Menu

存储

介绍

介绍

概念

访问模式与卷模式

Kubernetes 中的访问模式

Kubernetes 中的卷模式

存储功能：快照与扩容

总结

核心概念

Persistent Volume (PV)

Persistent Volume Claim (PVC)

通用临时卷（Generic Ephemeral Volumes）

emptyDir

hostPath

ConfigMap

Secret

StorageClass

Container Storage Interface (CSI)

Persistent Volume

动态 Persistent Volume 与静态 Persistent Volume

Persistent Volume 的生命周期

功能指南

创建 CephFS 文件存储类型存储类

部署卷插件

创建存储类

创建 CephRBD 块存储类

部署卷插件

创建存储类

创建 **TopoLVM** 本地存储类

背景信息

部署卷插件

创建存储类

后续操作

创建 **NFS** 共享存储类

前提条件

部署 Alauda Container Platform NFS CSI 插件

创建 NFS 共享存储类

部署 **Volume Snapshot** 组件

通过 Web 控制台部署

通过 YAML 部署

创建 **PV**

前提条件

示例 PersistentVolume

通过 Web 控制台创建 PV

通过 CLI 创建 PV

相关操作

额外资源

创建 PVCs

先决条件

示例持久卷声明：

通过 Web 控制台创建持久卷声明

通过 CLI 创建持久卷声明

操作

通过 Web 控制台扩展持久卷声明存储容量

通过 CLI 扩展持久卷声明存储容量

其他资源

使用卷快照

先决条件

示例 VolumeSnapshot 自定义资源（CR）

通过 Web 控制台创建卷快照

通过 CLI 创建卷快照

从卷快照创建持久卷声明

额外资源

实用指南

通用临时卷

示例临时卷

主要特性

何时使用通用临时卷

它们与 emptyDir 的区别

使用 emptyDir

示例 emptyDir

可选的 Medium 设置

主要特性

常见用例

使用 NFS 配置持久存储

前提条件

操作步骤

通过分区导出实现磁盘配额

NFS 卷安全性

资源回收

第三方存储能力注解指南

1. 快速开始
2. 示例 ConfigMap
3. 更新已有能力描述
4. 与旧格式的兼容性
5. 常见问题解答

故障排除

从 PVC 扩容失败中恢复

操作步骤

额外提示

介绍

Kubernetes 提供灵活且可扩展的存储机制，用于在容器化环境中管理数据持久性。通过抽象存储资源（如 Volumes、PersistentVolumes 和 PersistentVolumeClaims），Kubernetes 将应用与底层存储解耦，支持动态供给、自动挂载以及跨节点的数据持久化。

其核心功能包括支持多种后端存储系统（例如，本地磁盘、NFS、云存储服务）、动态供给、访问模式控制（如读写权限）和生命周期管理，满足有状态应用的存储需求。对于需要高可用性、数据持久性和多租户隔离的企业级工作负载，Kubernetes 存储是重要的基础能力。

Kubernetes 的存储功能旨在帮助开发者、运维工程师和平台团队高效且安全地管理容器化工作负载中的数据。

Menu

概念

访问模式与卷模式

Kubernetes 中的访问模式

Kubernetes 中的卷模式

存储功能：快照与扩容

总结

核心概念

Persistent Volume (PV)

Persistent Volume Claim (PVC)

通用临时卷（Generic Ephemeral Volumes）

emptyDir

hostPath

ConfigMap

Secret

StorageClass

Container Storage Interface (CSI)

Persistent Volume

动态 Persistent Volume 与静态 Persistent Volume

Persistent Volume 的生命周期

访问模式与卷模式

在 Kubernetes 中，PersistentVolumeClaims（PVC）与 StorageClass 协同工作，用于管理工作负载如何配置和访问存储。在这一领域中，两个关键概念是 访问模式（**Access Modes**） 和卷模式（**Volume Modes**）。本文将探讨这些概念，并说明不同存储系统对它们的支持情况。

目录

Kubernetes 中的访问模式

各存储类对访问模式的支持情况

Kubernetes 中的卷模式

各存储类对卷模式的支持情况

存储功能：快照与扩容

总结

Kubernetes 中的访问模式

访问模式定义了一个卷可以被 Pods 挂载和使用的方式。主要的访问模式包括：

- **ReadWriteOnce (RWO)**：卷可以被单个节点以读写方式挂载。
- **ReadOnlyMany (ROX)**：卷可以被多个节点以只读方式挂载。
- **ReadWriteMany (RWX)**：卷可以被多个节点以读写方式挂载。

各存储类对访问模式的支持情况

存储类	支持 **RWO**	支持 **ROX**	支持 **RWX**
CephFS 文件存储	是	否	是
CephRBD 块存储	是	否	否
TopoLVM	是	否	否
NFS 共享存储	是	否	是

如上所示，**CephFS** 和 **NFS** 等基于文件的存储系统支持多节点的共享读写操作，适用于多副本或并发访问的场景。而像 **CephRBD** 和 **TopoLVM** 这样的块存储系统则更适合单节点独占访问。

Kubernetes 中的卷模式

卷模式定义了数据如何暴露给 Pods：

- **Filesystem**（文件系统）：将卷作为文件系统挂载进 Pods。
- **Block**（块设备）：将卷作为原始块设备呈现。

各存储类对卷模式的支持情况

存储类	类型	支持的卷模式
CephFS 文件存储	文件存储	文件系统
CephRBD 块存储	块存储	文件系统、块设备
TopoLVM	块存储	文件系统、块设备
NFS 共享存储	文件存储	文件系统

如 **CephRBD** 和 **TopoLVM** 等块存储系统同时支持文件系统模式与原始块模式，为不同应用需求提供了更大的灵活性。而 **CephFS** 与 **NFS** 等文件存储系统则仅支持文件系统模式。

存储功能：快照与扩容

Kubernetes 还支持一些高级功能，如卷快照与 PVC 动态扩容，具体支持情况如下：

存储类	支持卷快照	支持扩容
CephFS 文件存储	支持	支持
CephRBD 块存储	支持	支持
TopoLVM	支持	支持
NFS 共享存储	不支持	不支持

仅当使用 StorageClass 动态创建 PVC 时，平台才支持卷快照功能。这一功能适合用于数据备份和环境克隆等场景。

总结

在 Kubernetes 中配置存储时，了解 PVC 及其背后 StorageClass 的 访问模式 与 卷模式 是选择合适解决方案的关键。像 CephFS 与 NFS 这样的文件存储适用于共享访问场景，而 CephRBD 与 TopoLVM 等块存储则更适合高性能、单节点部署。此外，对快照与扩容等功能的支持也能显著提升存储的灵活性和数据管理能力。

核心概念

Kubernetes 存储围绕三个关键概念展开：**PersistentVolume (PV)**、**PersistentVolumeClaim (PVC)** 和 **StorageClass**。它们定义了集群内存储的请求、分配和配置方式。在底层，**CSI**（Container Storage Interface）驱动程序通常负责实际的存储供应和挂载。下面我们简要介绍每个组件，并突出 CSI 驱动的作用。

目录

Persistent Volume (PV)
Persistent Volume Claim (PVC)
通用临时卷（Generic Ephemeral Volumes）
emptyDir
hostPath
ConfigMap
Secret
StorageClass
Container Storage Interface (CSI)

Persistent Volume (PV)

PersistentVolume (PV) 是集群中已被预配的存储（可以是管理员静态配置，也可以通过 StorageClass 动态配置）。它代表底层存储——例如云提供商的磁盘或网络附加文件系统——并作为集群中的一种资源存在，类似于节点。

Persistent Volume Claim (PVC)

PersistentVolumeClaim (PVC) 是对存储的请求。用户定义所需的存储容量和访问模式（例如读写）。如果有合适的 PV 可用或可以通过 StorageClass 动态供应，PVC 就会与该 PV “绑定”。绑定后，Pod 可以引用 PVC 来持久化或共享数据。

通用临时卷（Generic Ephemeral Volumes）

Kubernetes 的通用临时卷功能允许你在 Pod 生命周期中使用由 CSI 驱动的 `temporary` 卷，这类似于 `emptyDir`，但功能更强大，支持挂载任何类型的 CSI 卷（支持快照、扩展等功能）。

更多用法请参考 Generic ephemeral volumes

emptyDir

1. emptyDir 是一种临时存储卷，类型为空目录。
2. 它在 Pod 调度到节点时创建，存储位于该节点的本地文件系统（默认是节点磁盘）上。
3. 当 Pod 被删除时，emptyDir 中的数据也会被清除。

更多用法请参考 Using an emptyDir

hostPath

在 Kubernetes 中，hostPath 卷是一种特殊类型的卷，它将宿主节点文件系统中的文件或目录直接映射到 Pod 的容器中。

- 它允许 Pod 访问宿主节点上的文件或目录。
- 适用于：
  - 访问宿主级资源（例如 Docker socket）

- 调试
- 使用节点上已有的数据

ConfigMap

Kubernetes 中的 ConfigMap 是一种 API 对象，用于以键值对形式存储非敏感的配置信息。它使配置与应用代码解耦，提高应用的可移植性和易管理性。

Secret

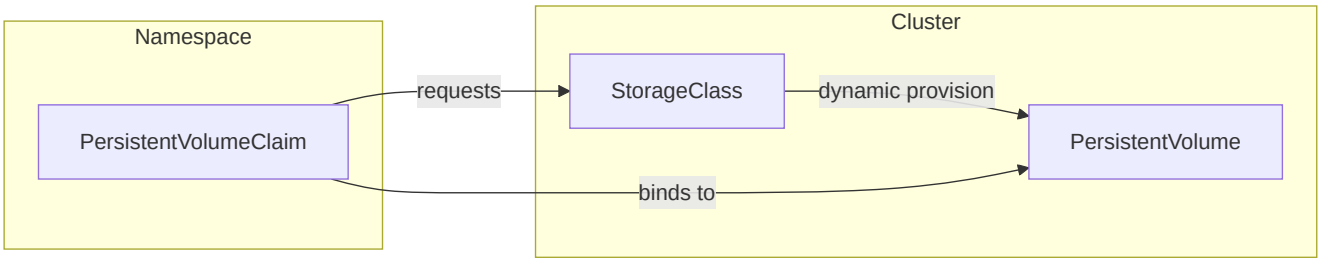
在 Kubernetes 中，Secret 是一种 API 对象，用于存储敏感数据，例如：

- 密码
- OAuth 令牌
- SSH 密钥
- TLS 证书
- 数据库凭据

Secret 通过避免将敏感数据直接存储在 Pod 规格或容器镜像中，帮助保护这些数据。

StorageClass

StorageClass 描述了卷应如何动态供应。它映射到特定的供应器（通常是 CSI 驱动），并可以包含存储层级、性能特征或其他后端配置参数。通过创建多个 StorageClass，可以向开发者提供多种类型的存储选择。



图示：PVC、PV 和 StorageClass 之间的关系。

Container Storage Interface (CSI)

Container Storage Interface (CSI) 是 Kubernetes 用于集成存储驱动的标准 API。它允许第三方存储提供商构建外部插件，这意味着你可以安装或更新存储驱动而无需修改 Kubernetes 本身。

一个 CSI 驱动 通常包含两个组件：

1. 控制器组件：运行在集群中（通常以 Deployment 形式），负责高级操作，如创建或删除卷。对于网络存储，它还可能负责将卷挂载或卸载到节点。

2. 节点组件：运行在每个节点上（通常以 DaemonSet 形式），负责在该节点上挂载和卸载卷。它与 kubelet 通信，确保卷对 Pod 可用。

当用户创建引用使用 CSI 驱动的 StorageClass 的 PVC 时，CSI 驱动会监测该请求并相应地供应存储（如果需要动态供应）。存储创建完成后，驱动通知 Kubernetes，Kubernetes 创建对应的 PV 并将其绑定到 PVC。每当 Pod 使用该 PVC 时，驱动的节点组件负责挂载卷，使存储在容器内可用。

通过利用 **PV**、**PVC**、**StorageClass** 和 **CSI**，Kubernetes 实现了强大且声明式的存储管理方式。管理员可以定义一个或多个 StorageClass 来表示不同的存储后端或性能层级，而开发者只需通过 PVC 请求存储，无需关心底层基础设施。

Persistent Volume

PersistentVolume（PV）表示 Kubernetes 集群中与后端存储卷的映射关系，作为 Kubernetes API 资源存在。它是由管理员统一创建和配置的集群资源，负责抽象实际的存储资源，构成集群的存储基础设施。

PersistentVolume 拥有独立于 Pod 的生命周期，能够实现 Pod 数据的持久化存储。

管理员可以手动创建静态 PersistentVolume，也可以基于存储类生成动态 PersistentVolume。如果开发人员需要为应用获取存储资源，可以通过 PersistentVolumeClaim（PVC）进行申请，PVC 会匹配并绑定合适的 PersistentVolume。

目录

动态 Persistent Volume 与静态 Persistent Volume

Persistent Volume 的生命周期

动态 Persistent Volume 与静态 Persistent Volume

平台支持管理员管理两种类型的 PersistentVolume，即动态和静态 Persistent Volume。

- 动态 **Persistent Volume**：基于存储类实现。存储类由管理员创建，定义了描述存储资源类别的 Kubernetes 资源。当开发人员创建关联存储类的 PersistentVolumeClaim 时，平台会根据 PersistentVolumeClaim 和存储类中配置的参数动态创建合适的 PersistentVolume，并绑定到该 PersistentVolumeClaim，实现存储资源的动态分配。

- 静态 **Persistent Volume**：由管理员手动创建的 Persistent Volume。目前支持创建 **HostPath** 或 **NFS** 共享存储 类型的静态 Persistent Volume。当开发人员创建不使用存储类的 PersistentVolumeClaim 时，平台会根据 PersistentVolumeClaim 中配置的参数匹配并绑定合适的静态 PersistentVolume。

  - **HostPath**：使用节点主机上的文件目录（不支持本地存储）作为后端存储，例如：`/etc/kubernetes`。通常仅适用于单计算节点集群的测试场景。

  - **NFS** 共享存储：指网络文件系统，是 Persistent Volume 常见的后端存储类型。用户和程序可以像访问本地文件一样访问远程系统上的文件。

Persistent Volume 的生命周期

1. **Provisioning**（配置）：管理员手动创建静态 Persistent Volume，创建后 Persistent Volume 进入 **Available** 状态；或者平台根据关联存储类的 PersistentVolumeClaim 动态创建合适的 Persistent Volume。

2. **Binding**（绑定）：静态 Persistent Volume 匹配并绑定到 PersistentVolumeClaim 后，进入 **Bound** 状态；动态 Persistent Volume 根据请求动态创建并匹配 PersistentVolumeClaim，创建成功后也进入 **Bound** 状态。

3. **Using**（使用）：开发人员将 PersistentVolumeClaim 关联到计算组件的容器实例，使用 Persistent Volume 映射的后端存储资源。

4. **Releasing**（释放）：开发人员删除 PersistentVolumeClaim 后，Persistent Volume 被释放。

5. **Reclaiming**（回收）：Persistent Volume 被释放后，根据 Persistent Volume 或存储类的回收策略参数对其进行回收操作。

Menu

功能指南

创建 **CephFS** 文件存储类型存储类

部署卷插件

创建存储类

创建 **CephRBD** 块存储类

部署卷插件

创建存储类

创建 **TopoLVM** 本地存储类

背景信息

部署卷插件

创建存储类

后续操作

创建 **NFS** 共享存储类

前提条件

部署 Alauda Container Platform NFS CSI 插件

创建 NFS 共享存储类

部署 Volume Snapshot 组件

通过 Web 控制台部署

通过 YAML 部署

创建 PV

前提条件

示例 PersistentVolume

通过 Web 控制台创建 PV

通过 CLI 创建 PV

相关操作

额外资源

创建 PVCs

先决条件

示例持久卷声明：

通过 Web 控制台创建持久卷声明

通过 CLI 创建持久卷声明

操作

通过 Web 控制台扩展持久卷声明存储容量

通过 CLI 扩展持久卷声明存储容量

其他资源

使用卷快照

先决条件

示例 VolumeSnapshot 自定义资源（CR）

通过 Web 控制台创建卷快照

通过 CLI 创建卷快照

从卷快照创建持久卷声明

额外资源

创建 CephFS 文件存储类型存储类

CephFS 文件存储是内置的 Ceph 文件存储系统，为平台提供基于 Container Storage Interface (CSI) 的存储访问方式，提供安全、可靠且可扩展的共享文件存储服务，适用于文件共享和数据备份等场景。操作前，您需要先创建 CephFS 文件存储类。

在 Persistent Volume Claim (PVC) 中绑定存储类后，平台会根据持久卷声明动态在节点上创建持久卷，供业务应用使用。

目录

部署卷插件

创建存储类

部署卷插件

点击 部署 后，在 分布式存储 页面，进行创建存储服务或接入存储服务。

创建存储类

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 创建存储类。

注意：以下内容以表单形式示例展示，您也可以选择使用 YAML 创建。

4. 选择 **CephFS** 文件存储，点击 下一步。

5. 按照以下说明配置相关参数。

参数	说明
回收策略	持久卷的回收策略。 - Delete：当持久卷声明被删除时，绑定的持久卷也会被删除。 - Retain：即使持久卷声明被删除，绑定的持久卷仍然保留。
访问模式	当前存储支持的所有访问模式。后续声明持久卷时只能选择其中一种。 - ReadWriteOnce (RWO)：可被单个节点以读写方式挂载。 - ReadWriteMany (RWX)：可被多个节点以读写方式挂载。
分配项目	请分配可以使用此类型存储的项目。 如果当前没有需要使用此类型存储的项目，也可暂不分配，后续再更新。

提示：以下参数需在分布式存储中设置，并会直接应用到此处。

- 存储集群：当前集群内置的 Ceph 存储集群。
- 存储池：存储集群中用于数据存储的逻辑分区。

6. 点击 创建。

创建 CephRBD 块存储类

CephRBD 块存储是平台内置的 Ceph 块存储，提供基于 Container Storage Interface (CSI) 的存储访问方式，能够提供高 IOPS 和低延迟的存储服务，适用于数据库、虚拟化等场景。使用前需要先创建 CephRBD 块存储类。

当 Persistent Volume Claim (PVC) 绑定到该存储类后，平台会根据 Persistent Volume Claim 动态创建 Persistent Volume 供业务应用使用。

目录

部署卷插件

创建存储类

部署卷插件

点击 部署 后，在 分布式存储 页面，进行创建存储服务或接入存储服务。

创建存储类

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 创建存储类。

注意：以下内容为表单形式示例，也可以选择 YAML 方式完成操作。

4. 选择 **CephRBD** 块存储，点击 下一步。

5. 按需配置参数。

参数	描述
文件系统	默认为 **EXT4**，Linux 的日志文件系统，能够提供范围文件存储并处理大文件。文件系统容量可达 1 EiB，支持的文件大小最高可达 16 TiB。
回收策略	持久卷的回收策略。 - Delete：绑定的持久卷会随着持久卷声明一起删除。 - Retain：即使持久卷声明被删除，绑定的持久卷仍会被保留。
访问模式	仅支持 ReadWriteOnce (RWO)：可被单个节点以读写模式挂载。
分配项目	请分配可以使用此类型存储的项目。 如果当前没有需要此类型存储的项目，可以选择不分配，后续再更新。

提示：以下参数需在分布式存储中设置，并会直接应用到此处。

- 存储集群：当前集群内置的 Ceph 存储集群。
- 存储池：存储集群中用于存储数据的逻辑分区。

6. 点击 创建。

创建 **TopoLVM** 本地存储类

TopoLVM 是一种基于 LVM 的本地存储解决方案，提供简单、易维护且高性能的本地存储服务，适用于数据库、中间件等场景。使用前需要先创建 TopoLVM 存储类。

当 Persistent Volume Claim（PVC）绑定到该存储类后，平台会根据 Persistent Volume Claim 动态在节点上创建持久卷供业务应用使用。

目录

背景信息

使用优势

使用场景

约束与限制

部署卷插件

创建存储类

后续操作

背景信息

使用优势

- 相较于远程存储（如 **NFS** 共享存储）：TopoLVM 类型的存储位于节点本地，提供更好的 IOPS 和吞吐性能，以及更低的延迟。

- 相较于 hostPath（如 **local-path**）：虽然两者均为节点本地存储，但 TopoLVM 支持灵活调度容器组到资源充足的节点，避免因资源不足导致容器组无法启动的问题。
- TopoLVM 默认支持自动扩容。修改 Persistent Volume Claim 中的存储配额后，无需重启容器组即可自动完成扩容。

使用场景

- 仅需临时存储时，如开发调试。
- 存储 I/O 需求较高时，如实时索引。

约束与限制

请尽量仅在应用层能实现数据复制和备份的应用（如 MySQL）中使用本地存储，避免因本地存储缺乏数据持久性保障而导致数据丢失。

了解更多↗

部署卷插件

点击部署后，在新打开的页面进行配置本地存储。

创建存储类

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 创建存储类。
4. 选择 **TopoLVM**，然后点击 下一步。
5. 按照以下说明配置存储类参数。

注意：以下内容以表单示例形式展示，也可选择使用 YAML 创建。

参数	说明
名称	存储类名称，在当前集群中必须唯一。
显示名称	用于帮助识别或筛选的名称，如存储类的中文描述。
设备类	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似特性的存储设备。如无特殊需求，使用 自动分配 设备类。
文件系统	• **XFS** 是高性能的日志文件系统，适合处理并行 I/O 工作负载，支持大文件处理和流畅的数据传输。 • **EXT4** 是 Linux 下的日志文件系统，提供范围文件存储，支持大文件处理，最大文件系统容量为 1 EiB，最大文件大小为 16 TiB。
回收策略	持久卷的回收策略。 • Delete：绑定的持久卷会随 PVC 一起删除。 • Retain：即使 PVC 被删除，绑定的持久卷仍然保留。
访问模式	ReadWriteOnce (RWO)：可被单个节点以读写方式挂载。
PVC 重建	支持跨节点的 PVC 重建。启用时需配置 重建等待时间。当使用该存储类创建的 PVC 所在节点故障时，等待时间后 PVC 会自动在其他节点重建，确保业务连续性。 注意： • 重建的 PVC 不包含原始数据。 • 请确保存储节点数量大于应用实例副本数，否则会影响 PVC 重建。
分配项目	该类型的持久卷声明只能在指定项目中创建。 若当前无分配项目，也可后续更新。

6. 确认配置信息无误后，点击 创建 按钮。

后续操作

准备就绪后，可通知开发人员使用 TopoLVM 功能。例如，在容器平台的 存储 > 持久卷声明 页面创建 Persistent Volume Claim 并绑定到 TopoLVM 存储类。

创建 NFS 共享存储类

基于社区的 NFS CSI（Container Storage Interface）存储驱动，提供访问多个 NFS 存储系统或账户的能力。

与传统的 NFS 访问客户端-服务器模式不同，NFS 共享存储采用社区的 NFS CSI（Container Storage Interface）存储插件，更符合 Kubernetes 设计理念，并允许客户端访问多个服务器。

目录

前提条件
部署 Alauda Container Platform NFS CSI 插件
通过 Web 控制台部署
通过 YAML 部署
创建 NFS 共享存储类

前提条件

- 必须配置好 NFS 服务器，并获取其访问方式。目前平台支持三种 NFS 协议版本：`v3`、`v4.0` 和 `v4.1`。可以在服务器端执行 `nfsstat -s` 来查看版本信息。

部署 Alauda Container Platform NFS CSI 插件

通过 Web 控制台部署

1. 进入 **Administrator**。
2. 在左侧导航栏点击 **Storage** > **StorageClasses**。
3. 点击 **Create StorageClass**。
4. 在 **NFS CSI** 右侧点击 Deploy，跳转到 **Plugins** 页面。
5. 在 `Alauda Container Platform NFS CSI` 插件右侧点击⋮> **Install**。
6. 等待部署状态显示 **Deployment Successful** 后完成部署。

通过 YAML 部署

参考 Installing via YAML

`Alauda Container Platform NFS CSI` 是一个 **Non-config plugin**，模块名为 `nfs`

创建 NFS 共享存储类

1. 点击 **Create Storage Class**。

   注意：以下内容以表单形式展示，您也可以选择通过 YAML 完成操作。

2. 选择 **NFS CSI**，点击 **Next**。

3. 按照以下说明配置相关参数。

参数	说明
Name	存储类名称，在当前集群内必须唯一。
Service Address	NFS 服务器的访问地址。例如：`192.168.2.11`。
Path	服务器节点上 NFS 文件系统的挂载路径。例如：`/nfs/data`。
NFS Protocol	当前支持三种版本：`v3`、`v4.0` 和 `v4.1`。

参数	说明
Version	
Reclaim Policy	持久卷的回收策略。 - Delete：当持久卷声明被删除时，绑定的持久卷也会被删除。 - Retain：即使持久卷声明被删除，绑定的持久卷仍然保留。
Access Modes	当前存储支持的所有访问模式。在后续声明持久卷时，只能选择其中一种模式挂载持久卷。 - ReadWriteOnce (RWO)：允许单个节点以读写方式挂载。 - ReadWriteMany (RWX)：允许多个节点以读写方式挂载。 - ReadOnlyMany (ROX)：允许多个节点以只读方式挂载。
Allocated Projects	请分配可以使用此类存储的项目。 如果当前没有项目需要此类存储，可以暂时不分配，后续再更新。
subDir	使用 NFS 共享存储类创建的每个 PersistentVolumeClaim (PVC) 对应 NFS 共享中的一个子目录。默认子目录名称采用 `${pv.metadata.name}`（即 PersistentVolume 名称）模式生成。如果默认生成的名称不符合需求，可以自定义子目录命名规则。

NOTE

`subDir` 字段仅支持以下三种变量，NFS CSI Driver 会自动解析：

- `${pvc.metadata.namespace}`：PVC 所在命名空间。
- `${pvc.metadata.name}`：PVC 名称。
- `${pv.metadata.name}`：PV 名称。

`subDir` 命名规则必须保证子目录名称唯一，否则多个 PVC 可能共享同一子目录，导致数据冲突。

推荐配置示例：

- `${pvc.metadata.namespace}_${pvc.metadata.name}_${pv.metadata.name}`
- `<cluster-identifier>_${pvc.metadata.namespace}_${pvc.metadata.name}_${pv.metadata.name}`

该配置适用于多个 Kubernetes 集群共享同一 NFS 服务器的场景，通过在子目录命名规则中加入集群标识（如集群名称），确保集群间区分明确。

不推荐配置示例：

- `${pvc.metadata.namespace}-${pvc.metadata.name}-${pv.metadata.name}` 避免使用 - 作为分隔符，可能导致子目录名称歧义。例如：两个 PVC 分别命名为 `ns-1/test` 和 `ns/1-test`，都可能生成相同子目录 `ns-1-test`。
- `${pvc.metadata.namespace}/${pvc.metadata.name}/${pv.metadata.name}` 不要配置 subDir 生成嵌套目录。NFS CSI Driver 删除 PVC 时只会删除最末级目录 `${pv.metadata.name}`，会在 NFS 服务器上留下孤立的父目录。

4. 确认配置信息无误后，点击 **Create**。

部署 **Volume Snapshot** 组件

Volume snapshot 指的是持久卷的快照，是持久卷在某一特定时间点的副本。如果集群使用支持快照功能的持久卷，可以部署 volume snapshot 组件以启用该功能。

目前平台仅支持为通过存储类动态创建的 PVC 创建 volume snapshot。您可以基于这些快照创建新的 PVC 绑定。

提示：通过快照创建 PVC 时支持的访问模式与通过存储类创建 PVC 时支持的访问模式不同，下面表格中以加粗标示。

用于创建 **Volume Snapshot** 的存储类	单节点读写 **(RWO)**	多节点只读 **(ROX)**	多节点读写 **(RWX)**
TopoLVM	支持	不支持	不支持
CephRBD Block Storage	支持	不支持	不支持
CephFS File Storage	支持	支持	支持

目录

通过 Web 控制台部署

通过 YAML 部署

通过 **Web** 控制台部署

1. 进入 管理员。

2. 点击 **Marketplace** > 集群插件，进入 集群插件 列表页面。

3. 找到 `Alauda Container Platform Snapshot Management` 集群插件，点击 安装，等待部署成功。

通过 YAML 部署

参考 通过 YAML 安装

`Alauda Container Platform Snapshot Management` 是一个非配置插件，模块名为 `snapshot`

创建 PV

手动创建类型为 **HostPath** 或 **NFS Shared Storage** 的静态持久卷。

- **HostPath**：将容器所在主机的文件目录挂载到容器内指定路径（对应 Kubernetes 的 HostPath），允许容器使用主机的文件系统作为持久存储。如果主机不可访问，则 HostPath 可能无法访问。
- **NFS Shared Storage**：NFS Shared Storage 使用社区 NFS CSI（Container Storage Interface）存储插件，更符合 Kubernetes 设计理念，支持多个服务的客户端访问能力。使用前请确保当前集群已部署 **NFS** 存储插件。

目录

前提条件

示例 PersistentVolume

通过 Web 控制台创建 PV

存储信息

通过 CLI 创建 PV

访问模式

回收策略

相关操作

额外资源

前提条件

- 确认要创建的持久卷大小，并确保后端存储系统当前有能力提供相应存储容量。
- 获取后端存储访问地址、要挂载的文件路径、凭证访问（如需）等相关信息。

示例 PersistentVolume

```yaml
# example-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi (1)
  accessModes:
    - ReadWriteOnce (2)
  persistentVolumeReclaimPolicy: Retain (3)
  storageClassName: manual (4)
  hostPath: (5)
    path: "/mnt/data"
```

1. 存储容量。
2. 卷的挂载方式。
3. PVC 删除后的处理方式（Retain、Delete、Recycle）。
4. StorageClass 名称（用于动态绑定）。
5. 存储后端类型。

通过 Web 控制台创建 PV

1. 进入 **Administrator**。
2. 在左侧导航栏点击 **Storage Management** > **Persistent Volumes (PV)**。
3. 点击 **Create Persistent Volume**。

4. 参考以下说明配置参数后点击 **Create**。

存储信息

类型	参数	说明
HostPath	Path	支持存储卷的节点上文件目录的路径。例如：`/etc/kubernetes`。
NFS Shared Storage	Server Address	NFS 服务器的访问地址。
	Path	NFS 文件系统在服务器节点上的挂载路径，如 `/nfs/data`。
	NFS Protocol Version	平台当前支持的 NFS 协议版本为 `v3`、`v4.0` 和 `v4.1`。可在服务器端执行 `nfsstat -s` 查看版本信息。

通过 CLI 创建 PV

```
kubectl apply -f example-pv.yaml
```

访问模式

持久卷的访问模式由后端存储设置的相关参数决定。

访问模式	含义
ReadWriteOnce (RWO)	只能被单个节点以读写方式挂载。
ReadWriteMany (RWX)	可以被多个节点以读写方式挂载。
ReadOnlyMany (ROX)	可以被多个节点以只读方式挂载。

回收策略

回收策略	含义
Delete	删除持久卷声明的同时，删除绑定的持久卷及后端存储卷资源。 注意：NFS Shared Storage 类型的 PV 不支持 **Delete** 回收策略。
Retain	即使持久卷声明被删除，绑定的持久卷和存储数据仍会保留。之后需要手动处理存储数据并删除持久卷。

相关操作

您可以点击列表页右侧的⋮，或在详情页右上角点击 **Operations**，根据需要更新或删除持久卷。

删除持久卷适用于以下两种场景：

- 删除未绑定的持久卷：该卷未被写入且不再需要写入，删除后释放对应存储空间。
- 删除 **Retain** 状态的持久卷：持久卷声明已删除，但由于 Retain 回收策略，持久卷未被同时删除。如果持久卷中的数据已备份至其他存储或不再需要，删除该持久卷也可释放对应存储空间。

额外资源

- 创建 PVC

创建 PVCs

创建一个持久卷声明（PVC），并根据需要设置请求的持久卷（PV）的参数。

您可以通过可视化 UI 表单或使用自定义 YAML 编排文件来创建持久卷声明。

目录

先决条件
示例持久卷声明：
通过 Web 控制台创建持久卷声明
通过 CLI 创建持久卷声明
操作
通过 Web 控制台扩展持久卷声明存储容量
通过 CLI 扩展持久卷声明存储容量
其他资源

先决条件

确保在命名空间中有足够的剩余 存储 配额，以满足此创建操作所需的存储大小。

示例持久卷声明：

```yaml
# example-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
  namespace: k-1
  annotations: {}
  labels: {}
spec:
  storageClassName: cephfs
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 4Gi
```

通过 **Web** 控制台创建持久卷声明

1. 转到 容器平台。

2. 在左侧边栏中点击 存储 **>** 持久卷声明（**PVC**）。

3. 点击 创建 **PVC**。

4. 根据需要配置参数。

   注意：以下内容作为使用表单方法的示例提供；您也可以切换到 YAML 模式以完成操作。

参数	描述
名称	持久卷声明的名称，必须在当前命名空间内唯一。
创建	- 动态创建：根据存储类动态生成持久卷并绑定。 - 静态绑定：根据配置的参数和现有持久卷进行匹配和绑定。

参数	描述
方式	
存储类	选择动态创建方法后，平台将根据指定存储类中的描述动态创建持久卷。
访问模式	- ReadWriteOnce (RWO)：可以被单个节点以读写模式挂载。 - ReadWriteMany (RWX)：可以被多个节点以读写模式挂载。 - ReadOnlyMany (ROX)：可以被多个节点以只读模式挂载。 提示：建议考虑计划绑定到当前持久卷声明的工作负载实例数量和部署控制器的类型。例如，在创建多实例部署（Deployment）时，由于所有实例使用相同的持久卷声明，因此不建议选择只能附加到单个节点的 RWO 访问模式。
容量	请求的持久卷的大小。
卷模式	- 文件系统：将持久卷绑定为挂载到 Pod 的文件目录。此模式适用于任何类型的工作负载。 - 块设备：将持久卷绑定为挂载到 Pod 的原始块设备。此模式仅适用于虚拟机。
更多	- 标签 - 注释 - 选择器：选择静态绑定方法后，您可以使用选择器来定位带有特定标签的持久卷。持久卷标签可用于表示存储的特殊属性，例如磁盘类型或地理位置。

5. 点击 创建。等待持久卷声明状态变为 `Bound`，表示持久卷已成功匹配。

通过 **CLI** 创建持久卷声明

```
kubectl apply -f example-pvc.yaml
```

操作

- 绑定持久卷声明：在创建需要持久数据存储的应用程序或工作负载时，绑定持久卷声明以请求符合要求的持久卷。
- 使用卷快照创建持久卷声明：这有助于备份应用程序数据并在需要时恢复，确保业务应用程序数据的可靠性。请参阅 使用卷快照。
- 删除持久卷声明：您可以在详情页面的右上角点击 操作 按钮，根据需要删除持久卷声明。在删除之前，请确保持久卷声明未绑定到任何应用程序或工作负载，并且不包含任何卷快照。删除持久卷声明后，平台将根据回收策略处理持久卷，这可能会清除持久卷中的数据并释放存储资源。请根据数据安全考虑谨慎操作。

通过 Web 控制台扩展持久卷声明存储容量

1. 在左侧导航栏中，点击存储 > 持久卷声明（PVC）。
2. 找到持久卷声明并点击⋮> 扩展。
3. 填写新容量。
4. 点击扩展。扩展过程可能需要一些时间，请耐心等待。

通过 CLI 扩展持久卷声明存储容量

```
kubectl patch pvc example-pvc -n k-1 --type='merge' -p '{
  "spec": {
    "resources": {
      "requests": {
        "storage": "6Gi"
      }
    }
  }
}'
```

INFO

当 PVC 扩展在 Kubernetes 中失败时，管理员可以手动恢复持久卷声明（PVC）状态并取消扩展请求。请参阅 从 PVC 扩展失败中恢复

其他资源

- 如何注释第三方存储能力

使用卷快照

卷快照是持久卷声明（PVC）的一个时间点副本，可用于配置新的持久卷声明（预填充快照数据）或将现有持久卷声明回滚到先前状态，从而实现备份应用数据并根据需要恢复，确保应用数据的可靠性。

目录

先决条件

示例 VolumeSnapshot 自定义资源（CR）

通过 Web 控制台创建卷快照

基于指定持久卷声明（PVC）创建卷快照

以自定义方式创建卷快照

通过 CLI 创建卷快照

从卷快照创建持久卷声明

方法一

方法二

额外资源

先决条件

- 管理员已为当前集群部署了卷快照组件 **Snapshot Controller** 并在存储集群中启用了快照相关功能。
- 持久卷声明必须动态创建，并且其状态必须为 **Bound**。

- 绑定到持久卷声明的存储类必须支持快照功能，例如 **CephRBD** 内置存储、**CephFS** 内置存储或 **TopoLVM**。

示例 **VolumeSnapshot** 自定义资源（**CR**）

这将使用 CSI 快照类创建 example-pvc `PVC` 的快照。

```
# example-snapshot.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-pvc-20250527-111124
  namespace: k-1
  labels:
    snapshot.cpaas.io/sourcepvc: example-pvc
  annotations:
    cpaas.io/description: demo
spec:
  volumeSnapshotClassName: csi-cephfs-snapshotclass
  source:
    persistentVolumeClaimName: example-pvc
```

通过 **Web** 控制台创建卷快照

基于指定持久卷声明（**PVC**）创建卷快照

方法一

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Storage > Persistent Volume Claims (PVC)**。
3. 在列表中点击相应持久卷声明旁的⋮，选择 **Create Volume Snapshot**。
4. 填写快照描述。此描述可以帮助您记录持久卷的当前状态，例如 *应用升级前*。

5. 点击 **Create**。快照所需时间取决于网络状况和数据量，请耐心等待。

   当快照状态变为 `Available` 时，表示创建成功。

方法二

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Storage > Persistent Volume Claims (PVC)**。
3. 在列表中点击持久卷声明的名称。
4. 切换到 **Volume Snapshots** 标签。
5. 点击 **Create Volume Snapshot**，根据需要配置相关参数。
6. 点击 **Create**。快照所需时间取决于网络状况和数据量，请耐心等待。

   当快照状态变为 `Available` 时，表示创建成功。

以自定义方式创建卷快照

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Storage > Volume Snapshots**。
3. 点击 **Create Volume Snapshot**，根据需要配置相关参数。
4. 点击 **Create**。快照所需时间取决于网络状况和数据量，请耐心等待。

   当快照状态变为 `Available` 时，表示创建成功。

通过 **CLI** 创建卷快照

```
kubectl apply -f example-snapshot.yaml
```

从卷快照创建持久卷声明

目前，平台仅支持使用从具有 动态配置 的存储类创建的 PVC 来创建卷快照。您可以基于该快照创建新的 PVC 并进行绑定。

注意：从快照创建 PVC 时支持的访问模式与从存储类创建 PVC 时支持的访问模式不同，如表中 加粗 部分所示。

用于创建卷快照的存储类	单节点读写（**RWO**）	多节点只读（**ROX**）	多节点读写（**RWX**）
TopoLVM	支持	不支持	不支持
CephRBD 块存储	支持	不支持	不支持
CephFS 文件存储	支持	支持	支持

方法一

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Storage > Persistent Volume Claims (PVC)**。
3. 在列表中点击持久卷声明的名称。
4. 切换到 **Volume Snapshots** 标签。
5. 点击相应卷快照旁的⋮，选择 **Create Persistent Volume Claim**，配置相关参数。
6. 点击 **Create**。

方法二

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Storage > Volume Snapshots**。
3. 在列表中点击相应卷快照旁的⋮，选择 **Create Persistent Volume Claim**，配置相关参数。

4. 点击 **Create**。

额外资源

- 创建 PVCs

Menu

实用指南

通用临时卷

示例临时卷

主要特性

何时使用通用临时卷

它们与 emptyDir 的区别

使用 **emptyDir**

示例 emptyDir

可选的 Medium 设置

主要特性

常见用例

使用 **NFS** 配置持久存储

前提条件

操作步骤

通过分区导出实现磁盘配额

NFS 卷安全性

资源回收

第三方存储能力注解指南

1. 快速开始

2. 示例 ConfigMap

3. 更新已有能力描述

4. 与旧格式的兼容性

5. 常见问题解答

通用临时卷

Kubernetes 中的通用临时卷是一项功能，允许您使用现有的存储类和 CSI 驱动程序为每个 Pod 提供临时（短暂）卷，而无需预先定义持久卷声明（PVC）。

它们结合了动态供给的灵活性和 Pod 级卷声明的简单性。

- 它们是临时卷，自动：
  - 在 Pod 启动时创建
  - 在 Pod 终止时删除
- 使用与持久卷声明相同的底层机制
- 需要支持动态供给的 CSI（容器存储接口）驱动程序

目录

示例临时卷

主要特性

何时使用通用临时卷

它们与 emptyDir 的区别

示例临时卷

这将使用指定的 `StorageClass` 自动为 Pod 创建一个临时 PVC。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ephemeral-demo
spec:
  containers:
    - name: app
      image: busybox
      command: ["sh", "-c", "echo hello > /data/hello.txt && sleep 3600"]
      volumeMounts:
        - mountPath: /data
          name: ephemeral-volume
  volumes:
    - name: ephemeral-volume
      ephemeral: (1)
        volumeClaimTemplate:
          metadata:
            labels:
              type: temporary
          spec:
            accessModes: [ "ReadWriteOnce" ]
            resources:
              requests:
                storage: 1Gi
            storageClassName: standard
```

1. `Pod` 将使用此模板创建一个 `PVC`。

主要特性

特性	描述
临时性	当 Pod 被删除时，卷也会被删除
动态供给	由任何支持动态供给的 CSI 驱动程序提供支持
无需单独的 **PVC**	卷声明直接嵌入在 Pod 规格中
基于 **CSI**	与任何兼容的 CSI 驱动程序（EBS、RBD、Longhorn 等）一起工作

何时使用通用临时卷

- 当您需要具有以下特性的临时存储时：
  - 可调整大小的卷
  - 快照
  - 加密
  - 非节点本地存储（例如，云块存储）
- 理想的用途：
  - 缓存中间数据
  - 临时工作目录
  - 流水线、AI/ML 工作流

它们与 **emptyDir** 的区别

特性	`emptyDir`	通用临时卷
支持的存储	节点的本地磁盘或内存	任何支持 CSI 的后端
存储特性	基本	支持快照、加密等
使用场景	简单的临时存储	复杂的临时存储需求
可重新调度	否（与节点绑定）	是（如果 CSI 卷可附加）

使用 **emptyDir**

在 Kubernetes 中，emptyDir 是一种简单的临时卷类型，为 Pod 在其生命周期内提供临时存储。它在 Pod 被分配到节点时创建，并在 Pod 从该节点移除时删除。

目录

示例 emptyDir

可选的 Medium 设置

主要特性

常见用例

示例 **emptyDir**

该 Pod 创建了一个挂载在 /data 的临时卷，并与容器共享。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: emptydir-demo
spec:
  containers:
    - name: app
      image: busybox
      command: ["sh", "-c", "echo hello > /data/hello.txt && sleep 3600"]
      volumeMounts:
        - mountPath: /data
          name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

可选的 **Medium** 设置

您可以选择数据存储的位置：

```yaml
emptyDir:
  medium: "Memory"
```

Medium	描述
(默认)	根据您的环境使用节点的磁盘、SSD 或网络存储
`Memory`	使用 RAM (`tmpfs`) 以获得更快的访问速度（但不持久）

主要特性

特性	描述
初始为空	创建时没有数据

特性	描述
跨容器共享	同一卷可以被 Pod 中的多个容器使用
随 Pod 删除	当 Pod 被移除时，卷也会被销毁
节点本地	卷存储在节点的本地磁盘或内存中
快速	适合性能敏感的临时存储空间

常见用例

- 缓存中间构建工件
- 缓冲日志
- 临时工作目录
- 在同一 Pod 中的容器之间共享数据（如 sidecar）

使用 **NFS** 配置持久存储

Alauda Container Platform 集群支持使用 NFS 的持久存储。Persistent Volumes（PVs）和 Persistent Volume Claims（PVCs）为项目内存储卷的配置和使用提供了抽象层。虽然可以将 NFS 配置细节直接嵌入 Pod 定义中，但这种方式不会将卷创建为独立的、隔离的集群资源，增加了冲突的风险。

目录

前提条件

操作步骤

创建 PV 的对象定义

验证 PV 是否创建成功

创建引用该 PV 的 PVC

验证持久卷声明是否创建成功

通过分区导出实现磁盘配额

NFS 卷安全性

组 ID

用户 ID

导出设置

资源回收

前提条件

- 底层基础设施中必须存在存储，才能在 Alauda Container Platform 中挂载为卷。

- 要配置 NFS 卷，只需提供 NFS 服务器列表和导出路径。

操作步骤

1 创建 PV 的对象定义

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-example 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 10.0.0.3 6
  persistentVolumeReclaimPolicy: Retain 7
EOF
```

1. 卷的名称。
2. 存储容量。
3. 虽然看似用于控制对卷的访问，实际上它类似于标签，用于匹配 PVC 和 PV。目前，基于 accessModes 不会强制执行访问规则。
4. 使用的卷类型，此处为 nfs 插件。
5. NFS 服务器地址。
6. NFS 导出路径。
7. PVC 删除后卷的处理策略（Retain、Delete、Recycle）。

2 验证 PV 是否创建成功

命令

```
kubectl get pv
```

示例输出

```
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
pv-nfs-example   1Gi        RWO            Retain           Available
10s
```

3 创建引用该 PV 的 PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
  - ReadWriteOnce (1)
  resources:
    requests:
      storage: 1Gi (2)
  volumeName: pv-nfs-example (3)
  storageClassName: ""
```

1. 访问模式不用于安全控制，而是作为标签匹配 PV 和 PVC。
2. 该声明请求容量为 1Gi 或更大容量的 PV。
3. 要使用的 PV 名称。

4 验证持久卷声明是否创建成功

命令

```
kubectl get pvc
```

示例输出

```
NAME           STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS
AGE
nfs-claim1     Bound    pv-nfs-example   1Gi        RWO
10s
```

通过分区导出实现磁盘配额

要强制执行磁盘配额和大小限制，可以利用磁盘分区。将每个分区作为专用导出点，每个导出对应一个独立的 PersistentVolume（PV）。

虽然 Alauda Container Platform 要求 PV 名称唯一，但管理员需确保每个导出的 NFS 卷的服务器地址和路径唯一。

这种分区方式实现了精确的容量管理。开发者请求指定容量（例如 10Gi）的持久存储，ACP 会匹配容量至少满足该请求的分区/导出对应的 PV。请注意：配额限制作用于分配分区/导出内的可用存储空间。

NFS 卷安全性

本节介绍 NFS 卷的安全机制，重点是权限匹配。假设读者具备 POSIX 权限、进程 UID 和附加组的基础知识。

开发者通过以下方式请求 NFS 存储：

- 通过名称引用 PersistentVolumeClaim（PVC），或
- 在 Pod 规范的 volumes 部分直接配置 NFS 卷插件。

在 NFS 服务器上，/etc/exports 文件定义了可访问目录的导出规则。每个导出目录保留其原生的 POSIX 所有者/组 ID。

Alauda Container Platform 的 NFS 插件关键行为：

1. 挂载卷到容器时，保留源目录的精确 POSIX 所有权和权限
2. 运行容器时不强制进程 UID 与挂载所有权匹配——这是有意的安全设计

例如，考虑一个 NFS 目录的服务器端属性：

命令

```
ls -l /share/nfs -d
```

示例输出

```
drwxrws---. nfsnobody 5555   /share/nfs
```

命令

```
id nfsnobody
```

示例输出

```
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

此时，容器必须以 UID 65534（nfsnobody 所有者）运行，或在其附加组中包含 5555，才能访问该目录。

NOTE

注意
65534 所有者 ID 仅为示例。虽然 NFS 的 root_squash 会将 root（uid 0）映射为 nfsnobody（uid 65534），但 NFS 导出可以拥有任意所有者 ID。NFS 导出不强制必须是所有者 65534。

组 ID

推荐的 NFS 访问管理（当导出权限固定时）
当无法修改 NFS 导出权限时，推荐通过附加组管理访问。

在 Alauda Container Platform 中，附加组是控制共享文件存储（如 NFS）访问的常用机制。

与块存储对比：块存储卷（如 iSCSI）的访问通常通过在 Pod 的 securityContext 中设置 fsGroup 来管理，该方法依赖挂载时更改文件系统组所有权。

NOTE

通常建议使用附加组 ID 而非用户 ID 来获得持久存储访问权限。

由于示例目标 NFS 目录的组 ID 是 5555，Pod 可以在其 securityContext 的 supplementalGroups 中定义该组 ID。例如：

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1. securityContext 必须定义在 Pod 级别，而非某个具体容器下。
2. 这是为 Pod 定义的 GID 数组，此处仅包含一个元素，多个 GID 用逗号分隔。

用户 ID

用户 ID 可在容器镜像中定义，也可在 Pod 定义中指定。

NOTE

通常建议使用附加组 ID 来获得持久存储访问权限，而非使用用户 ID。

以上示例目标 NFS 目录中，容器需要将 UID 设置为 65534（暂不考虑组 ID），可在 Pod 定义中添加：

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

1. Pod 包含针对每个容器的 securityContext 定义，以及适用于所有容器的 Pod 级 securityContext。
2. 65534 是 nfsnobody 用户。

导出设置

为使任意容器用户能读写卷，NFS 服务器上的每个导出卷应满足以下条件：

- 每个导出必须使用如下格式导出：

```
# 将 10.0.0.0/24 替换为可信任的 CIDR 或主机
/<example_fs> 10.0.0.0/24(rw,sync,root_squash,no_subtree_check)
```

- 防火墙必须配置允许访问挂载点的流量。
  - 对于 NFSv4，配置默认端口 2049（nfs）。

```
iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

  - 对于 NFSv3，需要配置三个端口：2049（nfs）、20048（mountd）和 111（portmapper）。

```
iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- NFS 导出和目录必须设置为目标 Pod 可访问。要么将导出所有权设置为容器的主 UID，要么通过 supplementalGroups 为 Pod 提供组访问权限，如上文组 ID 所示。

资源回收

NFS 实现了 Alauda Container Platform 的 Recyclable 插件接口。基于每个持久卷设置的策略，自动流程处理资源回收任务。

默认情况下，PV 的回收策略为 Retain。

当 PVC 的声明被删除且 PV 被释放后，该 PV 对象不应被重用。应创建一个新的 PV，使用与原 PV 相同的基本卷信息。

例如，管理员创建了名为 nfs1 的 PV：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

用户创建 PVC1，绑定到 nfs1。用户删除 PVC1，释放对 nfs1 的声明，导致 nfs1 状态变为 Released。如果管理员想继续使用相同的 NFS 共享，应创建一个新的 PV，使用相同的 NFS 服务器信息，但 PV 名称不同：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

不建议删除原 PV 并用相同名称重新创建。尝试手动将 PV 状态从 Released 改为 Available 会导致错误和潜在数据丢失。

第三方存储能力注解指南

> 功能概述： 通过在 `kube-public` 命名空间中添加一个 **StorageDescription** 类型的 ConfigMap，平台会自动检测每个第三方 StorageClass 的快照支持情况以及支持的卷模式和访问模式（包括块设备专用访问模式）。PVC 创建界面将仅显示有效选项，帮助您轻松选择和使用合适的存储功能。

目录

1. 快速开始
   1.1 创建或更新 ConfigMap
   1.2 填充 `data` 字段
   1.3 应用配置
2. 示例 ConfigMap
3. 更新已有能力描述
4. 与旧格式的兼容性
5. 常见问题解答

---

1. 快速开始

1.1 创建或更新 ConfigMap

> 重要提示： 请在 `kube-public` 命名空间内 执行以下操作，否则平台无法识别存储能力。

编辑或创建一个名称以 `sd-` 开头的 ConfigMap，例如 `sd-capabilities-enhanced`：

```
kubectl -n kube-public edit configmap sd-capabilities-enhanced
```

必需的标签

```
metadata:
  labels:
    features.alauda.io/type: StorageDescription
```

1.2 填充 `data` 字段

每个 `key` 对应一个 StorageClass 的 `provisioner`，其值是描述该存储能力的 YAML 字符串。主要字段说明：

字段	类型	说明
`snapshot`	`Boolean`	表示是否支持卷快照
`volumeMode`	`List[String]`	支持的卷模式；至少包含 `Filesystem` 或 `Block` 中的一个
`accessModes`	`List[String]`	当 `volumeMode` 为 `Filesystem` 时可用的访问模式
`blockAccessModes`	`List[String]`	块设备卷专用的访问模式（可选）

> 如果省略 `blockAccessModes`，平台会对块设备卷回退使用 `accessModes`。

1.3 应用配置

```
kubectl apply -f sd-capabilities-enhanced.yaml
```

应用后，UI 会自动调整可用选项，例如：

- 选择 **Block** 卷模式时，访问模式下拉框会显示 `blockAccessModes` 中的选项。
- 如果 `snapshot: true`，则 PVC 页面会启用与快照相关的操作。

2. 示例 ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: sd-capabilities-enhanced
  namespace: kube-public
  labels:
    features.alauda.io/type: StorageDescription
data:
  storage.advanced-block-fs.com: |-
    snapshot: true
    volumeMode:
      - Filesystem
      - Block
    accessModes:
      - ReadWriteOnce
      - ReadOnlyMany
    blockAccessModes:
      - ReadWriteOnce
  storage.filesystem-basic.com: |-
    snapshot: false
    volumeMode:
      - Filesystem
    accessModes:
      - ReadWriteOnce
      - ReadWriteMany
```

3. 更新已有能力描述

1. 找到需要修改的 `provisioner` 键。
2. 调整字段值以反映实际能力。
3. 使用 `kubectl apply -f ...` 重新应用 ConfigMap。平台会轮询更新并自动刷新 UI，您也可以刷新浏览器以立即查看更改。

4. 与旧格式的兼容性

- 如果缺少 `blockAccessModes`，块设备卷将继承 `accessModes`。
- 无需删除旧的 ConfigMap，只需添加新字段即可实现平滑升级。

5. 常见问题解答

现象	可能原因	解决方案
块设备卷访问模式列表为空	`blockAccessModes` 和 `accessModes` 均为空	至少提供其中一个
UI 仍显示过时的能力信息	ConfigMap 未保存或浏览器缓存	使用 `kubectl get cm` 验证，刷新页面

Menu

故障排除

从 **PVC** 扩容失败中恢复

操作步骤

额外提示

从 PVC 扩容失败中恢复

当 Kubernetes 中的 PVC 扩容失败时，管理员可以手动恢复 Persistent Volume Claim (PVC) 状态并取消扩容请求。

目录

操作步骤

额外提示

操作步骤

1. 修改绑定到 PVC 的 Persistent Volume (PV) 的回收策略为 `Retain`。为此，编辑对应的 PV 并将 `persistentVolumeReclaimPolicy` 字段设置为 `Retain`。
2. 删除原有的 PVC。
3. 手动编辑 PV，删除其规格中的 `claimRef` 条目。这样可以确保新的 PVC 能绑定到该 PV，使 PV 状态变为 `Available`。
4. 重新创建一个较小的 PVC，或者创建一个底层存储提供商支持的大小的 PVC。
5. 在新的 PVC 中显式指定 `volumeName` 字段，使其与原 PV 名称匹配。这样可以确保新的 PVC 准确绑定到指定的 PV。
6. 最后，恢复 PV 的原始回收策略。

额外提示

- 确保所使用的 `StorageClass` 已启用卷扩容功能，即将 `allowVolumeExpansion` 设置为 `true`。
- 操作时请谨慎，避免数据丢失风险。