移账号仅保留桶级权限，不授予 `user` 管理能力。

提取并解码 Ceph 凭证以备后用：

```bash
# 获取 Access Key
kubectl -n rook-ceph get secret rook-ceph-object-user-object-store-volsyn
c-migration-user -o jsonpath='{.data.AccessKey}' | base64 -d
# 获取 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（清单）

建议在目标端命名空间（Ceph RGW）或专用运维命名空间部署迁移任务。

应用清单前，先创建两个清单共用的命名空间：

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

> 部署位置建议（重要）
> 迁移过程中，Rclone 本地读取数据处理后上传至目标 S3 集群，迁移任务所在集群的网络位置直接影响带宽和完成时间。建议将 VolSync Operator/迁移 Job 部署在 **MinIO** 或 **Ceph** 集群，以减少跨集群网络开销，提高传输稳定性。

3.1 创建 Rclone 配置 Secret

将源和目标 S3 凭证写入 Secret。重要：不要在 **endpoint** 或密钥值周围添加引号。

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

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

3.2 定义迁移 Job

部署一个调用安全加固的 Alauda VolSync 镜像执行 S3 数据同步的 Job。

> 关于 `remote:bucket[/prefix]` 与 `remote:`（重要）
>
> - 对于 S3 后端，Rclone 常用模式是 `remote:bucket` 或 `remote:bucket/prefix`，这是定义作用域最清晰且安全的方式。
> - 本文档故意保持 `source-minio:` -> `dest-ceph:`，以支持全实例级迁移（源端所有可见桶）。
> - 风险提示：`remote:` 范围更广，若凭证权限过大，可能迁移到非预期桶。务必先在测试环境验证，生产切换时建议改用明确的 `remote:bucket[/prefix]` 白名单模式。

请务必将下方 **YAML** 中的 `<OPERATOR_VERSION>` 替换为第 **2.1** 节中获取的实际版本号。

```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
          # 使用 Alauda 加固的 volsync 镜像，版本必须与 OperatorHub 完全一致。
          image: build-harbor.alauda.cn/acp/volsync:<OPERATOR_VERSION>
          # 注意：ACP 集群中，该镜像引用在 Pod 创建后会自动重写为内网镜像地址
          # （例如：registry.alauda.cn:60070/acp/volsync:<OPERATOR_VERSION
>）。
          # 这是预期行为。
          # 可通过 `kubectl describe pod <pod-name>` 查看实际 Image / ImageI
D。
          imagePullPolicy: IfNotPresent
          # 覆盖默认入口，直接调用镜像内的 rclone
          command: ["rclone"]
          args:
            - "sync"
            - "source-minio:"
            - "dest-ceph:"
            - "--progress"
            - "--create-empty-src-dirs"
            - "--ignore-errors"
            # --- 性能调优参数 ---
            - "--transfers=32"       # 并行文件传输数
            - "--checkers=64"        # 并行校验数
            - "--s3-chunk-size=64M" # 大对象分块大小，减少 RGW 碎片
            - "--fast-list"          # 使用 ListObjectsV2 减少 API 请求
            - "--metadata"           # 同步对象元数据
          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增数据，按需重复执行此步骤。

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

机制说明：默认情况下，Rclone 通过比较 Size 和 ModTime 决定是否需要传输，已存在且未修改的文件会被跳过。

阶段 3：最终切换

1. 停止写入：在应用层停止对 MinIO 的写入，或通过负载均衡/Ingress 阻断写流量。

2. 严格同步校验：

- 删除当前 Job：`kubectl -n migration-ops delete job volsync-s3-sync-job`
- 修改 `rclone-job.yaml`：去掉最终切换运行中的 `- "--ignore-errors"`，并在 `args` 中追加 `- "--checksum"`。此举避免掩盖失败对象，优先使用 Size+Checksum（可用时）进行差异检测。
- 一致性建议（必需）：切换前不要单纯依赖对象数量或 ETag。运行 `rclone check source-minio: dest-ceph: --download --checksum`（或对关键对象抽样下载并计算 SHA256）以减少因分段/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 状态变为 `Completed` 且日志无错误后，更新应用的 S3 Endpoint 和凭证为 Ceph RGW。

---

5. 故障排查与调优建议

现象	技术诊断	解决方案
ImagePullBackOff	节点无法拉取镜像，或版本不匹配。ACP 集群会自动将 `build-harbor.alauda.cn` 重写为内网镜像地址。	先运行 `kubectl describe pod <pod-name>` 验证实际 `Image`（是否被重写）和 `ImageID`，再确认 `<OPERATOR_VERSION>` 是否正确，检查网络/认证配置（`imagePullSecrets`）是否正确指向有效镜像仓库。
Pod OOMKilled	`--fast-list` 在大量小对象场景下批量加载元数据，消耗较高内存。	增加 Pod 内存限制至 8Gi 以上，或移除 `--fast-list`（但会增加 API 请求次数，遍历速度变慢）。
403 Forbidden	目标 Ceph 用户缺少桶创建权限。	检查 `CephObjectStoreUser` 配置，确保 `capabilities` 包含 `bucket: "*"`.
dial tcp: lookup "http: no such	Secret 配置格式错误，endpoint URL 被加了引	编辑 Secret，去除 **endpoint URL** 周围的引号，确保格式严格为：`endpoint`

现象	技术诊断	解决方案
host"	号。	`= http://...` 。
503 Service Unavailable	迁移并发过高，导致 Ceph RGW 写压力过大。	降低 Job 中 `args` 的 `--transfers` 参数，或增加 `--bwlimit` 限制传输带宽。

TopoLVM 本地存储

介绍

介绍

安装

安装

前提条件

操作步骤

操作指南

设备管理

前提条件

添加设备

监控与告警

监控

告警

实用指南

使用 Velero 备份和恢复 TopoLV

- 前提条件
- 限制
- 操作步骤

从 ACP TopoLVM 本地存储中移除

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

配置条带逻辑

- 前提条件
- 操作步骤

介绍

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

主要特性和优势：

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

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

安装

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

目录

前提条件

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

操作步骤

1 部署 Alauda Container Platform Storage Essentials

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

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

2 部署存储

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

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

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

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

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

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

7. 点击 **Next**，等待资源创建完成。

操作指南

设备管理

- 前提条件
- 添加设备

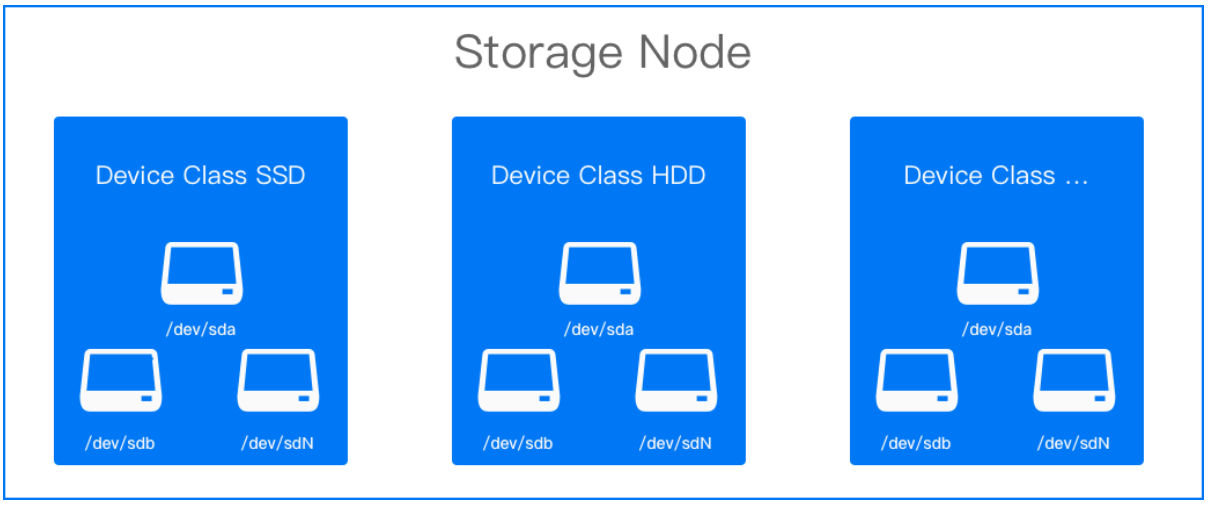
监控与告警

- 监控
- 告警

设备管理

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

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



目录

前提条件

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

- 节点上必须至少存在 1 个裸盘。

添加设备

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

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

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

监控与告警

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

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

目录

监控

性能监控

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

容量监控

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

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

告警

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

配置通知

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

处理告警

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

检查项	对应状态	原因说明
健康状态	Alert	由节点服务异常或设备类型问题引起。
服务状态	Unknown	节点处于 `notready` 状态，可能因网络故障或断电导致。
设备类型状态	Unavailable	使用的磁盘可能不是裸盘，或磁盘缺失。

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

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

事后分析

告警历史 记录了所有已触发且当前不需立即处理的告警。事后分析时应考虑：

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

实用指南

使用 Velero 备份和恢复 TopoLVM 文件系统 PVC

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

本指南专门针对 TopoLVM 文件系统的 PVC。

目录

前提条件

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

限制

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

操作步骤

第 1 步：配置备份仓库

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

第 2 步：执行备份

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

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

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

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

3. 备份完成后，验证 S3 桶中的数据（例如 MinIO）：

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

示例输出：

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

第 3 步：恢复集群

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

从 ACP TopoLVM 本地存储中移除磁盘、设备类卷组或节点

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

根据故障范围，本文档涵盖以下场景：

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

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

目录

检查设备类卷组是否仍可恢复

操作步骤

第 1 步：停止 topolvm-operator

第 2 步：查找受影响的 LVM 逻辑卷

第 3 步：查找受影响的 PVC 和 PV

第 4 步：停止使用受影响 PVC 的工作负载

第 5 步：删除受影响的 Kubernetes 存储资源

第 6 步：清理残留的 LVM 逻辑卷

第 7 步：从 LVM 卷组中移除缺失的物理卷

第 8 步：更新 TopolvmCluster 资源

第 9 步：更新 lvmdconfig ConfigMap

第 10 步：启动 topolvm-operator

第 11 步：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完全损坏

操作步骤

第 1 步：停止 topolvm-operator

第 2 步：查找受影响的 PVC 和 PV

第 3 步：停止使用受影响 PVC 的工作负载

第 4 步：删除受影响的 Kubernetes 存储资源

第 5 步：更新 TopolvmCluster 资源

第 6 步：更新 lvmdconfig ConfigMap

第 7 步：启动 topolvm-operator

第 8 步：验证设备类卷组已移除

场景 3：移除 TopoLVM 存储节点

用户场景

前提条件

操作步骤

第 1 步：停止 topolvm-operator

第 2 步：查找受影响的 PVC 和 PV

第 3 步：停止使用受影响 PVC 的工作负载

第 4 步：删除受影响的 Kubernetes 存储资源

第 5 步：更新 TopolvmCluster 资源

第 6 步：更新 lvmdconfig ConfigMap

第 7 步：启动 topolvm-operator

第 8 步：验证节点已移除

术语

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

选择正确的场景

请使用下表判断适用于您环境的场景。

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

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

场景 1：从设备类卷组中移除卷组设备

用户场景

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

前提条件

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

检查设备类卷组是否仍可恢复

在目标节点执行以下命令：

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

参数：

- `<vg-name>`：目标 LVM 卷组名称。

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

示例输出：

```
WARNING: Couldn't find device with uuid VJes6j-2a8V-8Cxf-eW84-yEJK-K24A-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

第 1 步：停止 topolvm-operator

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

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

第 2 步：查找受影响的 LVM 逻辑卷

在目标节点执行以下命令，查找使用故障磁盘的逻辑卷：

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

参数：

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

例如：

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

示例输出：

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

第 3 步：查找受影响的 PVC 和 PV

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

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

参数：

- `<lv-name>`：受影响的 LVM 逻辑卷名称。

示例输出：

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

输出说明：

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

第 4 步：停止使用受影响 PVC 的工作负载

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

第 5 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

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

参数：

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

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 6 步：清理残留的 LVM 逻辑卷

在目标节点执行以下命令，检查是否仍有逻辑卷残留：

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

参数：

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

若命令仍有输出，逐一删除残留逻辑卷：

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

参数：

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

如有需要，可添加 `--force`。

第 7 步：从 LVM 卷组中移除缺失的物理卷

在目标节点执行以下命令：

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

参数：

- `<vg-name>`：目标 LVM 卷组名称。

如有需要，可添加 `--force`。

第 8 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

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

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

之前：

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

之后：

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

第 9 步：更新 **lvmdconfig ConfigMap**

在 control-plane 节点执行以下命令：

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

参数：

- `<node-name>`：目标节点名称。

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

之前：

```
status.json: '{"node":"192.168.133.50","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-2ab8f0a2-7d1d-42d7-b
a6b-da94c6185c33","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdb","state":"Online"},{"name":"/dev/vdc","s
tate":"Online"}]}],"loops":[],"raids":[]}'
```

之后：

```
status.json: '{"node":"192.168.133.50","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-2ab8f0a2-7d1d-42d7-b
a6b-da94c6185c33","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdc","state":"Online"}]}],"loops":[],"raid
s":[]}'
```

第 10 步：启动 topolvm-operator

在 control-plane 节点执行以下命令：

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

第 11 步：验证卷组设备已移除

在 control-plane 节点执行以下命令：

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

确认目标节点的 `deviceStates` 中不再出现已移除的卷组设备。

场景 2：从设备类中移除设备类卷组

用户场景

- 当节点上的设备类卷组完全损坏，无法仅通过移除单个卷组设备恢复时，使用此操作步骤。

前提条件

- 目标节点仍在集群中且可访问。
- 目标设备类卷组已完全损坏。
- 移除目标设备类卷组后，节点上至少还保留有另一个设备类卷组。

检查设备类卷组是否完全损坏

在目标节点执行以下命令：

```
vgs
```

若输出中不再出现目标 `LVM 卷组`，则设备类卷组已完全损坏，可继续执行本操作步骤。

> **注意**
>
> 本场景移除的是节点上的整个设备类卷组，而非仅移除该卷组中的单个卷组设备。

操作步骤

第 1 步：停止 topolvm-operator

在 control-plane 节点执行以下命令：

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

第 2 步：查找受影响的 PVC 和 PV

在 control-plane 节点执行以下命令，查找目标节点上指定存储类关联的 PV、PVC 及 `logicalvolumes.topolvm.cybozu.com` 资源：

```
kubectl get pv -o json | jq -r --arg NODE <node-name> --arg SC <storageclass-name> '
  .items[]
  | select(.spec.nodeAffinity.required.nodeSelectorTerms[]?.matchExpressions[]? | select(.key=="topology.topolvm.cybozu.com/node") | .values[]? == $NODE)
  | select(.spec.storageClassName == $SC)
  | [.metadata.name, .spec.claimRef.namespace, .spec.claimRef.name]
  | @tsv
'
```

参数：

- `<node-name>`：目标节点名称。
- `<storageclass-name>`：目标设备类卷组关联的存储类名称。若涉及多个存储类，请分别执行查询。

示例输出：

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

输出说明：

- 第 1 列为 `PersistentVolume` 和 `logicalvolumes.topolvm.cybozu.com` 资源名称。
- 第 2、3 列为 `PersistentVolumeClaim` 的命名空间和名称。

若目标设备类卷组关联多个存储类，请对每个存储类重复执行查询及后续清理步骤。

第 3 步：停止使用受影响 PVC 的工作负载

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

第 4 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

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

参数：

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

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 5 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

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

在编辑器中，从目标节点配置中移除设备类卷组。例如，在 `nodeName: 192.168.140.13` 的配置中，移除 `className: hdd` 及其关联的 `volumeGroup` 和 `devices` 配置。

之前：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
          - className: hdd
            devices:
              - name: /dev/vdb
                type: disk
            volumeGroup: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
        nodeName: 192.168.140.13
```

之后：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
        nodeName: 192.168.140.13
```

若被移除的 `className` 条目含有 `default: true`，请指定另一个剩余的类为默认类。

第 6 步：更新 lvmdconfig ConfigMap

在 control-plane 节点执行以下命令：

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

参数：

- `<node-name>`：目标节点名称。

在编辑器中，从 `lvmd.yaml` 和 `status.json` 中移除对应目标设备类卷组的配置。例如，移除 `className: hdd` 的配置。

之前：

```
lvmd.yaml: |
  socket-name: /run/topolvm/lvmd.sock
  device-classes:
  - name: ssd
    volume-group: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
    default: true
    type: thick
  - name: hdd
    volume-group: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
    default: false
    type: thick
status.json: '{"node":"192.168.140.13","phase":"","failClasses":[],"s
uccessClasses":[{"className":"hdd","vgName":"hdd-97dc00f3-1df6-4f64-8
ddc-7b0b6c5d6de5","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdb","state":"Online"}]},{"className":"ss
d","vgName":"ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1","state":"Read
y","message":"create successful","deviceStates":[{"name":"/dev/vd
c","state":"Online"}]}],"loops":[],"raids":[]}'
```

之后：

```
lvmd.yaml: |
  socket-name: /run/topolvm/lvmd.sock
  device-classes:
  - name: ssd
    volume-group: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
    default: true
    type: thick
status.json: '{"node":"192.168.140.13","phase":"","failClasses":[],"s
uccessClasses":[{"className":"ssd","vgName":"ssd-4a8737fc-48d3-4c61-8
82d-0a5bcc6f77a1","state":"Ready","message":"create successful","devi
ceStates":[{"name":"/dev/vdc","state":"Online"}]}],"loops":[],"raid
s":[]}'
```

若被移除的 `className` 条目含有 `default: true`，请指定另一个剩余的类为默认类。

第 **7** 步：启动 **topolvm-operator**

在 control-plane 节点执行以下命令：

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

第 **8** 步：验证设备类卷组已移除

在 control-plane 节点执行以下命令：

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

确认目标节点不再显示已移除的设备类卷组，且剩余设备类卷组状态正常。

场景 **3**：移除 **TopoLVM** 存储节点

用户场景

- 目标节点不可恢复，或您决定永久从 TopolvmCluster 中移除该节点。

前提条件

- 您决定不保留目标节点上的任何设备类卷组。
- 使用目标节点本地卷的工作负载已停止或迁移至其他节点。
- 移除目标节点后，剩余节点仍能承载所需工作负载。

操作步骤

第 1 步：停止 **topolvm-operator**

在 control-plane 节点执行以下命令：

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

第 2 步：查找受影响的 **PVC** 和 **PV**

在 control-plane 节点执行以下命令，查找与目标节点关联的 PV、PVC 及 `logicalvolumes.topolvm.cybozu.com` 资源：

```
kubectl get pv -o json | jq -r --arg NODE <node-name> '
  .items[]
  | select(.spec.nodeAffinity.required.nodeSelectorTerms[]?.matchExpressions[]? | select(.key=="topology.topolvm.cybozu.com/node") | .values[]? == $NODE)
  | [.metadata.name, .spec.claimRef.namespace, .spec.claimRef.name]
  | @tsv
'
```

参数：

- `<node-name>`：目标节点名称。

第 3 步：停止使用受影响 **PVC** 的工作负载

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

第 4 步：删除受影响的 Kubernetes 存储资源

在 control-plane 节点执行以下命令：

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

参数：

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

若查询结果包含多个资源，按映射关系逐一删除。

若资源无法正常删除，可根据需要添加 `--force`。

第 5 步：更新 TopolvmCluster 资源

在 control-plane 节点执行以下命令：

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

在编辑器中，删除目标节点的整个配置块。例如，删除 `nodeName: 192.168.140.13` 的配置块。

之前：

```
spec:
  storage:
    deviceClasses:
      - classes:
          - className: hdd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: hdd-2ab8f0a2-7d1d-42d7-ba6b-da94c6185c33
        nodeName: 192.168.133.50
      - classes:
          - className: ssd
            default: true
            devices:
              - name: /dev/vdc
                type: disk
            volumeGroup: ssd-4a8737fc-48d3-4c61-882d-0a5bcc6f77a1
          - className: hdd
            devices:
              - name: /dev/vdb
                type: disk
            volumeGroup: hdd-97dc00f3-1df6-4f64-8ddc-7b0b6c5d6de5
        nodeName: 192.168.140.13
```

之后：

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

第 6 步：更新 **lvmdconfig ConfigMap**

在 control-plane 节点执行以下命令：

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

参数：

- `<node-name>`：目标节点名称。

在编辑器中，删除整个 `lvmd.yaml` 和 `status.json` 部分，不保留任何已移除节点的配置或状态。

第 7 步：启动 topolvm-operator

在 control-plane 节点执行以下命令：

```
kubectl -n nativestor-system scale --replicas 1 deployment topolvm-operator
```

第 8 步：验证节点已移除

在 control-plane 节点执行以下命令：

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

确认 `status.nodeStorageState` 中不再出现目标节点，例如 `192.168.140.13` 不再显示。

配置条带逻辑卷

当您向 LVM 逻辑卷写入数据时，文件系统会将数据分布到底层的物理卷上。您可以通过创建条带逻辑卷来控制数据写入物理卷的方式。对于大规模的顺序读写操作，这可以提高数据 I/O 的效率。

条带化通过以轮询方式将数据写入预定数量的物理卷来增强性能。使用条带化，I/O 可以并行进行。在某些情况下，每增加一个物理卷，性能几乎呈线性提升。

TopoLVM 通过在 `StorageClass` 中指定 `lvcreate-option-class` 来实现这一点。

目录

前提条件

- 设备类必须在单个节点上包含至少两个设备。

操作步骤

1 使用默认的 `lvcreate-option-class`

1. 进入 管理员。
2. 在左侧边栏，依次进入 存储管理 > 存储类。
3. 点击 创建存储类。
4. 选择 块存储。
5. 选择 **TopoLVM**，然后点击 下一步。
6. 配置所需的存储类参数。
7. 切换到 **YAML** 视图 并添加 `topolvm.io/lvcreate-option-class` 参数：

```
parameters:
  topolvm.io/lvcreate-option-class: striped-default
```

> **NOTE**
>
> `striped-default` 是 Alauda Container Platform 内置的 lvcreate-option-class。当 TopoLVM CSI 驱动创建逻辑卷时，它会自动向 lvcreate 命令追加 `--stripes=2` 和 `--stripesize=64`。

2 创建自定义 lvcreate-option-class（可选）

如果内置的 striped-default 类不能满足您的需求，您可以定义自定义的 LVCreateOptionClass：

```
cat << EOF | kubectl apply -f -
apiVersion: topolvm.cybozu.com/v2
kind: LVCreateOption
metadata:
  name: custom
  namespace: nativestor-system
spec:
  options:
  - name: striped-custom
    type: striped
    options:
    - --stripes=3
    - --stripesize=64
EOF
```

该清单创建了一个名为 striped-custom 的自定义 LVCreateOptionClass，具有 3 条条带和 64 KiB 的条带大小。应用后，您可以在 StorageClass 中通过 `topolvm.io/lvcreate-option-class: striped-custom` 引用它。