

# CeaStor CSI Plugin 3.2.2

## Usage Guidelines

---

**Confidentiality Level: Public**

Document Version: 01

Release Date: 2024-06-30


CECloud Computing Technology Co., Ltd

### **[Copyright]**

**Copyright © CECloud Computing Technology Co., Ltd 2024 All rights reserved. All rights reserved.**

The copyright of this document belongs to CECloud Computing Technology Co., Ltd Without the written permission of CECloud Computing Technology Co., Ltd no one is allowed to use any content of this document without authorization, including monitoring, copying, transmitting, displaying, mirroring, uploading, downloading, excerpting, etc. through programs or devices or in any other way.

### **[Trademark Notice]**

 中国电子云 and other trademarks of CECloud Computing Technology Co., Ltd and/or its affiliates shown in this document are the property of CECloud Computing Technology Co., Ltd and/or its affiliates. No one may use them in any form without the written permission of CECloud Computing Technology Co., Ltd and/or its affiliates, nor may you indicate to others that you have the right to display, use or otherwise deal with them. If you have any need for publicity, display or any other use, you must obtain prior written authorization from CECloud Computing Technology Co., Ltd and/or its affiliates.

Trademarks or registered trademarks of other companies appearing in this document are owned by their respective owners.

### **[Attention]**

Your purchase of products, services or features, etc. shall be subject to the agreement in the commercial contract of CECloud Computing Technology Co., Ltd All or part of the products, services or features described in this document may not be within the scope of your purchase or use. Unless otherwise agreed in the contract, CECloud Computing Technology Co., Ltd makes no representations or warranties, express or implied, with respect to the contents of this document.

The contents of this document may be updated from time to time due to product version upgrades or other reasons. This document is intended as a guide to use only, and none of the statements, information, or recommendations contained herein constitute any warranty, express or implied.

# Preface

## Brief





This document primarily provides a user guide for the CeaStor CSI plug-in, covering installation, configuration, deployment and usage, etc., to help you quickly get started with the use of CeaStor CSI plug-in.

## Target readers

This document is intended for the following readers:

- Technical Support Engineer
- Delivery Engineer
- Maintenance Engineer

## Conventions

 <b>Warning</b>	The notes following this sign require extra attention, and improper handling may cause personal injury.
 <b>Caution</b>	Reminds of the precautions to be taken during operation, improper operation may result in data loss or equipment damage. "Caution" does not involve bodily harm.
 <b>Note</b>	Provide additional explanations of key information in the body of the text as necessary. "Note" is not a safety warning message and does not relate to personal, equipment or environmental injury information.
 <b>Tips</b>	Tips, tricks for configuring, operating or using the product.

## Revision record

Document version	Release date	Revision description
01	2024-05-30	First official release.

# Contents

<b>1 Overview</b>	<b>1</b>
1.1 Introduction	1
1.2 Features	2
1.3 Compatibility	2
1.4 Key specifications	3
<b>2 Prepare for installation</b>	<b>4</b>
2.1 Check the CSI dependency environment	4
2.2 Prepare CSI component packages	5
2.2.1 Understand CSI component packages	5
2.2.2 Upload CSI component package	9
2.3 Prepare the configuration file	9
2.3.1 Prepare the CSI configuration file	9
2.3.2 Prepare PVC configuration file	10
2.4 Prepare tool	16
<b>3 Install CSI</b>	<b>17</b>
3.1 Install and configure CSI	17
3.2 Configure and deploy PVC	19
3.3 Enable NVMe/iSCSI service	21
3.3.1 Configure the NVMe initiator	21
3.3.2 Configuring the iSCSI initiator	22
3.4 Enable multipathing	22
3.4.1 Enable multipathing access (NVMe-oF)	22
3.4.2 Enable multipathing (iSCSI)	23
<b>4 Use CSI</b>	<b>25</b>
4.1 Manage PVC	25
4.1.1 Dynamic Volume Provisioning (PVC)	25
4.1.2 Clone volume(PVC)	25
4.1.3 Delete volume (PVC)	26
4.1.4 Expand volume (PVC)	27
4.1.5 Mount PVC	29
4.1.6 Remove PVC	31
4.2 Manage volume snapshot	32

4.2.1 Create volume snapshot .....	32
4.2.2 Clone volume snapshot .....	34
4.2.3 Delete volume snapshot .....	35
<b>5 Manage CSI .....</b>	<b>36</b>
5.1 Upgrade CSI .....	36
5.2 Configure and view CSI log .....	37
5.3 Uninstall CSI manually .....	38
5.4 Query deployment results of CSI pre-built yaml file .....	38
<b>6 Appendices .....</b>	<b>41</b>
6.1 Get the redundancy policies supported by the disk pool .....	41
6.2 Assign K8s cluster node role .....	41
6.3 Load CSI image package .....	42
6.4 Install third-party program .....	42
6.4.1 Install nvme-cli .....	42
6.4.2 Install open-iscsi .....	43
6.4.3 Install DM-Multipath .....	43

# 1 Overview

## 1.1 Introduction

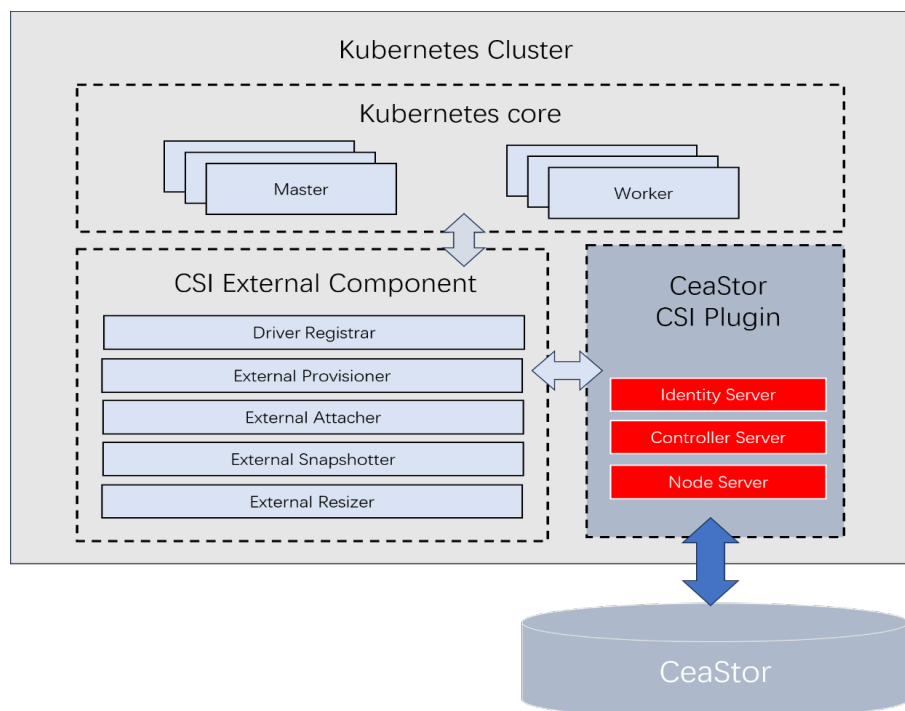
Kubernetes (or K8s), is a portable and scalable open source container orchestration management platform for managing containerized workloads and services.

CSI (Container Storage Interface) is an industry standard for exposing block storage systems to container workloads on Container Orchestration Systems (CO) such as Kubernetes. The CeaStor CSI plugin is used to provide storage services to container workloads on Kubernetes and is the required plug-in for the use of the CeaStor storage system in a Kubernetes environment.

PVC (Persistent Volume Claim) capability is provided to Kubernetes by installing the CeaStor CSI plug-in and deploying PVCs in the Kubernetes cluster to interface with CeaStor storage system resources.

Kubernetes registers to listen to Kubernetes object resources via the sidecar component and initiates calls to the CeaStor CSI plugin. The CeaStor CSI plugin implements the sidecar-initiated calls on the CeaStor storage system, e.g., initiating the Create a Persistent Volume ( Persistent Volume (PV) operation on Kubernetes is implemented on CeaStor to create a CBD volume/filesystem. The schematic diagram of how the CeaStor CSI plugin works between a Kubernetes cluster and the CeaStor storage system is shown in the figure below:

Figure 1-1 Working principle of CeaStor CSI plug-in



## 1.2 Features

The functional features supported by CeaStor CSI are listed in the following table:

Table 1-1 Supported functional features

Feature	Description	Operating limitations
Create volume	Dynamic Volume Provisioning (Create PVC )	Storage capacity 1 MiB~512 TiB
	Clone volume (Clone PVC)	Storage capacity 1 MiB~512 TiB, capacity greater than or equal to the source volume.
Delete volume	Delete volume	No subvolumes or snapshots
Expand volume	Expand volume (Expand PVC) If the PV is mounted, include the expanded filesystem.	The expansion size needs to be larger than the original volume size. <ul style="list-style-type: none"><li>• ROX does not support online expansion.</li><li>• RWX does not support online expansion by default, you need to configure the startup item field -- EnableRWXExpendOnline=true</li></ul>
Create snapshot	Create a volume snapshot	-
Delete snapshot	Delete a volume snapshot	-
Mount PVC	Mount the PVC to a Pod for using. Support both Block mode as a block device and Filesystem mode as a file system type.	<ul style="list-style-type: none"><li>• Block mode support RWO/ROX/RWX</li><li>• Filesystem mode support for RWO/ROX</li></ul>
Remove PVC	Remove PVC from Pod	-
Storage capacity information	Track used capacity	-

## 1.3 Compatibility

Container management platforms supported by the CeaStor CSI plugin, along with operating system and multipath version information.

Table 1-2 Supported container management platform

Container management platform	Version
Kubernetes	1.18~1.26

Table 1-3 Supported operating systems and versions

Operating system name	Operating system version	Native Multipath version
CCLinux x86_64 CCLinux ARM	20.09.2	Comes with OS, supports FC/iSCSI

Table 1-4 Compatibility requirements

Supported feature	Compatibility requirement
Support Mounting filesystem	Kubernetes 1.18 and above
Support for volume snapshots and cloning	Kubernetes 1.20 and above
Support for volume expansion	Kubernetes 1.24 and above

## 1.4 Key specifications

Table 1-5 CeaStor CSI plug-in key specifications

Specification	Description
Volume	Subject to the volume specifications of the CeaStor storage system, a single cluster supports the creation of up to 100,000 volumes.
Snapshot	Subject to the CeaStor Storage System Volume Snapshot specification, a single volume supports the creation of up to 256 snapshots.



# 2 Prepare for installation

## 2.1 Check the CSI dependency environment

Before installing and configuring the CeaStor CSI plug-in, check that the CSI plug-in runtime environment meets the configuration requirements.

Table 2-1 Dependency environment checklist

Matrix	Checklist	Description
Kubernetes Cluster Management Platform	The management platform is functioning properly.	Ensure that the container platform status is healthy.
	The management platform version meets the compatibility requirements.	Meet requirements in <a href="#">Table 1-4</a> .
	Host Connectivity Configuration	Ensure that network connectivity to the CeaStor storage system is normal.
Kubernetes Cluster Business Hosts	Multipath software configuration	(Optional) If multipathing needs to be enabled, make sure that all nodes that need to have the CeaStor CSI plug-in enabled have multipathing software installed.
	NVMe client software configuration	(Optional) If you choose to use the NVMe-OF protocol, make sure that the NVMe client software is installed on all nodes that need to launch the CeaStor CSI plug-in.
	iSCSI Client Software Configuration	(Optional) If you choose to use the iSCSI protocol, make sure that the iSCSI client software is installed on all nodes that need to launch the CeaStor CSI plug-in.
CeaStor Storage Systems	License authorization	<ul style="list-style-type: none"><li>Confirm that the License authorization time is within valid period.</li><li>Ensure that the Capacity to be Licensed meets the business requirements.</li></ul>
	System Cluster Operational Status	Verify that the cluster is running in a "healthy" state.
	System Host Service Status	Verify that the host service status is "running normally".
	Storage System Network Status	Verify that the host service status is "running normally".
	Hard disk pool status and usage	Verify that there is a hard disk pool for Block Service and that its status is "Normal".
	Access Path Status	Verify that the access path status is "Healthy".

## 2.2 Prepare CSI component packages

### 2.2.1 Understand CSI component packages

#### Component package name

The naming convention for CeaStor CSI component package is: CeaStor-CSI-{version number}-{version number}-{release time}.tar.gz. For example, CeaStor-CSI-3.2.2-2906-20240324003106.tar.gz.

#### Component package catalog structure

The CSI component package directory is structured as follows:

```
CeaStor-CSI-3.2.2-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

## Image package

The `images/amd64` and `images/arm64` directories contain the CeaStor CSI image packages required for the Kubernetes cluster. The image packages to be loaded, along with their respective node requirements and descriptions, are shown in the table below:

Table 2-2 Image package description

Image package name	Description	Loading node
csi-snapshot-controller	CSI snapshot controller component image	All master nodes
csi-external-snapshotter	CSI external snapshotter component image	
csi-external-resizer	CSI external resizer component image	
csi-external-provisioner	CSI external provisioner component image	
csi-external-attacher	CSI external attacher component image	
csi-snapshot-validation-webhook	(Optional) CSI snapshot validation webhook image	Any master node
csi-livenessprobe	(Optional) CSI Health Screening	
cluster-storage-operator	(Optional) Define OpenShift Container Platform cluster-wide storage defaults to ensure that a default storage class exists for OpenShift Container Platform clusters.	
cluster-csi-snapshot-controller-operator	(Optional) Define CSI Snapshot Controller deployment and configuration. The CSI Snapshot Controller is responsible for monitoring the VolumeSnapshot CRD object and	

Image package name	Description	Loading node
	managing lifecycle of volume snapshot creation and deletion.	
cds-csi-nvmf_	CeaStor CSI Plugin image package. Startup nodes such as master, need to add startup item configuration: --lsControllerServer = true	Requires a CSI worker or /master node
csi-node-driver-registrar	CSI driver registry component image	

## yaml configuration file

In the deploy/kubernetes directory, you will find the yaml configuration files for the CeaStor CSI components. The configuration files to be deployed, along with their respective node requirements and descriptions, are shown in the table below:

### Caution

CeaStor CSI provides a pre-built yaml configuration file. If you need to modify the yaml file configuration, please contact technical support.

Table 2-3 Description of yaml configuration file

File name	Description	Loading node
configmap-csi-iscsiadm.yaml	The configuration of iSCSI control command. Map iscsiadm commands in containers.	Any master node
configmap-csi-multipathd.yaml	The configuration of iSCSI multipath command. Map multipathd commands in containers.	
configmap-csi-multipath.yaml	The configuration of iSCSI multipath command. Map the multipath command in a container.	
csi_controller_deployment.yaml	The configuration of snapshots control container.	
csi-iscsi-controller.yaml	The configuration of CSI controller, including containers such as node-registrar, csi-provisioner, csi-attacher, csi-resizer, csi-snapshotter.	
csi-iscsi-driver.yaml	The configuration of CSI driver.	
csi-iscsi-node.yaml	The configuration of worker node, including containers node-registrar and csi-iscsi-plugin.	
csi-node-rbac.yaml	Authorize node driver program to operate related APIs:	
csi-nvmf-controller.yaml	The configuration of CSI controller including containers such as node-registrar, csi-provisioner, csi-attacher, csi-resizer, csi-snapshotter.	
csi-nvmf-driver.yaml	The configuration of CSI driver.	
csi-nvmf-node.yaml	The configuration of worker node, including containers like node-registrar, csi-nvmf-plugin.	
operand_rbac.yaml	Authorize snapshot driver program to operate related APIs:	

serviceaccount.yaml	The configuration of service account.	
snapshot.storage.k8s.io_volumesnapshotsclasses.yaml	The configuration of volumesnapshotclasses, the resource defined by K8s user.	
snapshot.storage.k8s.io_volumesnapshotscontents.yaml	The configuration of volumesnapshotcontents, the resource defined by K8s user.	
snapshot.storage.k8s.io_volumesnapshots.yaml	The configuration of volumesnapshot, the resource defined by K8s user.	

## yaml configuration template

In the examples/kubernetes directory, you will find the yaml configuration templates required for the installation and configuration of the CeaStor CSI plugin. The configuration templates to be deployed, along with their node requirements and descriptions, are described in the table below:

Table 2-4 Description of yaml configuration template

File	Description	Loading node
clone.yaml	A template for cloning a PVC.	Any master node
pod-block.yaml	A template for a Pod that mounts a PVC in Block mode.	
pod.yaml	A template for a Pod that mounts a PVC in Mount mode.	
pvc.yaml	A generic template for creating a Persistent Volume Claim.	
restore.yaml	A template for creating a PVC from a volume snapshot clone.	
secret.yaml	A template for configuring storage authentication credentials.	
storageclass.yaml	A template for setting up a storage class.	
volumesnapshotclass.yaml	A template for configuring the properties of a volume snapshot class.	
volumesnapshot-dynamic.yaml	A template for creating a volume snapshot resource.	

## CSI image package

After extracting the ceastor-csi-driver image package, you can view the CSI components. The CSI components contained by CSI image package are shown in the table below:

Table 2-5 Description of CSI image package

CSI component	Description
csi-external-provisioner	<ul style="list-style-type: none"> <li>When creating a PVC, invoke the CSI Controller service to create a CBD volume on the storage as a PV, and bind the PV to the PVC.</li> <li>When deleting a PVC, invoke the CSI Controller service to unbind the PV from the PVC, and then delete the corresponding CBD volume on the storage.</li> </ul>

CSI component	Description
csi-external-attacher	When creating/deleting a Pod, it is the corresponding pv that is visible to the node that generates the volumeattach resource.
csi-external-resizer	When expanding a PVC, invoke CSI to provide more storage capacity for the PVC.
csi-external-snapshotter	When creating/deleting a VolumeSnapshot, invoke CSI to complete the snapshot creation and deletion on the storage side.
csi-snapshot-controller	When creating/deleting a VolumeSnapshot, listen for changes to the VolumeSnapshot and VolumeSnapshotContent objects in the Kubernetes API, and trigger the csi-snapshotter to complete snapshot creation on the storage.
csi-node-driver-registrar	To obtain CSI information, use the kubelet's plugin registration mechanism to register the node with kubelet, allowing Kubernetes to know that the node has been connected to the CeaStor storage system.

---

**Note**

For more descriptions of CSI components , see the [Kubernetes CSI Developer Documentation](#).

---

## 2.2.2 Upload CSI component package

### Prerequisites

An account and password have been obtained for the Kubernetes cluster management IP, with administrator privileges.

### Procedure

- (1) Log in to any master node of the Kubernetes cluster via the management IP address.
- (2) Upload the CSI component package.  
Upload the CSI component package to any directory of the node that requires the CSI plug-in.
- (3) Upload and import the CSI component package to the node.

## 2.3 Prepare the configuration file

### 2.3.1 Prepare the CSI configuration file

To support more value-added features, you can configure the yaml parameters according to the specific configuration protocols in use, before installing the CeaStor CSI plugin.

- nvme protocol: set parameters in csi-nvme-controller.yaml or csi-nvme-node.yaml.
- iscsi protocol: set parameters in csi-iscsi-controller.yaml or csi-iscsi-node.yaml.

---

**! Caution**

Do not modify other parameters. If you need to modify the yaml file configuration, please contact technical support.

---

Table 2-6 CSI plugin startup parameter

Parameter	Description
LogLevel	Log Level. From 1 to 5, default value is 4.
EnableRWXExpendOnline	Determine whether PVC with RWX access support online expansion The default value is false.

## 2.3.2 Prepare PVC configuration file

### 2.3.2.1 General configuration parameter

#### About this task

Prepare the yaml configuration file and understand its parameters.

#### Parameter description

The general configuration parameters in the yaml file are described in the table below:

Table 2-1 Description of public parameters

Parameter	Description
apiVersion	The version of the Kubernetes API used by the YAML description file. The default value is <b>storage.k8s.io/v1</b> .
kind	Resource type of Kubernetes object.
metadata.name	(Required) The name of the customized resource object.
metadata.namespace	(Required) A customized resource object namespace.
spec	The specification of a Pod

### 2.3.2.2 secret.yaml

#### About this task

Configure the Secret object with the secret.yaml file to hold sensitive data, i.e., information such as administrative IP, username, and password for logging into the CeaStor storage system.

## Parameter description

The parameter configurations in the secret.yaml file are described in the table below :

Table 2-2 Description of Secret configuration parameter

Parameter	Description
endpoint	(Required) Management IP of the storage system.
userName	Account for accessing the CeaStor storage system.
userkey	The password for the CeaStor storage system.

## Configuration example

An example of configuration of the secret.yaml file is shown below:

```
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

#### About this task

The StorageClass resource object is configured through the storageclass.yaml file and is used to define the storage resource as some kind of class (Class), which is used to mark the characteristics and performance of the storage resource.

## Parameter description

The parameters configuration in the storageclass.yaml file are described in the table below:

Table 2-3 Description of StorageClass configuration parameter

Parameter	Description
provisioner	(Required)The Provider of storage resource. Backend Storage Driver. The default value is <b>cds.csi.nvmf.com</b> .
PoolUuid	(Required) The UUID of the disk pool for the CeaStor storage system.
TargetName	The name of the access path.



Parameter	Description
	<ul style="list-style-type: none"> <li>Can be left empty, and default value can be InnerCSITarget or InnerCSITarget2.</li> <li>You can set the name of an existing access path in the CeaStor storage system.</li> </ul> <p><b>Note</b></p> <p>If Protocol value is iscsi, it must be set to an existing path in the CeaStor storage system.</p> <p>If set to an existing access path in the CeaStor storage system, ensure that the access path is associated with at least two gateway nodes. The system supports automatic connection between clients and storage volumes; there is no need to set up mapping relationships for the access path.</p>
Protocol	<p>The protocol type of access path .</p> <ul style="list-style-type: none"> <li>Can be left empty, and default value is nvmeof.</li> <li>Optional values can be nvmeof or iscsi.</li> </ul>
RedundancyMode	<p>The mode of storage volume redundancy .</p> <ul style="list-style-type: none"> <li>Can be left empty and default value is <b>RP_3GX</b> (3 copies).</li> <li>Redundancy policies can be configured, see <a href="#">Get the redundancy policies supported by the disk pool</a> for the redundancy policies supported by the disk pool.</li> </ul>
CapacityMode	<p>The type of storage volume capacity.</p> <ul style="list-style-type: none"> <li>Can be left empty and default value is Thin.</li> <li>Optional values can be Thin (thin configuration) or Thick (thick configuration).</li> </ul>
isFlatten	<p>Whether the cloned volume splits or not.</p> <ul style="list-style-type: none"> <li>It can be left empty or take a value other than NO_Flatten to take effect according to the global default configuration or the isFlatten configuration in pvc.yaml.</li> <li>Optional value can be NO_Flatten (no splitting).</li> </ul> <p><b>Note</b></p> <p>The global default configuration is volume splitting. Configuration priority: pvc.yaml &gt; storageclasses.yaml &gt; global.</p>
fsType	<p>The type of filesystem.</p> <ul style="list-style-type: none"> <li>Can be left empty and the default value is ext4.</li> <li>Optional values can be ext4 or xfs.</li> </ul>
QosPolicyName	<p>(Optional)The policy name of QoS limiting.</p> <p>Can be configured as an existing QoS policy in the CeaStor storage system.</p>
WriteType	<p>(Optional) The type of volume based on the frequency of data overwrite. The storage engine optimizes mechanisms such as data placement and garbage collection by recognizing the type of write, which helps improve the performance of block storage. The optional values are as follows:</p> <ul style="list-style-type: none"> <li>High for volumes with high-frequency overwrite data.</li> <li>Standard for volumes with non-high-frequency overwrite data.</li> </ul>
csi.storage.k8s.io/provisioner-secret-name	<p>(Required) The name of the secret used by the provisioner container.</p>

Parameter	Description
csi.storage.k8s.io/provisioner-secret-namespace	(Required) The namespace of the secret used by the provider container. The default value is <b>default</b> .
csi.storage.k8s.io/controller-expand-secret-name	(Required) The name of the secret used by the controller container.
csi.storage.k8s.io/controller-expand-secret-namespace	(Required) The namespace of the secret used by the controller container. The default value is default.
csi.storage.k8s.io/node-stage-secret-name	(Required) The name of the secret used by the node-stage container.
csi.storage.k8s.io/node-stage-secret-namespace	(Required) The namespace of secret used by the node-stage container. The default value is <b>default</b> .
csi.storage.k8s.io/node-publish-secret-name	(Required) The name of the secret used in the publish step during the PVC mounting process.
csi.storage.k8s.io/node-publish-secret-namespace	(Required) The namespace of the secret used in the publish step during the PVC mounting process. The default value is default.
csi.storage.k8s.io/controller-publish-secret-name	(Required) The name of the secret used in the publish step during the PVC mounting process.
csi.storage.k8s.io/controller-publish-secret-namespace	(Required) The namespace of the secret used during the PVC mounting process in the publish step. The default value is <b>default</b> .
reclaimPolicy	<p>(Required) The reclaim policy. The default value is <b>Delete</b>. The optional values are as follows:</p> <ul style="list-style-type: none"> <li>• <b>Retain</b> indicates that a persistent volume retains its volume and data after it is released from the persistent volume statement.</li> <li>• <b>Delete</b> indicates the complete deletion of the underlying storage resource.</li> </ul>
allowVolumeExpansion	<p>Whether to allow volume expansion.</p> <ul style="list-style-type: none"> <li>• The default value is true.</li> <li>• Optional values can be true or false.</li> </ul>
volumeBindingMode	<p>The time for dynamic allocation and binding of PVs. The optional values are as follows:</p> <ul style="list-style-type: none"> <li>• <b>Immediate</b> indicates that dynamic allocation and binding of PVs is completed as soon as the PVC is created.</li> <li>• <b>WaitForFirstConsumer</b> indicates that the allocation and binding of the PV will be delayed until the Pod using this PVC is created.</li> </ul>

## Configuration example

An example of storageclass.yaml file configuration is shown below:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cds-csi-nvmf-sc-user
Provider: cds.csi.nvmf.com
parameters:
```

# The following example customizes the access path and client group, or the built-in access path and built-in client group if not set.

```
TargetName: "tgt1"
Protocol: "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

### About this task

Configure the PVC object with the pvc.yaml file for dynamic volume provisioning. Dynamic volume provisioning eliminates the need to pre-create a PV; instead, it automatically creates the necessary resources in the storage backend based on the StorageClass and simultaneously creates the PV when the PVC is created.

### Parameter description

The description of parameter configuration in the pvc.yaml file is described below:

Table 2-4 Description of PVC configuration parameter

Parameter	Description
accessModes	<p>The PVC access mode, to be more specific, the access permissions of user's applications to storage resources, optional values are as follows:</p> <ul style="list-style-type: none"><li>• ReadWriteOnce (RWO): the volume can be mounted as read-write by a single node.</li><li>• ReadOnlyMany (ROX): the volume can be mounted as read-only by many nodes.</li><li>• ReadWriteMany (RWX): the volume can be mounted as read-write by many nodes. Only Block volume mode supports RWX access.</li></ul>
annotations	<p>PVC annotations, used to configure advanced features of the volume. Configurable parameters include capacityMode, redundancyMode, qosIOType, qosMaxIOPS, qosMaxBPS, isFlatten, writeType .</p> <p><b>Note</b></p> <p>If the configuration conflicts with the configuration in storageclass.yaml, the configuration in pvc.yaml has higher priority.</p>

Parameter	Description
annotations.capacityMode	<p>Volume capacity type.</p> <ul style="list-style-type: none"> <li>Can be left empty and default value is Thin.</li> <li>Optional values can be Thin (thin configuration) or Thick (thick configuration).</li> </ul>
annotations.redundancyMode	<p>Volume Redundancy Policy.</p> <ul style="list-style-type: none"> <li>Can be left empty, and the default value is RP_3GX (3 copies).</li> <li>Redundancy policies can be configured, see <a href="#">Get the redundancy policies supported by the disk pool</a>.</li> </ul>
annotations.qosIOType	<p>Volume QoS limiting type. Optional values are rw, w, r.</p> <ul style="list-style-type: none"> <li>rw indicates read/write limiting.</li> <li>w denotes read limiting.</li> <li>r denotes write limiting.</li> </ul>
annotations.qosMaxIOPS	<p>Limit the maximum IOPS. The value 0 means no limit.</p> <p><b>Note</b></p> <p>Take effect only if qosIOType is configured.</p>
annotations.qosMaxBPS	<p>Limit the maximum bandwidth, with unit being bps. The value 0 means no limit.</p> <p><b>Note</b></p> <p>Take effect only if qosIOType is configured.</p>
annotations.isFlatten	<p>Whether the cloned volume splits or not.</p> <ul style="list-style-type: none"> <li>Can be left empty or take a value other than NO_Flatten to take effect with the global default configuration or the isFlatten configuration in storageclasss.yaml.</li> <li>Optional value NO_Flatten (no splitting).</li> </ul> <p><b>Note</b></p> <p>The global default configuration is volume splitting. Configuration priority: pvc.yaml &gt; storageclasss.yaml &gt; global.</p>
annotations.writeType	<p>(Optional) Volume types categorized based on the frequency of data overwrites. The storage engine optimizes data placement, garbage collection, and other mechanisms by recognizing the type of writes, which helps improve the performance of block storage. The optional values are as follows:</p> <ul style="list-style-type: none"> <li>High for volumes with high-frequency overwrite data.</li> <li>Standard for volumes with non-high-frequency overwrite data.</li> </ul>
storageClassName	StorageClass resource object name.
volumeMode	<p>Volume mode, two volume modes are supported.</p> <ul style="list-style-type: none"> <li>Can be left empty, and the default value is Filesystem.</li> <li>Filesystem indicates that the volume will be mounted to a directory by the Pod. If the storage for the volume comes from a device that is currently empty, Kubernetes will create a filesystem on the device before mounting the volume for the first time.</li> <li>Block indicates that the volume will be used as a raw block device.</li> </ul>

Parameter	Description
resources.requests.storage	The storage capacity of the persistent volume requested by the PVC. An example value is 3Gi.

## Configuration example

An example of pvc.yaml file configuration is shown below:

```
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 Prepare tool

Prepare the required tools and software as referenced in the table below

Table 2-5 Relevant software and tools

Name	Description
SSH Connection Tool	Used to log in to the node by SSH to execute configuration commands.
WinSCP	A tool for transferring file between Windows and Linux systems. You can find it on the <a href="#">WinSCP website</a> to download the corresponding version. It is recommended to use version 5.7.5 or higher and choose the SCP protocol when transferring files.

# 3 Install CSI

## 3.1 Install and configure CSI

### About this task

To enable CeaStor storage system to provide persistent volume storage for Kubernetes, install the CeaStor CSI image on both master and worker nodes of the Kubernetes cluster and deploy the pre-built CeaStor YAML files.

### Prerequisites

- Kubernetes has been confirmed to be version 1.24 and above.
- Cluster management IP, administrator account and password have been obtained.
- The CeaStor CSI plugin package has been obtained. Please contact technical support for obtaining it.
- Nodes of Kubernetes cluster that requires CSI have been planned and assigned master or worker roles. For details, see [Assign K8s cluster node roles](#).

### Procedure

- (1) Log in (you can use a tool such as kubectl) to any master node of the Kubernetes cluster via the management IP address.
- (2) Upload the CSI component package to .any directory on the master node.
- (3) Upload and import the CSI component package to the node.
  - a. In the CSI component package path, go to the images/xxx directory (xxx indicates the image type).
  - b. Load the CSI component tar package. For detailed instructions, see [Load CSI image package](#).
- (4) Create the namespace.

Execute the **kubectl create namespace <ns>** command to create namespaces named ccos-cluster-storage-operator and product-storage.

---

#### Note

ccos-cluster-storage-operator and product-storage are designated namespaces for CeaStor CSI. They must be created and cannot be modified. For inquiries, contact technical support.

---

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

- (5) Deploy the pre-built yaml file.
  - a. In the CSI component package path, go to the deploy/kubernetes directory.
  - b. Execute the **kubectl apply -f <yaml\_name>.yaml** command and, sequentially deploy the pre-built yaml files in the yaml configuration files in [2.2.1](#), according to the type of configuration protocols.

---

**Note**

CeaStor CSI provides pre-built yaml file, please do not modify it. If you need to modify the yaml file configuration, please contact technical support.

---

Take the example deployment of the csi-nvmf-driver.yaml file as follows:

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

- c. Execute the command to query the yaml file deployment result. For a detailed example of a successful deployment, see [Query CSI pre-built yaml file deployment results](#) for a detailed example of a successful deployment, see
- (6) Check if the CSI service is started.
  - a. Execute the **kubectl get csidriver** command to view the CSI Driver. The CSI Driver is deployed successfully if the following message is displayed.

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

- b. Execute the **kubectl get pod -A | grep csi** command to check the pod status. If the status of all pods shows Running, the CSI service has started successfully.

```
$ 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
```

csi-snapshot-controller-xxxx includes 1 container, cds-csi-nvmf-xxxx includes 6 containers, csi-nvmf-node-xxx includes 2 containers. The containers under each pod and their descriptions are described in the table below:

Table 3-1 Description of CSI Service Pods and their containers

Pod	Container	Description
csi-snapshot-controller-xxxx	csi-snapshot-controller	Responsible for Managing and controlling snapshot, deployed on the master.

Pod	Container	Description
cds-csi-nvmf-xxxx cds-csi-iscsi-xxxx	node-registrar	Responsible for plugin registration. All nodes initiated by the plugin need to be started.
	csi-provisioner	Responsible for managing the creation and deletion of PVs, deployed on master.
	csi-attacher	Responsible for posting PVs, deployed on the master .
	csi-resizer	Responsible for managing the expansion of PVs, deployed on the master.
	csi-snapshotter	Responsible for managing snapshot creation of PVs, deployed on master.
	csi-nvmf-plugin	CSI Driver. All nodes that support CSI mounting need to be started.
csi-nvmf-node-xxx csi-iscsi-node-xxx	node-registrar	Used to get CSI information, nodes can be registered into kubelet through the kubelet plugin registration mechanism. Must be initiated on all nodes where the plugin is active.
	csi-nvmf-plugin	Plugin driver for backend NVMe or iSCSI disks. All nodes that support CSI mounting need to be started.

## 3.2 Configure and deploy PVC

### About this task

After successfully installing the CeaStor CSI plug-in in the Kubernetes cluster, configure and deploy resource files to integrate with the CeaStor storage system and dynamically create PVs. To persist container data to hard disk for continued use when containers are rebuilt or rescheduled to new nodes, follow these steps:

- Use the secret.yaml file to configure a Secret object for storing sensitive data, such as the management IP, username, and password for logging into the CeaStor storage system
- Use the storageclass.yaml file to configure a StorageClass resource object, which categorizes storage resources as certain Class by their characteristics and performance
- Use the pvc.yaml file to configure a PVC object for dynamic volume provisioning. Dynamic provisioning eliminates the need to pre-create a PV; it automatically creates the necessary resources on the storage backend according to the StorageClass when the PVC is created.

### Prerequisites

- The CSI plugin has been installed on the Kubernetes cluster.
- Connectivity configuration between the CeaStor storage system and the Kubernetes cluster has been completed.
- The Kubectl tool has been installed on the Kubernetes cluster.
- An account and password have been obtained for the Kubernetes cluster management IP, with administrator privileges.



## Procedure

- (1) Log in (you can use a tool such as Kubectl) to any master node of the Kubernetes cluster via the management IP address.

- (2) Customize the creation of PVC configuration yaml files.

In the `example/kubernetes` directory of the CSI component package, execute the `vi <yaml_name>.yaml` command to set up the `<yaml_name>.yaml` file.

Refer to Prepare the configuration file to edit the `secret.yaml`, `storageclass.yaml`, and `pvc.yaml` files in turn.

- (3) Deploy and query the Secret object.

- a. Execute the `kubectl apply -f secret.yaml` command to create the Secret object.

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

- b. Execute the `kubectl get secret csi-cbd-secret-user` command to see if the Secret object was created successfully. If the following message is displayed, the secret was created successfully.

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

- (4) Deploy and query the StorageClass object.

- a. Execute the `kubectl apply -f storageclass.yaml` command to create the StorageClass object.

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

- b. Execute the `kubectl get sc -A | grep csi` command to see if the StorageClass object was created successfully. If the following message is displayed, the object was created successfully.

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

- (5) Create and query PVC objects.

- a. Execute the `kubectl apply -f pvc.yaml` command to create the PVC object.

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

- b. Execute the `kubectl get pvc -A` command to see if the PVC object was created successfully. If the following message is displayed, the PVC object was created successfully.

```
$ 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
```

## Follow-up operation

Once the PVC object is successfully created, the dynamic volume provisioning configuration is successful, and you can use CeaStor storage resources. CeaStor CSI also supports features such as PVC cloning, PVC expansion, snapshot creation, and cloning from volume snapshots. For detailed configuration instructions, see [Use CSI](#).

## 3.3 Enable NVMe/iSCSI service

### 3.3.1 Configure the NVMe initiator

#### About this task

In the Kubernetes cluster, load the NVMe initiator for all nodes that need to use CSI.

#### Prerequisites

- Kubernetes has been confirmed to be version 1.24 and above.
- Cluster management IP, administrator account and password have been obtained
- [Install nvme-cli](#) tool in the Kubernetes cluster.

#### Procedure

- (1) Log in (you can use a tool such as kubectl) to any master node of the Kubernetes cluster via the management IP address.
- (2) Execute the following commands to load the kernel NVMe TCP and NVMe RDMA driver modules.
  - If it is an NVMe TCP type, execute the **modprobe nvme-tcp** command.
  - If it is an NVMe RDMA type, execute the **modprobe nvme-rdma** command.
- (3) Execute the **lsmod | grep nvme** command to see if the driver is loaded successfully. An example is shown below:

```
$ 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) Execute the following command to write the NVMe driver configuration into the system modules.conf file, ensuring that the changes are persistent(the NVMe driver can be loaded automatically after a system reboot).

```
$ 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 Configuring the iSCSI initiator

### About this task

In a Kubernetes cluster, load the iSCSI initiator for all nodes that need to use CSI.

### Prerequisites

- Kubernetes has been confirmed to be version 1.24 and above.
- Cluster management IP, administrator account and password have been obtained

### Procedure

- (1) Log in to the Linux client.
- (2) Execute the **rpm -q iscsi-initiator-utils** command to check whether the iSCSI client package is installed.

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

- (3) Execute the **systemctl start iscsid.service** command to turn on the iSCSI service.
- (4) Execute the **cat /etc/iscsi/initiatorname.iscsi** command to identify the client IQN.

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

## 3.4 Enable multipathing

### 3.4.1 Enable multipathing access (NVMe-oF)

#### About this task

Enabling multipath configuration in the kernel on Kubernetes nodes allows node to choose the optimal path to access storage volumes, improving I/O performance and reliability.

This section, using CentOS 8.4 as an example, explains how to configure NVMe-oF multipathing in a Kubernetes cluster.

#### Prerequisites

- NVMe-oF access type has been configured and multiple storage nodes have been configured.
- The account and password of Kubernetes cluster administrator have been obtained.

#### Procedure

- (1) Log in (you can use a tool such as Kubectl) to any master node of the Kubernetes cluster via the management IP address.

- (2) Execute the command **cat /sys/module/nvme\_core/parameters/multipath** to check if native NVMe multipathing is enabled in the kernel.
  - If N is displayed, it indicate that native NVMe multipathing is disabled, execute [\(3\)](#).
  - If Y is displayed, it indicates that native NVMe multipathing is enabled.
- (3) Enable native NVMe multipathing.
  - a. Configure the **/etc/modprobe.d/nvme\_core.conf** file by adding the options **nvme\_core multipath=Y** field.
  - b. Execute the following commands in sequence to back up the initramfs filesystem.
 

```
$ cp /boot/initramfs-$(uname -r).img \
$ cp /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
```
  - c. Execute the following command to rebuild the initramfs filesystem.
 

```
$ dracut --force -verbose
```
  - d. Reboot the system.
- (4) (Optional) In the running system, you can execute the following commands to change the I/O policy of the NVMe device to distribute I/O across all available paths.
 

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

The default I/O policy is "numa" mode, and the meaning of each mode is as follows:

  - numa mode: I/O is distributed through a single gateway by default, and automatically switches to another gateway if the current working gateway fails
  - round-robin mode: I/O is distributed in a round-robin method, evenly distributed across all gateways.

## 3.4.2 Enable multipathing (iSCSI)

### About this task

Enabling multipath configuration in the kernel on Kubernetes nodes allows nodes to choose the optimal path to access storage volumes, improving I/O performance and reliability.

This section describes how to configure iSCSI multipathing in a Kubernetes cluster.

### Prerequisites

- The iSCSI access type has been configured and multiple storage nodes have been configured.
- The account and password of Kubernetes cluster administrator has been obtained.

### Procedure

- (1) Log in (you can use a tool such as Kubectl) to any master node of the Kubernetes cluster via the management IP address.
- (2) Check if the multipath package is installed in the cluster.

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

- (3) Copy the file **multipath.conf** file to the /etc directory.  

```
$ cp /usr/share/doc/device-mapper-multipath/multipath.conf /etc/multipath.conf
```
- (4) Enable multipathing.  

```
$ systemctl start multipathd
```
- (5) Configure multipath to start with the system.  

```
$ systemctl enable multipathd.service
```
- (6) Execute the following commands in sequence to activate the configuration.  

```
$ multipath --F  
$ systemctl restart multipath.service  
$ systemctl reload multipathd.service  
$ multipath -v3
```
- (7) Execute the **multipath -ll** command to verify that the configuration takes effect.

# 4 Use CSI

## 4.1 Manage PVC

### 4.1.1 Dynamic Volume Provisioning (PVC)

#### About this task

Dynamic volume provisioning eliminates the need to create PV beforehand. It automatically provisions the necessary resources for PVCs on the storage backend according to the StorageClass. it can also create a PV when PVC is created.

Using PVC requires the StorageClass resource object, which defines the storage resource as a certain Class to mark the characteristics and performance of the storage resource. It is also necessary to configure the Secret object, which is used to save sensitive data, such as usernames and passwords for logging into the CeaStor backend storage.

#### Procedure

Dynamically creating a volume is divided into the following steps:

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

Refer to [Prepare the configuration file](#) to sequentially create and deploy the secret.yaml, storageclass.yaml, and pvc.yaml files.

### 4.1.2 Clone volume(PVC)

#### About this task

In Kubernetes, cloning a PVC means creating a clone volume.

#### Configuration

In Kubernetes, configure and execute the clone.yaml file to clone storage volumes. The parameters configuration in the clone.yaml file are described in the table below:

Table 4-1 Descriptions of parameters in clone.yaml file

Parameter	Description
storageClassName	The name of the StorageClass resource object used. Must match the storageClassName value of the source PVC.
dataSource.name	(Required) The name of the source PVC to be cloned.
dataSource.kind	The category of the source PVC to be cloned. The default value is <b>PersistentVolumeClaim</b> .

Parameter	Description
accessModes	<p>Access mode of the cloned volume, with possible values:</p> <ul style="list-style-type: none"> <li>ReadWriteOnce (RWO): the volume can be mounted as read-write by a single node.</li> <li>ReadOnlyMany (ROX): the volume can be mounted as read-only by many nodes.</li> <li>ReadWriteMany (RWX): the volume can be mounted as read-write by many nodes. Only Block volume mode supports RWX access.</li> </ul>
resources.requests.storage	<p>Storage capacity of clones, example value: 1Gi.</p> <p>Must be greater than or equal to the storage capacity of source PVC.</p>

## Configuration example

An example of clone.yaml file configuration is shown below:

```
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
```

## Execute and query cloned volumes

```
$ 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 Delete volume (PVC)

#### About this task

In Kubernetes, the resources required for PVCs are automatically created on the storage backend based on the StorageClass, and PVs can be created at the same time as PVCs are created. For reclaimPolicy in the StorageClass, volume can be configured for Delete and Retain. The two reclaim policies have some differences when the PVC is deleted.

## Procedure

- (1) Query the reclaim policy of source PVC. Take the PVCs named pvc-test-01 and pvc-test02 as examples.

```
$ 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) Delete PVC.
- (3) If Reclaim Policy is Delete.

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

- o If Reclaim Policy is Retain.

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

```
$ kubectl get pv | grep pvc-test
NAMESPACE NAME STATUS VOLUME CAPACITY ACCESS MODES 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 Expand volume (PVC)

### About this task

In Kubernetes, expanding PVC means expanding storage volumes.

### Restrictions and guidelines

- Only expansion is supported not scaling down.
- ROX and RWX access modes do not support online expansion.

## Procedure

- (1) Query the source PVC. Take the volume with the expansion name pvc-test as an example.

```
$ 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) Expand PVC. Take the example of expansion from 2Gi to 3Gi.



```
$ 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) (Optional) After expansion, the PVC does not show the capacity immediately, because PVCs need to be mounted on a Pod to show the updated size. Take the example of configuring the pod.yaml file and creating a Pod to mount the PVC.

```
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
```

After the Pod is successfully mounted, query the PVC capacity to see that the capacity has been resized to the 3Gi as expected.

```
$ 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 Mount PVC

### About this task

After the PVC is configured and deployed, configure the pod.yaml or pod-block.yaml based on the different volume modes in volumeMode before using and expanding the PVC. This will determine the PVC mount path, or the mounting volume.

### Configuration

The descriptions of the configuration parameters in the pod.yaml file are described in the table below:

Table 4-2 Description of pod.yaml configuration parameters

Parameter	Description
volumes.name	The name of the volume.
volumes.persistentVolumeClaim.claimName	The name of the PVC to which the volume corresponds.
volumeMounts.name	If the value of volumeMode is <b>Filesystem</b> . Configure the name of the volume to be mounted, matching <b>volumes.name</b> .
volumeMounts.mountPath	If the value of volumeMode is <b>Filesystem</b> . Configure the mount path in the Pod for the volume to be mounted.

The description of the configuration parameter in the pod-block.yaml file are described in the table below:

Table 4-3 Descriptions of pod-block.yaml configuration parameters

Parameter	Description
volumes.name	Volume Name.
volumes.persistentVolumeClaim.claimName	The name of the PVC to which the volume corresponds.
volumeDevices.name	If the value of volumeMode is <b>Block</b> . Configure the name of block device, matching <b>volumes.name</b>
volumeDevices. devicePath	If the value of volumeMode is <b>Block</b> . Configure the path of the block device in the Pod.

### Configuration example

If the default value of PVC volumeMode is Filesystem, you must set the volume mount path when adding the PVC to the container.

An example of pod.yaml file configuration is shown below:

```
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

```

If the PVC volumeMode is Block, you must set the block device path when the PVC is added to the container.

An example of pod-block.yaml file configuration is shown below:

```

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

```

## Deploy and view deployment results

```

$ 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 Remove PVC

### About this task

In Kubernetes, unbinding a PVC from a Pod means removing the volume.

### Configuration

- If the default value of PVC volumeMode is Filesystem:  
Remove the mount PVC name-related configuration from the pod.yaml configuration file, including: `volumes.name`, `volumes.persistentVolumeClaim.claimName`, `volumeMounts.name`, and `volumeMounts.mountPath`.
- If the PVC volumeMode is Block:  
Remove the mount PVC name-related configuration from the pod-block.yaml configuration file, including `volumes.name`, `volumes.persistentVolumeClaim.claimName`, `volumeDevices.name`, and `volumeDevices.devicePath`.

### Configuration example

If the default value of PVC volumeMode is Filesystem, you must unmount the volume mount path when the PVC is deleted from the container.

An example of pod.yaml file configuration is shown below:

```
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: pvc-name
              mountPath: /data
          volumes:
            - name: pvc-name
              persistentVolumeClaim:
                claimName: pvc-name
```

## Steps

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

## 4.2 Manage volume snapshot

### 4.2.1 Create volume snapshot

#### About this task

In Kubernetes, a volume snapshot is a snapshot of a volume on a storage system.

Volume snapshots provide Kubernetes users with a standardized way to copy a volume's contents at a particular point in time without creating an entirely new volume., suitable for data backup. To create a VolumeSnapshot object, you need to create a VolumeSnapshotClass object first. To be more specific, configure and deploy the volumesnapshotclass.yaml file in the first and then configure and deploy the volumesnapshot-dynamic.yaml file.

- VolumeSnapshotClass provides a way to describe the storage "class" when configuring a volume snapshot, similar to StorageClass.
- The VolumeSnapshot object is the snapshot object.

#### yaml configuration parameters

The configuration parameters in the volumesnapshotclass.yaml file are described in the table below:

Table 4-4 Descriptions of parameters in volumesnapshotclass.yaml file

Parameter	Description
annotations. snapshot.storage.kubernetes.io/is- default-class	A default VolumeSnapshotClass can be specified for VolumeSnapshots that do not request any specific class binding. The default value is true.
driver	CSI drive name. Takes the value from the <b>metadata.name</b> parameter in csi-nvmf-driver.yaml.
deletionPolicy	(Required) Type of deletion policy. Supported values are Delete or Retain: <ul style="list-style-type: none"><li>• Delete means that the VolumeSnapshotContent and its physical snapshot on underlying storage system are deleted. Required.</li><li>• Retain means that the VolumeSnapshotContent and its physical snapshot on underlying storage system are kept</li></ul>
PoolUuid	(Required) The UUID of the storage pool Takes the value from the <b>PoolUuid</b> parameter in storageclass.yaml.
csi.storage.k8s.io/snapshotter- secret-name	(Required) The name of the secret used by the snapshotter container. The default value is <b>csi-cbd-secret-user</b> .

Parameter	Description
csi.storage.k8s.io/snapshotter-secret-namespace	(Required) The secret namespace used by the snapshotter container. The default value is <b>product-storage</b> .

The configuration parameters in the volumesnapshot-dynamic.yaml file are described in the table below:

Table 4-5 Description of parameters in the volumesnapshot-dynamic.yaml file

Parameter	Description
volumeSnapshotClassName	(Required) The name of the VolumeSnapshotClass object used. The default value is <b>csi-cds-snap</b> .
source.persistentVolumeClaimName	(Required) The object name of Source PVC.

An example of configuration is shown below:

#### Volumesnapshotclass.yaml file

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

#### The volumesnapshot-dynamic.yaml file

```
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
```

## Deploy and query volume snapshots

Deploy the VolumeSnapshotClass object.

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

Deploy and query volume snapshot objects.

```
$ kubectl apply -f volumesnapshot-dynamic.yaml
volumesnapshot.snapshot.storage.k8s.io/snapshot1 created
$ kubectl get VolumeSnapshot -n product-storage
NAME READYTOUSE SOURCEPVC SOURCESNAPSHOTCONTENT RESTORESIZING SNAPSHOTCLASS
SNAPSHOTCONTENT CREATIONTIME AGE
snapshot1 true pvc1 1Gi csi-cds-snap snapcontent-4bfb12df-dd8e-4fe4-aaf3-3813d88e67f6
22d 22d 22d
```

## 4.2.2 Clone volume snapshot

### About this task

In Kubernetes, snapshot cloning is similar to cloning PVCs and is achieved by configuring and deploying the restore.yaml file.

### Configuration

The parameter configurations in the restore.yaml file are described in the table below:

Table 4-6 Description of the parameters in the restore.yaml file

Parameter	Description
storageClassName	The name of the StorageClass resource object used.
dataSource.name	(Required) The name of the snapshot object being cloned.
dataSource.kind	The category to which the cloned snapshot object belongs. The default value is <b>VolumeSnapshot</b> .
dataSource.apiGroup	(Required) The apiGroup to which the cloned snapshot object belongs. The fixed value is snapshot.storage.k8s.io.
accessModes	Access mode of the cloned volume, with possible values:: <ul style="list-style-type: none"> <li>ReadWriteOnce (RWO): the volume can be mounted as read-write by a single node.</li> <li>ReadOnlyMany (ROX): the volume can be mounted as read-only by many nodes.</li> <li>ReadWriteMany (RWX): the volume can be mounted as read-write by many nodes. Only Block volume mode supports RWX access.</li> </ul>
resources.requests.storage	Storage Capacity of clones, example value: 1Gi. Must be greater than or equal to the storage capacity of source PVC.

### Deploy and query cloned volume snapshots

```
$ 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-nvme-sc 48m
```

## 4.2.3 Delete volume snapshot

### About this task

In Kubernetes, deleting a volume snapshot is accomplished by creating the snapshot's `volumesnapshotclass.yaml` file.

### Delete volume snapshot

```
$ 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 Manage CSI

## 5.1 Upgrade CSI

### About this task

A new version of the CeaStor CSI plug-in has been released, adding new features and fixing some issues. After upgrading the CeaStor storage system, you must also upgrade the CeaStor CSI image to take advantage of the new features.

### Prerequisites

- Cluster management IP, administrator account and password have been obtained.
- The upgrade of the CeaStor storage system has been completed and a companion version of the CeaStor CS component package has been acquired.

### Procedure

- (1) Log in to any master node of the Kubernetes cluster via the management IP address.
- (2) Upload the CeaStor CSI component package to any directory on the master node.
- (3) Upload and import the CeaStor CSI component package to the nodes.
  - a. In the CSI component package path, go to the images/xxx directory (xxx indicates the image type).
  - b. Select either the scripted or manual method to load the CSI component tar package. For detailed instructions, see [Load CSI image package](#).
- (4) Deploy the pre-built yaml file.
  - a. In the CSI component package path, go to the deploy/kubernetes directory.
  - b. Update the image path in the **csi-nvmf-node.yaml** file.
  - c. Execute the **kubectl apply -f csi-nvmf-node.yaml** command to upgrade the CSI plugin.
- (5) The upgrade of the CeaStor CSI plugin will complete after CSI Pod restarts.

Execute the **kubectl get pod -A | grep csi** command to check the pod status. If the status of all pods shows Running, the CSI service has been successfully upgraded.

```
$ 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
```

## 5.2 Configure and view CSI log

### About this task

Container logs will be lost after container restarts. To avoid loss of CSI usage logs, you can configure the CSI plugin log path to fix the CSI log storage path.

CSI logs are named based on their associated Pod, Container, and namespace, following the pattern `<podname>-xxx_<namespace>_<container name>-xxx.log`. The plugin and external components have the same log path definition.

### Prerequisites

The account and password of Kubernetes cluster administrator have been obtained.

### Configure the CSI log path

- (1) Log in to any master node of the Kubernetes cluster via the management IP address.
- (2) Configure the log path field.

- a. Add the log path field to the CSI Plugin's corresponding container VolumeMounts.

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

- b. Add the Log Path field to the Volumes of the Pod where the CSI plugin resides in.

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

- (3) Create the path on the host where the CSI plugin resides in.

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

### View CSI logs

- After the CSI log path configuration is complete, you can view the CSI logs at the specified path.

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

- For scenarios where the log path is not fixed, CSI logs can be viewed directly based on CSI-related Pods, Containers, and namespaces. The command to view CSI is as follows:

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

## 5.3 Uninstall CSI manually

### About this task

If you no longer use the CSI service, you can manually uninstall the CeaStor CSI Driver by executing the command.

### Prerequisites

- All storage resources created through the CSI service have been deleted, including storage volumes, volume snapshots.
- The account and password of Kubernetes cluster administrator has been obtained.

### Procedure

- (1) Log in (you can use a tool such as Kubectl) to any master node of the Kubernetes cluster via the management IP address.
- (2) In the CSI component package path, go to the CSI pre-built yaml file directory.
- (3) Execute the **kubectl delete -f <yaml\_name>.yaml** command and sequentially deploy pre-built yaml files in the yaml configuration files in [2.2.1](#) according to the type of configuration protocols.

## 5.4 Query deployment results of CSI pre-built yaml file

### About this task

After the CSI pre-built yaml file has been deployed, you can query whether the deployment was successful with the following commands.

Table 5-1 The commands for querying yaml file deployment results.

yaml file	Command
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

Execute the command to query whether csi-node-rbac.yaml, operand\_rbac.yaml, and other files are installed successfully, the following result indicates that the installation is successful:

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

NAME	AGE
ccos-csi-snapshot-controller-runner	26d
csi-nvmf-cr	26d

## kubectl get ServiceAccount -A | grep csi

Execute the command to query whether the serviceaccount.yaml file is installed successfully, the following result indicates that the installation is successful:

```
$ 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

Execute the command to query whether csi-nvmf-driver.yaml and other files have been installed successfully, the following result indicates that the installation is successful:

```
$ 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

Execute the command to query snapshot.storage.k8s.io\_volumesnapshotclasses.yaml, snapshot.storage.k8s.io\_volumesnapshotcontents.yaml, snapshot.storage.k8s.io\_volumesnapshots.yaml and other files are deployed successfully, the following result indicates that the deployment is successful:

```
$ 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>

Execute the command to query csi\_controller\_deployment.yaml and other files whether the deployment is successful, the following results show that the deployment is successful:

```
$ 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>

Execute the commands to query whether csi-nvmf-controller.yaml, csi-nvmf-node.yaml and other files are successfully deployed, and the following results show that the deployment is successful:

\$ kubectl get DaemonSet -n product-storage

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
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 Appendices

## 6.1 Get the redundancy policies supported by the disk pool

### Prerequisites

The administrator account and password of CeaStor storage system have been obtained

### Procedure

- (1) Log in remotely to any CeaStor cluster node using an administrator account (tools like PuTTY or Xshell can be used).

```
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) Execute the **storage dmng pool redundancy-dump** <pool name> command to obtain the redundancy policies supported by the specified disk pool.

```
$ storage dmng pool redun-dump p1
Pool p1 Redun.
  RP_2
  RP_3
  EC_2P1
  EC_2P2_1
  EC_4P2_1
```

## 6.2 Assign K8s cluster node role

### About this task

If you need to install the CSI service using a Kubernetes cluster, tag all nodes that need to use CSI and assign the master or worker role.

### Operation commands

- (1) Execute the **kubectl label nodes** <node\_name> **node-role.kubernetes.io/<node\_role>=** command to label the nodes.
- (2) Execute the **kubectl get node --show-labels** command to view the status of node labels.

### Operation example

Labeling and querying examples for nodes named 'master,' 'node1,' and 'node2' are as follows

```
$ 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 Load CSI image package

### About this task

Support for loading CSI component packages in a manual manner after they have been uploaded to Kubernetes.

If you choose to load the CSI image package manually, you need to load it on all master nodes and worker nodes that use CSI.

Kubernetes clusters need to be loaded with nodes and CSI image packages that need to be installed, see [Image Packages in 2.2.1](#).

Use the following command to load the tar package to provide the image:

- If using podman, execute the **podman load -i <image\_name>.tar** command.
- If using docker, execute the **docker load -i <image\_name>.tar** command.
- If using container, execute the **ctr -n k8s.io images import <image\_name>.tar** command.

### View loading results

After all packages are loaded, execute the following commands on both master and worker nodes to view the image names.

- If using podman, execute the **podman image** command.
- If you are using docker, execute the **docker images | grep v1** command.
- If using containerd, execute the **ctr -n k8s.io image list | grep csi** command.

## 6.4 Install third-party program

### 6.4.1 Install nvme-cli

#### About this task

Install the nvme-cli management tool for all nodes that require NVMe services.

## Procedure

- (1) Use an administrator account to log into any node.
- (2) Execute the command `yum install nvme-cli` to install `nvme-cli`.
- (3) Execute the `nvme -version` command to see if the installation was successful.

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

## 6.4.2 Install open-iscsi

### About this task

Install the iSCSI management tool for all nodes that need to use the iSCSI service.

### Procedure

- (1) Use an administrator account to log into any node.
- (2) Execute the **`yum install -y iscsi-initiator-utils`** command to install iSCSI.
- (3) Execute the **`iscsiadm --version`** command to see if the installation was successful.

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

## 6.4.3 Install DM-Multipath

### About this task

Install the multipathing management tools for all nodes that need to use the multipathing service.

### Procedure

- (1) Use an administrator account to log in to any node.
- (2) Execute the **`yum install device-mapper-multipath`** command to install the multipathing software.
- (3) Execute the **`multipath -h`** command to see if the installation was successful.

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