

# CeaStor CSI Plugin 3.2.2

## 使用指导

---

文档密级：公开

文档版本：01

发布日期：2024-05-30

中电云计算技术有限公司

### 【版权声明】

版权所有 © 中电云计算技术有限公司 2024。 保留一切权利。

本文档的版权归中电云计算技术有限公司所有。非经中电云计算技术有限公司书面许可，任何人不得以包括通过程序或设备监视、复制、传播、展示、镜像、上载、下载、摘编等方式或以其他方式擅自使用本文档的任何内容。

### 【商标声明】



和本文档所示其他中电云计算技术有限公司及/或其他关联公司的商标均为中电云计算技术有限公司及/或其关联公司所有。未经中电云计算技术有限公司及/或其关联公司书面许可，任何人不得以任何形式使用，也不得向他人表明您有权展示、使用或做其他处理。如您有宣传、展示等任何使用需要，您必须取得中电云计算技术有限公司及/或其关联公司事先书面授权。

本文档中出现的其他公司的商标或注册商标，由各自的所有人拥有。

### 【注意】

您购买的产品、服务或特性等应以中电云计算技术有限公司商业合同中的约定为准，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，中电云计算技术有限公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容可能会不定期进行更新。本文档仅作为使用指导，其中的陈述、信息或建议等均不构成任何明示或暗示的担保。

# 前言

## 概述





本文档主要介绍 CeaStor CSI 插件的使用指导，包括安装配置、部署使用等信息，帮助您快速上手使用 CeaStor CSI 插件。

## 读者对象

本文档适用于以下读者：

- 技术支持工程师
- 交付工程师
- 维护工程师

## 符号约定

 警告	该标志后的注释需给予格外关注，不当的操作可能会对人身造成伤害。
 注意	提醒操作中应注意的事项，不当的操作可能会导致数据丢失或者设备损坏。 “注意”不涉及人身伤害。
 说明	对正文的重点信息进行必要的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。
 提示	配置、操作或使用产品的技巧、窍门。

## 修订记录

文档版本	发布时间	修订说明
01	2024-05-30	第一次正式发布。

# 目 录

<b>1 概述</b>	<b>1</b>
1.1 简介	1
1.2 特性列表	1
1.3 兼容性	2
1.4 关键规格	3
<b>2 安装准备</b>	<b>4</b>
2.1 检查 CSI 依赖环境	4
2.2 准备 CSI 组件包	4
2.2.1 了解 CSI 组件包	4
2.2.2 获取 CSI 组件包	9
2.2.3 上传 CSI 组件包	9
2.3 准备配置文件	9
2.3.1 准备 CSI 配置文件	9
2.3.2 准备 PVC 配置文件	10
2.4 准备相关工具	16
<b>3 安装 CSI</b>	<b>17</b>
3.1 安装并配置 CSI	17
3.2 配置并部署 PVC	19
3.3 启用 NVMe/iSCSI 服务	20
3.3.1 配置 NVMe 启动器	20
3.3.2 配置 iSCSI 启动器	21
3.4 启用多路径	22
3.4.1 启用多路径访问 (NVMe-oF)	22
3.4.2 启用多路径访问 (iSCSI)	23
<b>4 使用 CSI</b>	<b>24</b>
4.1 管理 PVC	24
4.1.1 动态创建卷 (PVC)	24
4.1.2 克隆卷 (PVC)	24
4.1.3 删除卷 (PVC)	25
4.1.4 扩容卷 (PVC)	26
4.1.5 挂载 PVC	28
4.1.6 卸载 PVC	29

4.2 管理 VolumeSnapshot .....	31
4.2.1 创建卷快照 .....	31
4.2.2 克隆卷快照 .....	33
4.2.3 删除卷快照 .....	33
5 管理 CSI .....	35
5.1 升级 CSI .....	35
5.2 固化和查看 CSI 日志 .....	36
5.3 手动卸载 CSI .....	36
5.4 查询 CSI 预置 yaml 文件部署结果 .....	37
6 附录 .....	40
6.1 获取硬盘池支持的冗余策略 .....	40
6.2 分配 K8s 集群节点角色 .....	40
6.3 装载 CSI 镜像包 .....	41
6.4 安装第三程序 .....	41
6.4.1 安装 nvme-cli .....	41
6.4.2 安装 open-iscsi .....	42
6.4.3 安装 DM-Multipath .....	42

# 1 概述

## 1.1 简介

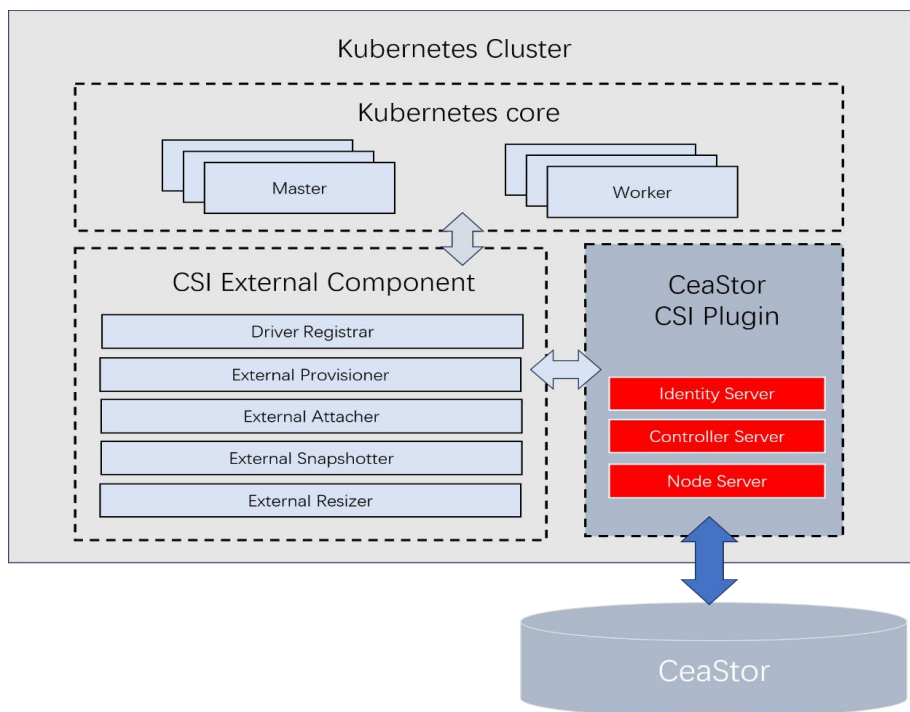
Kubernetes（简称 K8s），是一个可移植、可扩展的开源容器编排管理平台，用于管理容器化的工作负载和服务。

CSI（Container Storage Interface，容器存储接口）是一种行业标准，用于将块存储系统暴露给 Kubernetes 等容器编排系统（CO）上的容器工作负载。CeaStor CSI 插件用于向 Kubernetes 的容器工作负载提供存储服务，是 CeaStor 存储系统在 Kubernetes 环境中使用的必须插件。

通过在 Kubernetes 集群安装 CeaStor CSI 插件和部署 PVC，对接 CeaStor 存储系统资源，向 Kubernetes 提供 PVC（Persistent Volume Claim，持久卷声明）能力。

Kubernetes 通过 sidecar 组件注册监听 Kubernetes 对象资源，并发起对 CeaStor CSI 插件的调用。CeaStor CSI 插件将 sidecar 发起的调用在 CeaStor 存储系统上实施，例如在 Kubernetes 上发起创建一个持久卷（Persistent Volume，PV）操作，在 CeaStor 上被实施为创建一个 CBD 卷/文件系统。CeaStor CSI 插件在 Kubernetes 集群和 CeaStor 存储系统之间的工作原理图如下所示：

图 1-1 CeaStor CSI 插件工作原理图



## 1.2 特性列表

CeaStor CSI 支持的功能特性如下表：

表 1-1 支持的功能特性

特性功能	功能描述	使用限制
创建卷	动态创建卷（创建PVC）	卷容量1MiB~512TiB
	克隆卷（克隆PVC）	卷容量1MiB~512TiB，容量大于等于源卷。
删除卷	删除卷	不存在子卷或快照
扩容卷	扩容卷（扩容PVC） 若PV已挂载，包含扩容文件系统。	扩容大小需大于原卷大小。 <ul style="list-style-type: none"> <li>ROX 不支持在线扩容。</li> <li>RWX 默认不支持在线扩容，需配置启动项字段--EnableRWXExpendOnline=true</li> </ul>
创建快照	创建卷快照	-
删除快照	删除卷快照	-
挂载PVC	将PVC挂载给Pod使用 支持以块设备（Block模式）或文件系统（Filesystem模式）类型。	<ul style="list-style-type: none"> <li>Block 模式支持 RWO/ROX/RWX</li> <li>Filesystem 模式支持 RWO/ROX</li> </ul>
卸载PVC	将PVC从Pod中卸载	-
卷容量统计	统计卷已使用容量	-

## 1.3 兼容性

CeaStor CSI 插件支持的容器管理平台，以及操作系统和多路径版本信息。

表 1-2 支持的容器管理平台

容器管理平台	版本
Kubernetes	1.18~1.26

表 1-3 支持的操作系统及版本

操作系统名称	操作系统版本	原生 Multipath 版本
CCLinux x86_64 CCLinux ARM	20.09.2	随OS自带，支持FC/iSCSI

表 1-4 兼容性要求

支持功能	兼容性要求
支持Mount模式文件系统	Kubernetes 1.18以及以上
支持卷快照与克隆	Kubernetes 1.20以及以上
支持卷扩容	Kubernetes 1.24以及以上

## 1.4 关键规格

CeaStor CSI 插件关键规格：

表 1-5 关键规格说明

规格项	规格说明
Volume	受CeaStor存储系统卷规格限制，单集群最多支持创建10万个卷。
Snapshot	受CeaStor存储系统卷快照规格限制，单个卷最多支持创建256个快照。



# 2 安装准备

## 2.1 检查CSI依赖环境

在安装和配置 CeaStor CSI 插件前，请先检查 CSI 插件运行环境是否满足配置需求。

表 2-1 依赖环境检查项

环境	检查项	检查说明
Kubernetes集群 管理平台	管理平台运行正常。	确认容器平台状态为健康。
	管理平台版本满足兼容性要求。	满足 <a href="#">表1-4</a> 要求。
	主机连通性配置	确认与CeaStor存储系统之间网络连通性正常。
Kubernetes集群 业务主机	多路径软件配置	（可选）若需开启多路径，请确保所有需要启用CeaStor CSI插件的节点已经安装多路径软件。
	NVMe客户端软件配置	（可选）若选择使用NVMe-OF协议，请确保所有需要启动CeaStor CSI插件的节点已安装NVMe客户端软件。
	iSCSI客户端软件配置	（可选）若选择使用iSCSI协议，请确保所有需要启动CeaStor CSI插件的节点已安装iSCSI客户端软件。
CeaStor存储系统	License授权信息	<ul style="list-style-type: none"><li>• 确认 License 授权时间是否在使用期内。</li><li>• 确认 License 授权容量是否满足业务需求。</li></ul>
	系统集群运行状态	确认集群运行状态为“健康”。
	系统主机服务状态	确认主机服务状态为“运行正常”。
	存储系统网络状态	确认主机服务状态为“运行正常”。
	硬盘池状态和用途	确认存在用途为“块服务”的硬盘池，且状态为“正常”。
	访问路径状态	确认访问路径状态为“健康”。

## 2.2 准备CSI组件包

### 2.2.1 了解 CSI 组件包

#### 组件包名称

CeaStor CSI 组件包命名规则为：CeaStor-CSI-{版本号}-{版本编号}-{发版时间}.tar.gz。例如 CeaStor-CSI-3.2.1-2906-20240324003106.tar.gz。

#### 组件包目录结构

CSI 组件包目录结构如下：

CeaStor-CSI-3.2.1-xxxx-yyyyMMddHHmmss

- |— csi\_image\_list.json
- |— images
  - | |— amd64
    - | | |— cds-csi-nvmf\_amd64
    - | | |— csi-snapshot-validation-webhook\_amd64
    - | | |— csi-snapshot-controller\_amd64
    - | | |— csi-node-driver-registrar\_amd64
    - | | |— csi-livenessprobe\_amd64
    - | | |— csi-external-snapshotter\_amd64
    - | | |— csi-external-resizer\_amd64
    - | | |— csi-external-provisioner\_amd64
    - | | |— csi-external-attacher\_amd64
    - | | |— cluster-storage-operator\_amd64
    - | | |— cluster-csi-snapshot-controller-operator\_amd64
  - | |— arm64
    - | | |— cds-csi-nvmf\_arm64
    - | | |— csi-snapshot-validation-webhook\_arm64
    - | | |— csi-snapshot-controller\_arm64
    - | | |— csi-node-driver-registrar\_arm64
    - | | |— csi-livenessprobe\_arm64
    - | | |— csi-external-snapshotter\_arm64
    - | | |— csi-external-resizer\_arm64
    - | | |— csi-external-provisioner\_arm64
    - | | |— csi-external-attacher\_arm64
    - | | |— cluster-storage-operator\_arm64
    - | | |— cluster-csi-snapshot-controller-operator\_arm64
- |— deploy
  - | |— kubernetes
    - | | |— configmap-csi-iscsiadm.yaml
    - | | |— configmap-csi-multipathd.yaml
    - | | |— configmap-csi-multipath.yaml
    - | | |— csi\_controller\_deployment.yaml
    - | | |— csi-iscsi-controller.yaml
    - | | |— csi-iscsi-driver.yaml
    - | | |— csi-iscsi-node.yaml
    - | | |— csi-node-rbac.yaml
    - | | |— csi-nvmf-controller.yaml
    - | | |— csi-nvmf-driver.yaml
    - | | |— csi-nvmf-node.yaml
    - | | |— operand\_rbac.yaml
    - | | |— serviceaccount.yaml
    - | | |— snapshot.storage.k8s.io\_volumesnapshotclasses.yaml
    - | | |— snapshot.storage.k8s.io\_volumesnapshotcontents.yaml
    - | | |— snapshot.storage.k8s.io\_volumesnapshots.yaml
- |— examples
  - | |— kubernetes
    - | | |— clone.yaml

```

|      |—— pod-block.yaml
|      |—— pod.yaml
|      |—— pvc.yaml
|      |—— restore.yaml
|      |—— secret.yaml
|      |—— storageclass.yaml
|      |—— volumesnapshotclass.yaml
|      |—— volumesnapshot-dynamic.yaml

```

## 镜像包

在 `images/amd64` 和 `images/arm64` 路径下，为 Kubernetes 集群需装载的 CeaStor CSI 镜像包。镜像包需装载节点和说明如下表所示：

表 2-2 镜像包说明

镜像包名称	说明	装载节点
csi-snapshot-controller	CSI snapshot controller组件镜像	全部master节点
csi-external-snapshotter	CSI external snapshotter组件镜像	
csi-external-resizer	CSI external resizer组件镜像	
csi-external-provisioner	CSI external provisioner组件镜像	
csi-external-attacher	CSI external attacher组件镜像	
csi-snapshot-validation-webhook	（可选）CSI snapshot validation webhook 镜像	任意 master 节点
csi-livenessprobe	（可选）CSI健康检查	
cluster-storage-operator	（可选）定义OpenShift Container Platform集群范围内的存储默认设置，确保OpenShift Container Platform集群存在一个默认存储类。	
cluster-csi-snapshot-controller-operator	（可选）定义CSI Snapshot Controller部署和配置。 CSI Snapshot Controller负责监控VolumeSnapshot CRD 对象，并管理卷快照的创建和删除生命周期。	
cds-csi-nvmf_	CeaStor CSI Plugin镜像包。启动节点如master，需要增加启动项配置：-- IsControllerServer = true	需使用CSI的worker或/master节点
csi-node-driver-registrar	CSI driver registry组件镜像	

## yaml 配置文件

在 `deploy/kubernetes` 路径下，为 CeaStor CSI 组件的 yaml 配置文件。配置文件需装载节点和说明如下表所示：



注意

CeaStor CSI 提供的预置 yaml 配置文件，请勿随意修改。若需修改 yaml 文件配置，请联系技术支持。

表 2-3 yaml 配置文件说明

文件名称	说明	装载节点
configmap-csi-iscsiadm.yaml	iSCSI控制命令配置。在容器中映射iscsiadm命令。	任意master节点
configmap-csi-multipathd.yaml	iSCSI多路径命令配置。在容器中映射multipathd命令。	
configmap-csi-multipath.yaml	iSCSI多路径命令配置。在容器中映射multipath命令。	
csi_controller_deployment.yaml	快照控制容器配置。	
csi-iscsi-controller.yaml	CSI控制器配置，包含容器node-registrar、csi-provisioner、csi-attacher、csi-resizer、csi-snapshotter等。	
csi-iscsi-driver.yaml	CSI驱动配置。	
csi-iscsi-node.yaml	worker节点配置，包含容器node-registrar、csi-iscsi-plugin。	
csi-node-rbac.yaml	授权节点驱动程序操作相关API的权限。	
csi-nvmf-controller.yaml	CSI控制器配置，包含容器node-registrar、csi-provisioner、csi-attacher、csi-resizer、csi-snapshotter等。	
csi-nvmf-driver.yaml	CSI驱动配置。	
csi-nvmf-node.yaml	worker节点配置，包含容器node-registrar、csi-nvmf-plugin。	
operand_rbac.yaml	授权快照驱动程序操作相关API的权限配置。	
serviceaccount.yaml	服务账户配置。	
snapshot.storage.k8s.io_volumesnapshotclasses.yaml	K8s用户自定义资源volumesnapshotclasses配置。	
snapshot.storage.k8s.io_volumesnapshotcontents.yaml	K8s用户自定义资源volumesnapshotcontents配置。	
snapshot.storage.k8s.io_volumesnapshots.yaml	K8s用户自定义资源volumesnapshots配置。	

## yaml 配置模板

在 examples/kubernetes 路径下，为 CeaStor CSI 插件安装配置需要的 yaml 配置模板。配置模板需装载节点和说明如下表所示：

表 2-4 yaml 配置模板说明

文件	说明	装载节点
clone.yaml	克隆PVC模板。	任意master节点
pod-block.yaml	Block模式PVC挂载pod模板。	
pod.yaml	Mount模式PVC挂载pod模板。	
pvc.yaml	PVC模板。	
restore.yaml	快照克隆卷PVC模板。	
secret.yaml	对接存储认证配置模板。	
storageclass.yaml	存储类配置模板。	
volumesnapshotclass.yaml	快照类配置模板。	
volumesnapshot-dynamic.yaml	快照资源创建模板。	

## CSI 镜像包

ceastor-csi-driver 镜像包解压后，可查看 CSI 组件。CSI 镜像包所包含 CSI 组件和说明如下：

表 2-5 CSI 镜像包说明

CSI 组件	说明
csi-external-provisioner	<ul style="list-style-type: none"> <li>在创建 PVC 时，调用 CSI Controller 服务在存储上创建 CBD 卷作为 PV，并将 PV 绑定至 PVC。</li> <li>在删除 PVC 时，调用 CSI Controller 服务解除 PV 至 PVC 的绑定，然后在存储上删除该 PV 对应 CBD 卷</li> </ul>
csi-external-attacher	在创建/删除Pod时，是对应的pv对节点可见，生成volumeattach资源。
csi-external-resizer	在扩容PVC时，调用CSI给PVC提供 更多的存储容量空间。
csi-external-snapshotter	在创建/删除VolumeSnapshot时，调用CSI在存储侧完成快照的创建和 删除。
csi-snapshot-controller	在创建/删除VolumeSnapshot时， 监听Kubernetes API中关于 VolumeSnapshot和 VolumeSnapshotContent的对象， 并触发csi-snapshotter 在存储上完成 快照的创建。
csi-node-driver-registrar	用于获取CSI信息，并通过kubelet的 插件注册机制将节点注册到kubelet 中，从而Kubernetes能够感知该节点与CeaStor存储系统的对接。



说明

更多 CSI 组件说明，请参见 [Kubernetes CSI Developer Documentation](https://kubernetes.io/docs/concepts/storage/csi-developer-documentation/)。

## 2.2.2 获取 CSI 组件包

访问[samba.cestc.cn\CeaStor](http://samba.cestc.cn/CeaStor) 版本发布\CeaStor3.2.1 路径，获取 CeaSotr CSI 组件包和 MD5 校验文件。samba 服务器访问用户名和密码，请联系技术服务获取。

表 2-6 准备软件包

软件包名称	说明
CeaStor-CSI-3.2.1-{xxxx}-{yyyyMMdd}.tar.gz	CeaStor CSI组件包。

## 2.2.3 上传 CSI 组件包

### 前提条件

已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。

### 操作步骤

- (1) 通过管理 IP 地址，登录 Kubernetes 集群任意 master 节点。
- (2) 上传 CSI 组件包。  
将 CSI 组件包上传到需使用 CSI 插件节点任意目录下。
- (3) 上传并导入 CSI 组件包到节点。

## 2.3 准备配置文件

### 2.3.1 准备 CSI 配置文件

为支持更多增值功能，在安装 CeaStor CSI 插件前，可根据配置协议的不同，分别设置 yaml 中参数。

- nvme 协议：设置 csi-nvme-controller.yaml 或 csi-nvme-node.yaml 中置参数。
- iscsi 协议：设置 csi-iscsi-controller.yaml 或 csi-iscsi-node.yaml 中置参数。

---

#### 注意

其他参数配置，勿随意修改。若需修改 yaml 文件配置，请联系技术支持。

---

表 2-7 CSI 插件启动参数说明

参数	参数说明
LogLevel	日志级别。可填1~5，默认值为4。
EnableRWXExpendOnline	是否允许RWX挂在的PVC支持在线扩容。默认取值为false。

## 2.3.2 准备 PVC 配置文件

### 2.3.2.1 通用配置参数

#### 简介

PVC 资源 yaml 配置文件准备与参数说明。

#### 参数说明

在 yaml 文件中通用配置参数说明如下：

表 2-1 公共参数说明

参数	参数说明
apiVersion	YAML描述文件所使用的Kubernetes API版本。 默认值为storage.k8s.io/v1。
kind	Kubernetes对象资源类型。
metadata.name	（必选）自定义的资源对象名称。
metadata.namespace	（必选）自定义的资源对象名称空间。
spec	Pod规格。

### 2.3.2.2 secret.yaml

#### 简介

通过 secret.yaml 文件配置 Secret 对象，用于保存敏感数据，即登录 CeaStor 存储系统的管理 IP、用户名、密码等信息。

#### 参数说明

secret.yaml 文件中参数配置说明如下：

表 2-2 Secret 配置参数说明

参数	参数说明
endpoint	（必填）存储系统的管理IP。
userName	CeaStor存储系统的访问账号。
userkey	CeaStor存储系统的账号密码。

#### 配置示例

secret.yaml 文件配置示例如下：

```
apiVersion: v1
```

```

kind: Secret
metadata:
  name: csi-cbd-secret-user
  namespace: product-storage
stringData:
  # endpoint: https://{vip}
  endpoint: https://xxx.xxx.xxx.xxx
  # userName: <ID>
  userName: admin
  # userkey: <PASSWORD>
  userkey: admin@123

```

### 2.3.2.3 storageclass.yaml

#### 简介

通过 `storageclass.yaml` 文件配置 `StorageClass` 资源对象，用于将存储资源定义为某种类别（Class），用于标记存储资源的特性和性能。


#### 参数说明

`storageclass.yaml` 文件中参数配置说明如下：

表 2-3 StorageClass 配置参数说明

参数	参数说明
provisioner	（必填）存储资源提供者，后端存储驱动。 默认值为 <code>cds.csi.nvmf.com</code> 。
PoolUuid	（必填）CeaStor 存储系统的硬盘池 UUID。
TargetName	<p>访问路径的名称。</p> <ul style="list-style-type: none"> <li>可置空，默认值为 <code>InnerCSITarget</code> 或 <code>InnerCSITarget2</code>。</li> <li>可设置 CeaStor 存储系统已有访问路径名称。</li> </ul> <p> <b>说明</b></p> <p>若 <code>Protocol</code> 取值为 <code>iscsi</code>，必须设置为 CeaStor 存储系统已有访问路径。</p> <p>若设置为 CeaStor 存储系统已有访问路径，需确保设置的访问路径至少关联 2 个网关节点。系统支持自动关联客户端和存储卷，设置的访问路径无需关联映射关系。</p>
Protocol	<p>访问路径协议类型。</p> <ul style="list-style-type: none"> <li>可置空，默认值为 <code>nvmeof</code>。</li> <li>可选取值 <code>nvmeof</code> 或 <code>iscsi</code>。</li> </ul>
RedundancyMode	<p>存储卷冗余模式。</p> <ul style="list-style-type: none"> <li>可置空，默认值为 <code>RP_3GX</code>（3 副本）。</li> <li>可配置冗余策略，请参见<a href="#">获取硬盘池支持的冗余策略</a>。</li> </ul>
CapacityMode	存储卷容量类型。



参数	参数说明
	<ul style="list-style-type: none"> <li>可置空，默认值为 <b>Thin</b>。</li> <li>可选取值 <b>Thin</b>（精简配置）或 <b>Thick</b>（厚配置）。</li> </ul>
isFlatten	<p>克隆卷是否分裂。</p> <ul style="list-style-type: none"> <li>可置空或取值非 <b>NO_Flatten</b>，以全局默认配置或 <b>pvc.yaml</b> 中 <b>isFlatten</b> 配置生效。</li> <li>可选取值 <b>NO_Flatten</b>（不分裂）。</li> </ul> <p> <b>说明</b></p> <p>全局默认配置为卷分裂。配置优先级顺序 <b>pvc.yaml</b> &gt; <b>storageclasss.yaml</b> &gt; 全局。</p>
fsType	<p>文件系统类型。</p> <ul style="list-style-type: none"> <li>可置空，默认取值为 <b>ext4</b>。</li> <li>可选取值 <b>ext4</b> 或 <b>xfs</b>。</li> </ul>
QosPolicyName	<p>（选填）QoS限流策略名称。</p> <p>可配置为CeaStor存储系统已有QoS策略。</p>
WriteType	<p>（选填）基于数据覆盖写入的频率划分卷类型。存储引擎通过识别写入类型对数据放置、垃圾回收等机制进行优化，有助于提升块存储性能。可选取值如下：</p> <ul style="list-style-type: none"> <li><b>High</b> 用于高频覆盖写入数据的卷。</li> <li><b>Standard</b> 用于非高频写入数据的卷。</li> </ul>
csi.storage.k8s.io/provisioner-secret-name	（必填）provisioner容器使用的sercet名称。
csi.storage.k8s.io/provisioner-secret-namespace	（必填）provisioner容器使用的sercet命名空间。 默认值为 <b>default</b> 。
csi.storage.k8s.io/controller-expand-secret-name	（必填）controller容器使用的sercet名称。
csi.storage.k8s.io/controller-expand-secret-namespace	（必填）controller容器使用的sercet命名空间。 默认值为 <b>default</b> 。
csi.storage.k8s.io/node-stage-secret-name	（必填）node-stage容器使用的sercet名称。
csi.storage.k8s.io/node-stage-secret-namespace	（必填）node-stage容器使用的sercet命名空间。 默认值为 <b>default</b> 。
csi.storage.k8s.io/node-publish-secret-name	（必填）挂载PVC流程中publish使用的sercet名称。
csi.storage.k8s.io/node-publish-secret-namespace	（必填）挂载PVC流程中publish使用的sercet命名空间。 默认值为 <b>default</b> 。
csi.storage.k8s.io/controller-publish-secret-name	（必填）挂载PVC流程中publish使用的sercet名称。
csi.storage.k8s.io/controller-publish-secret-namespace	（必填）挂载PVC流程中publish使用的sercet命名空间。 默认值为 <b>default</b> 。
reclaimPolicy	（必填）回收策略。默认值为 <b>Delete</b> 。

参数	参数说明
	<p>可选取值如下：</p> <ul style="list-style-type: none"> <li>• <b>Retain</b> 表示在持久卷从持久卷声明中释放后仍能保留其卷和数据内容。</li> <li>• <b>Delete</b> 表示彻底删除底层存储资源。</li> </ul>
allowVolumeExpansion	<p>是否运行卷扩容。</p> <ul style="list-style-type: none"> <li>• 默认值为 <b>true</b>。</li> <li>• 可选取值 <b>true</b> 或者 <b>false</b>。</li> </ul>
volumeBindingMode	<p>PV动态分配与绑定的时间。可选取值如下：</p> <ul style="list-style-type: none"> <li>• <b>Immediate</b> 表示一旦创建了 PVC，立即完成 PV 的动态分配与绑定。</li> <li>• <b>WaitForFirstConsumer</b> 表示将延迟 PV 的分配与绑定，直到使用该 PVC 的 Pod 被创建。</li> </ul>

## 配置示例

storageclass.yaml 文件配置示例如下：

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cds-csi-nvmf-sc-user
provisioner: cds.csi.nvmf.com
parameters:
  # 以下示例自定义访问路径和客户端组，若不设置则为内置访问路径和内置客户端组。
  TargetName: "tgt1"
  Protocal: "nvmeof"
  RedundancyMode: "RP_3GX"
  CapacityMode: "Thin"
  PoolUuid: "ed01c454-6ffb-4dcf-926a-a8644f766e1c"
  fsType: ""
  isFlatten: "false"
  csi.storage.k8s.io/provisioner-secret-name: csi-cbd-secret-user
  csi.storage.k8s.io/provisioner-secret-namespace: product-storage
  csi.storage.k8s.io/controller-expand-secret-name: csi-cbd-secret-user
  csi.storage.k8s.io/controller-expand-secret-namespace: product-storage
  csi.storage.k8s.io/node-stage-secret-name: csi-cbd-secret-user
  csi.storage.k8s.io/node-stage-secret-namespace: product-storage
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate

```

## 2.3.2.4 pvc.yaml

### 简介

通过 pvc.yaml 文件配置 PVC 对象，即动态创卷。动态创卷无需提前创建 PV，其根据 StorageClass 自动在存储后端上创建 PVC 所需要的资源，并在创建 PVC 的同时创建 PV。

### 参数说明

pvc.yaml 文件中参数配置说明如下：

表 2-4 PVC 配置参数说明

参数	参数说明
accessModes	<p>PVC访问模式，即用户应用对存储资源的访问权限，可选取值如下：</p> <ul style="list-style-type: none"><li>• ReadWriteOnce（RWO）表示读写权限，仅允许被挂载至单个节点。</li><li>• ReadOnlyMany（ROX）表示只读权限，允许被挂载至多个节点。</li><li>• ReadWriteMany（RWX）表示读写权限，允许被挂载至多个节点。仅 Block 卷模式支持 RWX 模式。</li></ul>
annotations	<p>PVC注解，用于配置卷的高级特性。可配参数项包括capacityMode、redundancyMode、qosIOType、qosMaxIOPS、qosMaxBPS、isFlatten、writeType。</p> <p> 说明</p> <p>若配置与storageclass.yaml中配置冲突，pvc.yaml中配置优先级更高。</p>
annotations.capacityMode	<p>卷容量类型。</p> <ul style="list-style-type: none"><li>• 可置空，默认值为 Thin。</li><li>• 可选取值 Thin（精简配置）或 Thick（厚配置）。</li></ul>
annotations.redundancyMode	<p>卷冗余策略。</p> <ul style="list-style-type: none"><li>• 可置空，默认值为 RP_3GX（3 副本）。</li><li>• 可配置冗余策略，请参见<a href="#">获取硬盘池支持的冗余策略</a>。</li></ul>
annotations.qosIOType	<p>卷QoS限流类型。可选rw、w、r。</p> <ul style="list-style-type: none"><li>• rw 表示读写限流。</li><li>• w 表示读限流。</li><li>• r 表示写限流。</li></ul>
annotations.qosMaxIOPS	<p>限流最大IOPS。</p> <p>取值0表示无限制。</p> <p> 说明</p> <p>仅在同时配置 qosIOType 时生效。</p>
annotations.qosMaxBPS	<p>限流最大带宽，单位为bps。</p> <p>取值0表示无限制。</p> <p> 说明</p> <p>仅在同时配置qosIOType时生效。</p>

参数	参数说明
annotations.isFlatten	<p>克隆卷是否分裂。</p> <ul style="list-style-type: none"> <li>可置空或取值非 <code>NO_Flatten</code>，以全局默认配置或 <code>storageclasss.yaml</code> 中 <code>isFlatten</code> 配置生效。</li> <li>可选取值 <code>NO_Flatten</code>（不分裂）。</li> </ul> <p> <b>说明</b></p> <p>全局默认配置为卷分裂。配置优先级顺序 <code>pvc.yaml</code> &gt; <code>storageclasss.yaml</code> &gt; 全局。</p>
annotations.writeType	<p>（选填）基于数据覆盖写入的频率划分卷类型。存储引擎通过识别写入类型对数据放置、垃圾回收等机制进行优化，有助于提升块存储性能。可选取值如下：</p> <ul style="list-style-type: none"> <li><code>High</code> 用于高频覆盖写入数据的卷。</li> <li><code>Standard</code> 用于非高频写入数据的卷。</li> </ul>
storageClassName	StorageClass资源对象名称。
volumeMode	<p>卷模式，支持两种卷模式。</p> <ul style="list-style-type: none"> <li>可置空，默认值为 <code>Filesystem</code>。</li> <li><code>Filesystem</code> 表示卷会被 Pod 挂载到某个目录。如果卷的存储来自某块设备而该设备目前为空，Kubernetes 会在第一次挂载卷之前在设备上创建文件系统。</li> <li><code>Block</code> 表示将卷作为原始块设备来使用。</li> </ul>
resources.requests.storage	PVC申请的持久卷容量。取值示例3Gi。

## 配置示例

pvc.yaml 文件配置示例如下：

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: product-storage
  annotations:
    capacityMode: Thin
    redundancyMode: RP_3GX
    qosIOType: rw
    qosMaxIOPS: 0
    qosMaxBPS: 0
    isFlatten:
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: cds-csi-nvmf-sc
  resources:
    requests:
      storage: 1Gi

```

## 2.4 准备相关工具

可参考下表准备需要的工具软件。

表 2-5 准备软件工具

名称	说明
SSH连接工具	用于SSH方式登录节点，执行配置命令。
WinSCP	用于Windows与Linux系统之间的文件传输工具。您可以在 <a href="#">WinSCP官网</a> ，下载相应版本工具软件。 建议使用5.7.5或更高版本，并在传输文件时选用SCP协议。

# 3 安装 CSI

## 3.1 安装并配置CSI

### 简介

为达到 CeaStor 存储系统向 Kubernetes 提供持久化卷存储能力的目的，您需在 Kubernetes 集群的 master 和 worker 节点分别安装 CeaStor CSI 镜像，并需部署 CeaStor 预置 yaml 文件。

### 前提条件

- 已确认 Kubernetes 为 1.24 及其以上版本。
- 已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。
- 已获取 CeaStor CSI 插件组件包，请联系技术支持获取。
- 已为 Kubernetes 集群需使用 CSI 的节点规划并分配 master 或 worker 角色。详细操作，请参见[分配 K8s 集群节点角色](#)。

### 操作步骤

- (1) 通过管理 IP 地址，登录（可以使用 kubectl 等工具）Kubernetes 集群任意 master 节点。
- (2) 上传 CSI 组件包到。

将 CSI 组件包上传到 master 节点任意目录下。

- (3) 上传并导入 CSI 组件包到节点。
  - a. 在 CSI 组件包路径，进入 images/xxx 目录（xxx 表示镜像类型）。
  - b. 装载 CSI 组件 tar 包。详细操作说明，请参考[装载 CSI 镜像包](#)。

- (4) 创建命名空间。

执行 **kubectl create namespace <ns>** 命令，创建名为 ccos-cluster-storage-operator 和 product-storage 的命名空间。

---

#### 说明

ccos-cluster-storage-operator 和 product-storage 为 CeaStor CSI 指定命名空间，必须创建且不支持修改。若有疑问，请联系技术支持。

---

```
$ kubectl create namespace ccos-cluster-storage-operator
$ kubectl create namespace product-storage
```

- (5) 部署预置 yaml 文件。
  - a. 在 CSI 组件包路径，进入 deploy/kubernetes 目录。
  - b. 执行 **kubectl apply -f <yaml\_name>.yaml** 命令，并根据配置协议类型，依次部署 [2.2.1 yaml 配置文件](#)中预置 yaml 文件。



说明

CeaStor CSI 提供预置 `yaml` 文件，请勿随意修改。若需修改 `yaml` 文件配置，请联系技术支持。

以部署 `csi-nvmf-driver.yaml` 文件示例如下：

```
$ kubectl apply -f csi-nvmf-driver.yaml
csidriver.storage.k8s.io/cds.csi.nvmf.com created
```

- c. 执行命令查询 `yaml` 文件部署结果。详细部署成功示例，请参见[查询 CSI 预置 yaml 文件部署结果](#)。

(6) 检查 CSI 服务是否启动。

- a. 执行 `kubectl get csidriver` 命令，查看 CSI Driver。回显如下信息，则 CSI Driver 部署成功。

```
$ kubectl get csidriver
NAME                                ATTACHREQUIRED  PODINFOONMOUNT  STORAGECAPACITY
TOKENREQUESTS  REQUIRESREUBLISH  MODES          AGE
cds.csi.nvmf.com  true              true            false            <unset>
false           Persistent        5d23h
```

- b. 执行 `kubectl get pod -A | grep csi` 命令，查看 Pod 状态。全部 Pod 状态回显 `Running`，则 CSI 服务成功启动。

```
$ kubectl get pod -A | grep csi
NAMESPACE                                NAME                                READY
STATUS  RESTARTS  AGE
ccos-cluster-storage-operator  csi-snapshot-controller-684774fcbb-58hn2  1/1
Running      0      172m
product-storage                cds-csi-nvmf-lqlxg                        6/6
Running      0      174m
product-storage                csi-nvmf-node-w47fc                       2/2
Running      0      170m
product-storage                csi-nvmf-node-wz64s                       2/2
Terminating  0      3d3h
```

其中，`csi-snapshot-controller-xxxx` 包含 1 个容器，`cds-csi-nvmf-xxxx` 包含 6 个容器，`csi-nvmf-node-xxx` 包含 2 个容器，各 Pod 下容器及其说明如下：

表 3-2 CSI 服务 Pod 及其容器说明

Pod	Container	描述
csi-snapshot-controller-xxxx	csi-snapshot-controller	负责快照的管控，部署在master上。
cds-csi-nvmf-xxxx cds-csi-iscsi-xxxx	node-registrar	插件注册。插件启动的节点均需要启动。
	csi-provisioner	负责管理PV的创建和删除，部署在master上。
	csi-attacher	负责将PV发布出去，部署在master上。

Pod	Container	描述
	csi-resizer	负责管理PV的扩容，部署在master上。
	csi-snapshotter	负责管理PV的快照创，部署在master上。
	csi-nvmf-plugin	CSI驱动。所有要支持CSI挂载的节点均需要启动。
csi-nvmf-node-xxx csi-iscsi-node-xxx	node-registrar	用于获取CSI信息，可通过kubectl插件注册机制将节点注册到kubectl中。插件启动的节点均需要启动。
	csi-nvmf-plugin	后端NVMe或iSCSI盘的插件驱动。所有要支持CSI挂载的节点均需要启动。

## 3.2 配置并部署PVC

### 简介

在 Kubernetes 集群安装 CeaStor CSI 插件成功后，通过配置并部署资源文件对接 CeaStor 存储系统，并动态创建 PV。在处理存储业务的过程中，需要将容器中的数据持久化到硬盘上，以便于容器被重建或者分配到新节点时，能够继续使用持久化的升级。

- 通过 secret.yaml 文件配置 Secret 对象，用于保存敏感数据，即登录 CeaStor 存储系统的管理 IP、用户名、密码等信息。
- 通过 storageclass.yaml 文件配置 StorageClass 资源对象，用于将存储资源定义为某种类别（Class），用于标记存储资源的特性和性能。
- 通过 pvc.yaml 文件配置 PVC 对象，即动态创卷。动态创卷无需提前创建 PV，其根据 StorageClass 自动在存储后端上创建 PVC 所需要的资源，并在创建 PVC 的同时创建 PV。

### 前提条件

- 已在 Kubernetes 集群安装 CSI 插件。
- 已完成 CeaStor 存储系统与 Kubernetes 集群的连通性配置。
- 已在 Kubernetes 集群安装 Kubectl 工具。
- 已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。

### 操作步骤

- (1) 通过管理 IP 地址，登录（可以使用 Kubectl 等工具）Kubernetes 集群任意 master 节点。
- (2) 自定义创建 PVC 配置 yaml 文件。

在 CSI 组件包的 example/kubernetes 目录下，执行 `vi <yaml_name>.yaml` 命令，设置 <yaml\_name>.yaml 文件。

参考[准备 yaml 配置文件](#)，依次编辑 secret.yaml、storageclass.yaml、pvc.yaml 文件。

- (3) 部署并查询 Secret 对象。

- a. 执行 `kubectl apply -f secret.yaml` 命令，创建 Secret 对象。

```
$ kubectl apply -f secret.yaml
secret/csi-cbd-secret-user created
```



- b. 执行 **kubectl get Secret csi-cbd-secret-user** 命令，查看 Secret 对象是否创建成功。回显如下信息，则创建成功。

```
$ kubectl get Secret csi-cbd-secret-user
NAME                                TYPE      DATA   AGE
csi-cbd-secret-user                Opaque    3       46h
```

- (4) 部署并查询 StorageClass 对象。

- a. 执行 **kubectl apply -f storageclass.yaml** 命令，创建 StorageClass 对象。

```
$ kubectl apply -f storageclass.yaml
storageclass.storage.k8s.io/cds-csi-nvmf-sc created
```

- b. 执行 **kubectl get sc -A | grep csi** 命令，查看 StorageClass 对象是否创建成功。回显如下信息，则创建成功。

```
$ kubectl get sc -A | grep csi
cds-csi-nvmf-sc-user1    cds.csi.nvmf.com    Delete    Immediate    true    24h
```

- (5) 创建并查询 PVC 对象。

- a. 执行 **kubectl apply -f pvc.yaml** 命令，创建 PVC 对象。

```
$ kubectl apply -f pvc.yaml
persistentvolumeclaim/pvc-test
```

- b. 执行 **kubectl get pvc -A** 命令，查看 PVC 对象是否创建成功。回显如下信息，则创建 PVC 对象成功。

```
$ kubectl get pvc -A
NAMESPACE      NAME      STATUS      VOLUME      CAPACITY      ACCESS MODES
STORAGECLASS    AGE
product-storage pvc-test  Bound       pvc-xxx      1Gi           RWO
cds-csi-nvmf-sc 47m
```

## 后续操作

PVC 对象创建成功后，即动态创卷配置成功，即可使用 CeaStor 存储资源。CeaStor CSI 还支持克隆 PVC、扩容 PVC、创建快照、克隆卷快照等特性。详细配置说明，请参见[使用 CSI](#)。

## 3.3 启用 NVMe/iSCSI 服务

### 3.3.1 配置 NVMe 启动器

#### 简介

在 Kubernetes 集群，为需使用 CSI 的所有节点加载 NVMe 启动器。

#### 前提条件

- 已确认 Kubernetes 为 1.24 及其以上版本。
- 已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。

- 已在 Kubernetes 集群[安装 nvme-cli](#) 工具。

## 操作步骤

- (1) 通过管理 IP 地址，登录（可以使用 kubectl 等工具）Kubernetes 集群任意 master 节点。
- (2) 执行如下命令，加载内核 NVMe TCP 和 NVMe RDMA 驱动模块。

- 若为 NVMe TCP 类型，执行 **modprobe nvme-tcp** 命令。
- 若为 NVMe RDMA 类型，执行 **modprobe nvme-rdma** 命令。

- (3) 执行 **lsmod | grep nvme** 命令，查看驱动是否加载成功。回显示例如下：

```
$ lsmod | grep nvme
nvme_rdma          32768  0
nvme_tcp           32768  0
nvme_fabrics       24576  2 nvme_tcp,nvme_rdma
nvme_core          114688  3 nvme_tcp,nvme_rdma,nvme_fabrics
```

- (4) 执行如下命令，将 NVMe 驱动配置写入系统 **modules.conf** 配置文件中，持久化修改（在系统重启后可自动加载 NVMe 驱动）。

```
$ echo "nvme-tcp" >> /etc/modules-load.d/modules.conf
$ echo "nvme-rdma" >> /etc/modules-load.d/modules.conf
$ cat /etc/modules-load.d/modules.conf
nvme-tcp
nvme-rdma
```

## 3.3.2 配置 iSCSI 启动器

### 简介

在 Kubernetes 集群，为需使用 CSI 的所有节点加载 iSCSI 启动器。

### 前提条件

- 已确认 Kubernetes 为 1.24 及其以上版本。
- 已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。

## 操作步骤

- (1) 登录 Linux 客户端。
- (2) 执行 **rpm -q iscsi-initiator-utils** 命令，查询 iSCSI 客户端软件包是否已安装。

```
$ rpm -q iscsi-initiator-utils
iscsi-initiator-utils-6.2.1.4-2.git2a8f9d8.cl9.x86_64
```

- (3) 执行 **systemctl start iscsid.service** 命令，开启 iSCSI 服务。
- (4) 执行 **cat /etc/iscsi/initiatorname.iscsi** 命令，查询客户端 IQN。

```
$ cat /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.1994-05.com.redhat:a72fffd3726
```

## 3.4 启用多路径

### 3.4.1 启用多路径访问（NVMe-oF）

#### 简介

若在 Kubernetes 节点开启内核中的多路径配置，节点即可选择最优路径访问存储卷，可提升 I/O 访问速率并增加可靠性。

本小节以 Centos8.4 版本的操作系统为例，介绍如何在 Kubernetes 集群配置 NVMe-oF 多路径访问。

#### 前提条件

- 已配置 NVMe-oF 访问类型并配置了多个存储节点。
- 已获取具备 Kubernetes 集群管理员权限的账号和密码。

#### 操作步骤

- (1) 通过管理 IP 地址，登录（可以使用 Kubectl 等工具）Kubernetes 集群任意 master 节点。
- (2) 执行命令 `cat /sys/module/nvme_core/parameters/multipath`，检查内核中是否启用了原生 NVMe 多路径。

○若回显 N，表示禁用原生 NVMe 多路径，则执行[\(3\)](#)。

○若回显 Y，表示已启用原生 NVMe 多路径。

- (3) 启用原生 NVMe 多路径。

a. 配置 `/etc/modprobe.d/nvme_core.conf` 文件，添加 `options nvme_core multipath=Y` 字段。

b. 依次执行如下命令，备份 `initramfs` 文件系统。

```
$ cp /boot/initramfs-$(uname -r).img \  
$ cp /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
```

c. 执行如下命令，重建 `initramfs` 文件系统。

```
$ dracut --force -verbose
```

d. 重启系统。

- (4) （可选）在运行的系统中，可执行如下命令，更改 NVMe 设备中的 I/O 策略，以便在所有可用路径中分发 I/O。

```
$ echo "round-robin" > /sys/class/nvme-subsystem/nvme-subsys0/iopolicy
```

I/O 策略默认是“numa”模式，各模式的含义如下：

- numa 模式：IO 下发默认走一个网关，当前工作网关出现故障时，自动切换到其他网关。
- round-robin 模式：IO 下发使用轮询方式，均匀的分布到各个网关上。

## 3.4.2 启用多路径访问 (iSCSI)

### 简介

若在 **Kubernetes** 节点开启内核中的多路径配置，节点即可选择最优路径访问存储卷，可提升 I/O 访问速率并增加可靠性。

本小节主要介绍如何在 **Kubernetes** 集群配置 iSCSI 多路径访问。

### 前提条件

- 已配置 iSCSI 访问类型并配置了多个存储节点。
- 已获取具备 **Kubernetes** 集群管理员权限的账号和密码。

### 操作步骤

(1) 通过管理 IP 地址，登录（可以使用 **Kubect**l 等工具）**Kubernetes** 集群任意 **master** 节点。

(2) 检查集群是否安装多路径安装包 **multipath**。

```
$ rpm -qa | grep device-mapper-multipath
```

(3) 复制文件 **multipath.conf** 文件到 **/etc** 目录。

```
$ cp /usr/share/doc/device-mapper-multipath/multipath.conf /etc/multipath.conf
```

(4) 启用多路径。

```
$ systemctl start multipathd
```

(5) 配置多路径随系统启动。

```
$ systemctl enable multipathd.service
```

(6) 依次执行如下命令，生效配置。

```
$ multipath --F
```

```
$ systemctl restart multipath.service
```

```
$ systemctl reload multipathd.service
```

```
$ multipath -v3
```

(7) 执行 **multipath -ll** 命令，确认配置是否生效。

# 4 使用 CSI

## 4.1 管理PVC

### 4.1.1 动态创建卷（PVC）

#### 简介

动态创建卷不需要事先创建 PV，其根据 StorageClass 自动在存储后端上创建 PVC 所需要的资源，并且可以在创建 PVC 时同时创建 PV。

使用 PVC 需要用到 StorageClass 资源对象，它将存储资源定义为某种类别（Class），用于标记存储资源的特性和性能。同时也要配置 Secret 对象，用于保存敏感数据，即登录 CeaStor 后端存储的用户名、密码等信息。

#### 操作步骤

动态创建卷分为以下步骤：

- (1) 配置 Secret
- (2) 配置 StorageClass
- (3) 配置 PVC

参考[准备 yaml 配置文件](#)，依次创建和部署 secret.yaml、storageclass.yaml、pvc.yaml 文件。

### 4.1.2 克隆卷（PVC）

#### 简介

在 Kubernetes 中，克隆 PVC 即创建克隆卷。

#### 配置说明

在 Kubernetes 通过配置并执行 clone.yaml 文件，克隆存储卷。clone.yaml 文件中参数配置说明如下：

表 4-1 clone.yaml 文件参数说明

参数	参数说明
storageClassName	使用的StorageClass资源对象名称。 与源PVC的storageClassName值相同。
dataSource.name	（必填）被克隆的源PVC名称。
dataSource.kind	被克隆的源PVC所属类别。 默认值为 <b>PersistentVolumeClaim</b> 。
accessModes	克隆卷访问模式。可选取值如下：

参数	参数说明
	<ul style="list-style-type: none"> <li>ReadWriteOnce（RWO）表示读写权限，仅允许被挂载至单个节点。</li> <li>ReadOnlyMany（ROX）表示只读权限，允许被挂载至多个节点。</li> <li>ReadWriteMany（RWX）表示读写权限，允许被挂载至多个节点。仅 Block 卷模式支持 RWX 访问。</li> </ul>
resources.requests.storage	克隆卷容量，取值示例1Gi。 需大于等于源PVC卷容量。

## 配置示例

clone.yaml 文件配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: clone-example
  namespace: product-storage
spec:
  accessModes:
    - ReadOnlyMany
dataSource:
  name: pvc-example
  kind: PersistentVolumeClaim
  storageClassName: cds-csi-nvmf-sc
resources:
  requests:
    storage: 1Gi
```

## 执行并查询克隆卷

```
$ kubectl apply -f clone.yaml
persistentvolumeclaim/clone-example created
$ kubectl get pvc -A
NAMESPACE      NAME      STATUS      VOLUME      CAPACITY      ACCESS MODES
STORAGECLASS    AGE
product-storage clone1    Bound        pvc-xxx      1Gi           RWO
cds-csi-nvmf-sc 48m
```

## 4.1.3 删除卷（PVC）

### 简介

在 Kubernetes 中，根据 StorageClass 自动在存储后端上创建 PVC 所需要的资源，并且可以创建 PVC 时同时创建 PV。其中 StorageClass 中的 reclaimPolicy 可以配置 Delete（删除）和 Retain（保持）。两个不同的回收策略在删除 PVC 时有一定区别。

## 操作步骤

- (1) 查询源 PVC 回收策略。以 PVC 名为 `pvc-test-01/pvc-test02` 的卷为例。

```
$ kubectl get pv | grep pvc-test
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY
STATUS CLAIM			
STORAGECLASS REASON AGE			
pvc-a63fe993-1541-487e-b76b-07dbc519006c 10Mi		RWO	Retain
Bound product-storage/pvc-test-02			
sc-example02 22s			
pvc-baa969c7-48e8-47ab-804b-cb7350892dea 10Mi		RWO	Delete
Bound product-storage/pvc-test-01			
sc-example 2m30s			

- (2) 删除 PVC。

- (3) 如果 Reclaim Policy 为 Delete。

```
$ kubectl delete -f pvc-test-01.yaml
persistentvolumeclaim "pvc-test-01" deleted
```

- 如果 Reclaim Policy 为 Retain。

```
$ kubectl delete -f pvc-test-02.yaml
persistentvolumeclaim "pvc-test-02" deleted
```

```
$ kubectl get pv | grep pvc-test
```

NAMESPACE	NAME	CAPACITY	ACCESS MODES
STATUS VOLUME			
STORAGECLASS AGE			
pvc-a63fe993-1541-487e-b76b-07dbc519006c 10Mi		RWO	Retain
Released product-storage/pvc-test-02			
sc-example02 8m22s			

```
$ kubectl delete pv pvc-a63fe993-1541-487e-b76b-07dbc519006c
persistentvolume "pvc-a63fe993-1541-487e-b76b-07dbc519006c" deleted
```

## 4.1.4 扩容卷（PVC）

### 简介

在 Kubernetes 中，扩容 PVC 即扩容存储卷。

### 注意事项

- 仅支持扩容不支持缩容。
- ROX 和 RWX 访问模式不支持在线扩容。

### 操作步骤

- (1) 查询源 PVC。以扩容名为 `pvc-test` 的卷为例。

```
$ kubectl get pvc -A | grep pvc-test
```

NAMESPACE	NAME	STATUS	VOLUME
CAPACITY	ACCESS MODES	STORAGECLASS	AGE
product-storage	pvc-test	Bound	pvc-2171141e-9b48-45af-a31a-eb712d801bdb
2Gi	RWO	cds-csi-nvmf-sc-user	48s

(2) 扩容 PVC。以从 2Gi 扩容至 3Gi 为例。

```
$ kubectl patch pvc pvc-test -p
'{"spec":{"resources":{"requests":{"storage":"3Gi"}}}}' -n product-storage
persistentvolumeclaim/pvc-test patched
$ kubectl get pvc -A | grep pvc-test
product-storage  pvc-test  Bound      pvc-2171141e-9b48-45af-a31a-eb712d801bdb  2Gi
RWO  cds-csi-nvmf-sc-user  3m36s
$ kubectl get pv -A | grep pvc-test
pvc-2171141e-9b48-45af-a31a-eb712d801bdb  3Gi  RWO  Delete  Bound      product-
storage/pvc-test  cds-csi-nvmf-sc-user  4m59s
```

(3) (可选) 扩容后查看 PVC 不能立即显示扩容后容量，因扩容后 PVC 需要挂载到 Pod 后才回显扩容后容量。以配置 pod.yaml 文件，并创建一个 Pod 挂载 PVC 为例。

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: product-storage
spec:
  containers:
  - name: busybox
    image: busybox
    command:
    - sleep
    - "3600"
    volumeMounts:
    - mountPath: "/busybox-v-data"
      name: busybox-v
  volumes:
  - name: busybox-v
    persistentVolumeClaim:
      claimName: pvc-test
```

Pod 成功挂载后，查询 PVC 容量，可查看容量已经更改为预期的 3Gi。

```
$ kubectl get po -A | grep busybox
product-storage  busybox  1/1  Running  0  11m
$ kubectl get pvc -A | grep pvc-test
product-storage  pvc-test  Bound      pvc-2171141e-9b48-45af-a31a-eb712d801bdb  3Gi
RWO  cds-csi-nvmf-sc-user  3m36s
```



## 4.1.5 挂载 PVC

### 简介

在 PVC 配置并部署完成后，需在使用和扩容 PVC 前，基于 `volumeMode` 的不同卷模式，分别设置 `pod.yaml` 或 `pod-block.yaml`，以设置 PVC 挂载路径，即挂载卷。

### 配置说明

`pod.yaml` 文件中参数配置说明如下：

表 4-2 `pod.yaml` 配置参数说明

参数	参数说明
<code>volumes.name</code>	卷名称。
<code>volumes.persistentVolumeClaim.claimName</code>	卷所对应的PVC名称。
<code>volumeMounts.name</code>	若 <code>volumeMode</code> 值为 <b>Filesystem</b> 。 配置待挂载卷名称，与 <code>volumes.name</code> 相同。
<code>volumeMounts.mountPath</code>	若 <code>volumeMode</code> 值为 <b>Filesystem</b> 。 配置待挂载卷在Pod中的挂载路径。

`pod-block.yaml` 文件中参数配置说明如下：

表 4-3 `pod-block.yaml` 配置参数说明

参数	参数说明
<code>volumes.name</code>	卷名称。
<code>volumes.persistentVolumeClaim.claimName</code>	卷所对应的PVC名称。
<code>volumeDevices.name</code>	若 <code>volumeMode</code> 值为 <b>Block</b> 。 配置块设备名称，与 <code>volumes.name</code> 相同
<code>volumeDevices. devicePath</code>	若 <code>volumeMode</code> 值为 <b>Block</b> 。 配置块设备在Pod中的路径。

### 配置示例

若 PVC 的 `volumeMode` 为默认值 `Filesystem`，则容器中添加 PVC 时，要设置卷挂载路径。

`pod.yaml` 文件配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: product-storage
spec:
  containers:
    - name: busybox
```

```

image: busybox
command:
  - sleep
  - "3600"
volumeMounts:
  - name: busybox-v
    mountPath: "/busybox-v-data"
volumes:
  - name: busybox-v
    persistentVolumeClaim:
      claimName: pvc-test

```

若 PVC 的 `volumeMode` 为 `Block`，则容器中添加 PVC 时，要设置块设备路径。

`pod-block.yaml` 文件配置示例如下：

```

apiVersion: v1
kind: Pod
metadata:
  name: busybox-block
  namespace: product-storage
spec:
  containers:
    - name: busybox
      image: busybox:latest
      command:
        - sleep
        - "3600"
      volumeDevices:
        - name: busybox-v
          devicePath: "/busybox-dev"
  volumes:
    - name: busybox-v
      persistentVolumeClaim:
        claimName: pvc-block

```

## 部署并查看部署结果

```
$ kubectl apply -f pod.yaml
```

```
pod/busybox created
```

```
$ kubectl get pod -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
product-storage	busybox	1/1	Running	0	33s

## 4.1.6 卸载 PVC

### 简介

在 Kubernetes 中，解除 PVC 和 Pod 的绑定关系，即为卸载卷。

## 配置说明

- 若 PVC 的 volumeMode 为默认值 Filesystem:  
从 pod.yaml 配置文件中删除挂载 PVC 名字相关配置，包括：volumes.name，volumes.persistentVolumeClaim.claimName，volumeMounts.name，volumeMounts.mountPath。
- 若 PVC 的 volumeMode 为 Block  
从 pod-block.yaml 配置文件中删除挂载 PVC 名字相关配置，包括 volumes.name、volumes.persistentVolumeClaim.claimName、volumeDevices.name、volumeDevices.devicePath。

## 配置示例

若 PVC 的 volumeMode 为默认值 Filesystem，则容器中删除 PVC 时，要解除卷挂载路径。

pod.yaml 文件配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test-nvmf2
  namespace: product-storage
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: localhost/nginx
          # Filesystem PVC
          volumeMounts:
            name: nginx-storage
            mountPath: /data
          volumes:
            name: nginx-storage
            persistentVolumeClaim:
              claimName: nginx-storage
```

## 执行步骤

```
$ kubectl apply -f pod.yaml
pod/nginx created
```

## 4.2 管理VolumeSnapshot

### 4.2.1 创建卷快照

#### 简介

在 Kubernetes 中，VolumeSnapshot（卷快照）是一个存储系统上卷的快照。

VolumeSnapshot 为 Kubernetes 提供了一种标准的方式在指定时间点复制卷，无需创建新卷，可用数据备份。创建 VolumeSnapshot 对象，需先创建 VolumeSnapshotClass 对象，即先通过配置并部署 volumesnapshotclass.yaml 文件，再配置并部署 volumesnapshot-dynamic.yaml 文件。

- VolumeSnapshotClass 提供一种在配置卷快照时描述存储“类”的方法，类似于 StorageClass。
- VolumeSnapshot 对象即快照对象。

#### yaml 配置参数

volumesnapshotclass.yaml 文件中参数配置说明如下：

表 4-4 volumesnapshotclass.yaml 文件参数说明

参数	参数说明
annotations. snapshot.storage.kubernetes.io/is- default-class	可以为未请求任何特定类绑定的VolumeSnapshots指定默认的VolumeSnapshotClass。 默认值为true。
driver	CSI驱动名称。 取值为csi-nvmf-driver.yaml中metadata.name参数的值。
deletionPolicy	（必填）删除策略类型。可选取值Delete或者Retain： <ul style="list-style-type: none"><li>• Delete 表示底层存储快照会和 VolumeSnapshotContent 对象一起删除。</li><li>• Retain 表示底层快照和 VolumeSnapshotContent 对象都会被保留。</li></ul>
PoolUuid	（必填）存储系统硬盘池UUID。 取值为storageclass.yaml中PoolUuid参数的值。
csi.storage.k8s.io/snapshotter- secret-name	（必填）snapshotter容器使用的secret名称。 默认值为csi-cbd-secret-user。
csi.storage.k8s.io/snapshotter- secret-namespace	（必填）snapshotter容器使用的secret名称空间。 默认值为product-storage。

volumesnapshot-dynamic.yaml 文件中参数配置说明如下：

表 4-5 volumesnapshot-dynamic.yaml 文件参数说明

参数	参数说明
volumeSnapshotClassName	（必填）使用的VolumeSnapshotClass对象名称。 默认值为csi-cds-snap。

参数	参数说明
source. persistentVolumeClaimName	(必填) 源PVC对象名称。

配置示例如下：

#### volumesnapshotclass.yaml 文件

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: snapshot1
  namespace: product-storage
spec:
  volumeSnapshotClassName: csi-cds-snap
  source:
    persistentVolumeClaimName: pvc-test
```

#### volumesnapshot-dynamic.yaml 文件

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: volumesnapshot-test
  namespace: product-storage
spec:
  source:
    persistentVolumeClaimName: pvc-test-01
  volumeSnapshotClassName: csi-cds-snap
```

## 部署并查询卷快照

部署 VolumeSnapshotClass 对象。

```
$ kubectl apply -f volumesnapshotclass.yaml
volumesnapshotclass.snapshot.storage.k8s.io/csi-cds-snap created
```

部署并查询卷快照对象。

```
$ kubectl apply -f volumesnapshot-dynamic.yaml
volumesnapshot.snapshot.storage.k8s.io/snapshot1 created
$ kubectl get VolumeSnapshot -n product-storage
```

NAME	READYTOUSE	SOURCEPVC	SOURCESNAPSHOTCONTENT	RESTORESIZE	SNAPSHOTCLASS
snapshot1	true	pvc1		1Gi	csi-cds-snap
snapshotcontent-4bfb12df-dd8e-4fe4-aaf3-3813d88e67f6			22d	22d	

## 4.2.2 克隆卷快照

### 简介

在 Kubernetes 中，快照克隆类似于克隆 PVC，通过配置并部署 `restore.yaml` 文件实现。

### 配置说明

`restore.yaml` 文件中参数配置说明如下：

表 4-6 `restore.yaml` 文件参数说明

参数	参数说明
<code>storageClassName</code>	使用的 <b>StorageClass</b> 资源对象名称。
<code>dataSource.name</code>	（必填）被克隆的快照对象名称。
<code>dataSource.kind</code>	被克隆的快照对象所属类别。 默认值为 <b>VolumeSnapshot</b> 。
<code>dataSource.apiGroup</code>	（必填）被克隆的快照对象所属 <b>apiGroup</b> 。 固定值为 <b>snapshot.storage.k8s.io</b> 。
<code>accessModes</code>	克隆卷快照访问模式。可取值如下： <ul style="list-style-type: none"><li>• <b>ReadWriteOnce (RWO)</b> 表示读写权限，仅允许被挂载至单个节点。</li><li>• <b>ReadOnlyMany (ROX)</b> 表示只读权限，允许被挂载至多个节点。</li><li>• <b>ReadWriteMany (RWX)</b> 表示读写权限，允许被挂载至多个节点。仅 <b>Block</b> 卷模式支持 <b>RWX</b> 访问。</li></ul>
<code>resources.requests.storage</code>	克隆卷快照容量，取值示例 <b>1Gi</b> 。 需大于等于源卷快照容量。

### 部署并查询克隆卷快照

```
$ kubectl apply -f clone.yaml
persistentvolumeclaim/restore-snapshot1 configured
$ kubectl get pvc -A
NAMESPACE      NAME                               STATUS    VOLUME   CAPACITY   ACCESS
MODES          STORAGECLASS      AGE
product-storage restore-snapshot1  Bound    pvc-xxx   1Gi
RWO                                cds-csi-nvmf-sc  48m
```

## 4.2.3 删除卷快照

### 简介

在 Kubernetes 中，删除快照通过创建快照的 `volumesnapshotclass.yaml` 文件实现。

### 删除卷快照

```
$ kubectl delete -f volumesnapshot-dynamic.yaml
volumesnapshot.snapshot.storage.k8s.io "volumesnapshot-test" deleted
```

```
$ kubectl get pvc -A
```

NAMESPACE	NAME		STATUS	VOLUME	CAPACITY	ACCESS
MODES	STORAGECLASS	AGE				

# 5 管理 CSI

## 5.1 升级CSI

### 简介

CeaStor CSI 插件发布新版本，增加新功能和修复一些问题。您需要在 CeaStor 存储系统升级完成之后，配套升级 CeaStor CSI 镜像，使用新功能。

### 前提条件

- 已获取 Kubernetes 集群管理 IP、具备管理员权限的账号和密码。
- 已完成 CeaStor 存储系统升级，且已获取配套版本 CeaStor CS 组件包。

### 操作步骤

- (1) 通过管理 IP 地址，登录 Kubernetes 集群任意 master 节点。
- (2) 上传 CeaStor CSI 组件包到 master 节点任意目录下。
- (3) 上传并导入 CeaStor CSI 组件包到各节点。
  - a. 在 CSI 组件包路径，进入 images/xxx 目录（xxx 表示镜像类型）。
  - b. 选择脚本方式或手动方式，装载 CSI 组件 tar 包。详细操作说明，请参见[装载 CSI 镜像包](#)。
- (4) 部署预置 yaml 文件。
  - a. 在 CSI 组件包路径，进入 deploy/kubernetes 目录。
  - b. 更新 **csi-nvmf-node.yaml** 文件中镜像的路径。
  - c. 执行 **kubecttl apply -f csi-nvmf-node.yaml** 命令，升级 CSI 插件。
- (5) 检查 CSI Pod 重启之后，完成 CeaStor CSI 插件升级。

执行 **kubecttl get pod -A | grep csi** 命令，查看 Pod 状态。全部 Pod 状态回显 Running，则 CSI 服务成功升级。

```
$ kubecttl get pod -A | grep csi
```

NAMESPACE	STATUS	RESTARTS	AGE	NAME	READY
ccos-cluster-storage-operator	Running	0	172m	csi-snapshot-controller-684774fcbb-58hn2	1/1
product-storage	Running	0	174m	cds-csi-nvmf-lqlxg	6/6
product-storage	Running	0	170m	csi-nvmf-node-w47fc	2/2
product-storage	Terminating	0	3d3h	csi-nvmf-node-wz64s	2/2



## 5.2 固化和查看CSI日志

### 简介

由于容器日志在容器重启后会丢失。为避免 CSI 使用日志丢失，可以配置 CSI 插件日志路径，固化 CSI 日志存储路径。

CSI 日志 <podname>-xxx\_<namespace>\_<container name>-xxx.log，基于其相关 Pod、Container 和命名空间命名日志名称，插件与外部组件日志路径定义相同。

### 前提条件

已获取具备 Kubernetes 集群管理员权限的账号和密码。

### 固化 CSI 日志路径

- (1) 通过管理 IP 地址，登录 Kubernetes 集群任意 master 节点。
- (2) 配置日志路径字段。
  - a. 在 CSI Plugin 对应容器 VolumeMounts 中增加日志路径字段。

```
- mountPath: /var/log/storage/csi
  name: csipluginlogpath
```

- b. 在 CSI Plugin 所在 Pod 的 Volumes 中增加日志路径字段。

```
- hostPath:
  path: /var/log/storage/csi
  type: ""
  name: csipluginlogpath
```

- (3) 在 CSI 插件所在宿主机上创建路径。

```
$ sudo mkdir -p /var/log/storage/csi
```

### 查看 CSI 日志

- 固化 CSI 日志路径配置完成后，可在指定路径查看 CSI 日志。

```
$ cd /var/log/storage/csi
$ vi <podname>-xxx_<namespace>_<container name>-xxx.log
```

- 对于日志路径未固化的场景，可基于 CSI 相关 Pod、Container 和命名空间直接查看 CSI 日志。查看 CSI 命令如下：

```
$ cd /var/log/containers
$ vi <podname>-xxx_<namespace>_<container name>-xxx.log
```

## 5.3 手动卸载CSI

### 简介

若不再使用 CSI 服务，可通过执行命令手动卸载 CeaStor CSI Driver。

## 前提条件

- 已删除通过 CSI 服务创建的全部存储资源，包括存储卷、卷快照。
- 已获取具备 Kubernetes 集群管理员权限的账号和密码。

## 操作步骤

- (1) 通过管理 IP 地址，登录（可以使用 Kubectl 等工具）Kubernetes 集群任意 master 节点。
- (2) 在 CSI 组件包路径，进入 CSI 预置 yaml 文件目录。
- (3) 执行 **kubectl delete -f <yaml\_name>.yaml** 命令，并根据配置协议类型，依次部署 [2.2.1 yaml 配置文件](#)中预置 yaml 文件。

## 5.4 查询CSI预置yaml文件部署结果

### 简介

在 CSI 预置 yaml 文件部署完成后，可通过如下命令查询是否部署成功。

表 5-2 yaml 文件部署结果命令

yaml 文件	查询部署结果命令
csi-node-rbac.yaml	kubectl get ClusterRole -A   grep csi
operand_rbac.yaml	
serviceaccount.yaml	kubectl get ServiceAccount -A   grep csi
csi-nvmf-driver.yaml	kubectl get CSIDriver cds.csi. <protocol>.com
csi-iscsi-driver.yaml	
snapshot.storage.k8s.io_volumesnapshotclasses.yaml	kubectl get CustomResourceDefinition -A   grep volumesnapshot
snapshot.storage.k8s.io_volumesnapshotcontents.yaml	
snapshot.storage.k8s.io_volumesnapshots.yaml	
csi_controller_deployment.yaml	kubectl get Deployment -n <namespace>
csi-nvmf-controller.yaml	kubectl get DaemonSet -n <namespace>
csi-iscsi-controller.yaml	
csi-nvmf-node.yaml	
csi-iscsi-node.yaml	

### kubectl get ClusterRole -A | grep csi

执行命令查询 csi-node-rbac.yaml、operand\_rbac.yaml 等文件是否安装成功，出现如下结果表明安装成功：

```
$ kubectl get ClusterRole -A | grep csi
NAME                                     AGE
ccos-csi-snapshot-controller-runner    26d
csi-nvmf-cr                             26d
```

## kubectl get ServiceAccount -A | grep csi

执行命令查询 `serviceaccount.yaml` 等文件是否安装成功，出现如下结果表明安装成功：

```
$ kubectl get ServiceAccount -A | grep csi
```

NAMESPACE	NAME	SECRETS	AGE
ccos-cluster-storage-operator	csi-snapshot-controller	1	26d
ccos-cluster-storage-operator	csi-snapshot-controller-operator	1	26d
product-storage	csi-nvmf-sa	1	26d

## kubectl get CSIDriver cds.csi.nvmf.com

执行命令查询 `csi-nvmf-driver.yaml` 等文件是否安装成功，出现如下结果表明安装成功：

```
$ kubectl get CSIDriver cds.csi.nvmf.com
```

NAME	CREATED AT
cds.csi.nvmf.com	2023-03-09T06:42:22Z

## kubectl get CustomResourceDefinition -A | grep volumesnapshot

执行命令查询 `snapshot.storage.k8s.io_volumesnapshotclasses.yaml`、`snapshot.storage.k8s.io_volumesnapshotcontents.yaml`、`snapshot.storage.k8s.io_volumesnapshots.yaml` 等文件是否部署成功，出现如下结果表明部署成功：

```
$ kubectl get CustomResourceDefinition -A | grep volumesnapshot
```

NAME	CREATED AT
volumesnapshotclasses.snapshot.storage.k8s.io	2023-03-12T07:17:13Z
volumesnapshotcontents.snapshot.storage.k8s.io	2023-03-12T07:20:53Z
volumesnapshots.snapshot.storage.k8s.io	2023-03-12T07:21:06Z

## kubectl get Deployment -n <namespace>

分别执行命令查询 `csi_controller_deployment.yaml` 等文件是否部署成功，出现如下结果表明部署成功：

```
$ kubectl get Deployment -n ccos-cluster-storage-operator
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
csi-snapshot-controller	1/1	1	1	45h

```
$ kubectl get pod -n ccos-cluster-storage-operator
```

NAME	READY	STATUS	RESTARTS	AGE
csi-snapshot-controller-64bf858cd5-9gdp8	1/1	Running	7 (44h ago)	44h

## kubectl get DaemonSet -n <namespace>

分别执行命令查询 `csi-nvmf-controller.yaml`、`csi-nvmf-node.yaml` 等文件是否部署成功，出现如下结果表明部署成功：

```
$ kubectl get DaemonSet -n product-storage
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR
cds-csi-nvmf	1	1	1	1	1	node-

```
role.kubernetes.io/master= 4d17h
```

```

csi-nvmf-node      2          2          2          2          2          node-
role.kubernetes.io/worker= 4d17h
$ kubectl get pod -n product-storage -owide
NAME                                READY    STATUS    RESTARTS   AGE    IP                                NODE
NOMINATED NODE    READINESS GATES
cds-csi-nvmf-6rmcw    6/6      Running    0           42h    10.255.138.129    master1
<none>              <none>
csi-nvmf-node-f6qtl    2/2      Running    0           42h    10.255.138.131    worker2
<none>              <none>
csi-nvmf-node-ndnfn    2/2      Running    0           42h    10.255.138.130    worker1
<none>              <none>

```

# 6 附录

## 6.1 获取硬盘池支持的冗余策略

### 前提条件

已获取具备 CeaStor 存储系统管理员权限的账号和密码。

### 操作步骤

- (1) 使用管理员权限账号，远程登录（可以通过 **putty**、**xshell** 等工具）CeaStor 任意集群节点。

```
Connecting to 10.253.219.237:22...
Connection established.
To escape to local shell,press 'Ctrl+Alt+J'.
Activate the web console with:systemctl enable --now cockpit.socket
Last login:Sun Jan  8 11:47:30 2023 from 10.32.210.52
```

- (2) 执行 **storage dmg pool redun-dump <pool name>**命令，获取指定硬盘池支持的冗余策略。

```
$ storage dmg pool redun-dump p1
Pool p1 Redun:
  RP_2
  RP_3
  EC_2P1
  EC_2P2_1
  EC_4P2_1
```

## 6.2 分配K8s集群节点角色

### 简介

若需使用 Kubernetes 集群安装 CSI 服务，为需使用 CSI 的所有节点打标签，分配 **master** 或 **worker** 角色。

### 操作命令

- (1) 执行 **kubectl label nodes <node\_name> node-role.kubernetes.io/<node\_role>=**命令，为节点打标签。
- (2) 执行 **kubectl get node --show-labels** 命令，查看节点标签状态。

### 操作示例

以名为 **master**、**node1** 和 **node2** 的节点打标签和查询示例如下：

```
$ kubectl label nodes master node-role.kubernetes.io/master=
$ kubectl label nodes node1 node-role.kubernetes.io/worker=
$ kubectl label nodes node2 node-role.kubernetes.io/worker=
```

```
$ kubectl get node --show-labels
NAME          STATUS    ROLES    AGE   VERSION   LABELS
master        Ready     control-plane,master   23d   v1.24.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=master,kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=node-role.kubernetes.io/master=
node1         Ready     worker    23d   v1.24.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=node1,kubernetes.io/os=linux,node-role.kubernetes.io/worker=
node2         Ready     worker    23d   v1.24.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=node2,kubernetes.io/os=linux,node-role.kubernetes.io/worker=
```

## 6.3 装载CSI镜像包

### 简介

在 CSI 组件包上传 Kubernetes 后，支持以手动方式装载。

### 操作说明

选择通过手动方式装载 CSI 镜像包，需分别在所有 master 节点和使用 CSI 的 worker 节点进行装载。

Kubernetes 集群需要装载节点和需安装的 CSI 镜像包，请参见 [2.2.1 中镜像包](#)。

使用以下命令装载 tar 包以提供镜像：

- 若使用 podman，执行 **podman load -i <image\_name>.tar** 命令。
- 若使用 docker，执行 **docker load -i <image\_name>.tar** 命令。
- 若使用 containerd，执行 **ctr -n k8s.io images import <image\_name>.tar** 命令。

### 查看装载结果

全部装载完成后，分别在 master 和 worker 节点执行如下命令，可分别查看镜像名称。

- 若使用 podman，执行 **podman image** 命令。
- 若使用 docker，执行 **docker images | grep v1** 命令。
- 若使用 containerd，执行 **ctr -n k8s.io image list | grep csi** 命令。

## 6.4 安装第三程序

### 6.4.1 安装 nvme-cli

#### 简介

为需使用 NVMe 服务的所有节点安装 nvme-cli 管理工具。

## 操作步骤

- (1) 使用管理员权限账号，登录任意节点。
- (2) 执行 **yum install nvme-cli** 命令，安装 nvme-cli。
- (3) 执行 **nvme -version** 命令，查看是否安装成功。

```
$nvme -version
nvme version 1.14
```

## 6.4.2 安装 open-iscsi

### 简介

为需使用 iSCSI 服务的所有节点安装 iSCSI 管理工具。

### 操作步骤

- (1) 使用管理员权限账号，登录任意节点。
- (2) 执行 **yum install -y iscsi-initiator-utils** 命令，安装 iSCSI。
- (3) 执行 **iscsiadm --version** 命令，查看是否安装成功。

```
$iscsiadm --version
iscsiadm version 6.2.1.4
```

## 6.4.3 安装 DM-Multipath

### 简介

为需使用多路径服务的所有节点安装多路径管理工具。

### 操作步骤

- (1) 使用管理员权限账号，登录任意节点。
- (2) 执行 **yum install device-mapper-multipath** 命令，安装多路径软件。
- (3) 执行 **multipath -h** 命令，查看是否安装成功。

```
$ multipath -h
multipath-tools v0.8.7 (09/08, 2021)
```