浪潮存储系统
K8sCSIPlugin 插件
用户手册

文档版本 **1.0**

发布日期 **2020-09-28**

适用版本 **K8sCSIPlugin_V2.1.0 及以上**

尊敬的用户：

衷心感谢您选用浪潮存储系统！浪潮存储秉承“云存智用 运筹新数据”的新存储之道，致力于为您提供符合新数据时代需求的存储产品和解决方案。

本手册用于帮助您更详细地了解和便捷地使用存储系统，涉及的截图仅为示例，最终界面请以实际设备显示的界面为准。

由于产品版本升级或其他原因，本手册内容会不定期进行更新，如有变动恕不另行通知。除非另有约定，本手册仅作为使用指导，本手册中的所有陈述、信息和建议不构成任何明示或暗示的担保。

如果您对本手册有任何疑问或建议，请向浪潮电子信息产业股份有限公司垂询。

技术服务电话： 4008600011

地　　址： 中国济南市浪潮路 1036 号

浪潮电子信息产业股份有限公司

邮　　编： 250101

使用声明

在您正式使用本存储系统之前，请先阅读以下声明。只有您阅读并且同意以下声明后，方可正式开始使用本存储系统。如果您对以下声明有任何疑问，请和您的供货商联系或直接与我们联系。如您在开始使用本系统前未就以下声明向我们提出疑问，则默认您已经同意了以下声明。

1. 请不要自行拆卸本存储系统机箱及机箱内任何硬件设备。在本存储系统出现任何硬件故障或您希望对硬件进行任何升级时，请您将机器的详细硬件配置反映给我们的客户服务中心。
2. 请不要将本存储系统的设备与任何其他型号的相应设备混用。本存储系统的内存、CPU、CPU 散热片、风扇、硬盘托架、硬盘等都是特殊规格的。
3. 在使用本存储系统时遇到任何软件问题，请您首先和相应软件的提供商联系。由提供商和我们联系，以方便我们共同沟通和解决您遇到的问题。对于数据库、网络管理软件或其他网络产品的安装、运行问题，我们尤其希望您能够这样处理。
4. 上架安装本存储系统前，请先仔细阅读相关产品手册中的快速安装指南。我们致力于产品功能和性能的持续提升，部分功能及操作与手册描述可能会有所差异，但不会影响使用。如果您有任何疑难问题，请与我们的客户服务中心联系。
5. **我们特别提醒您：在使用过程中，注意对您的数据进行必要的备份。**
6. 本存储系统为 A 级产品，在生活环境中可能会造成无线电干扰，需要您对其干扰采取切实可行的措施。
7. 请仔细阅读并遵守本手册的安全声明和安全细则。
8. 本手册中涉及的各软、硬件产品的标识、名称版权归产品的相应公司拥有。

以上声明中，“我们”指代浪潮电子信息产业股份有限公司；浪潮电子信息产业股份有限公司拥有对以上声明的最终解释权。

安全声明

我们非常重视数据安全和隐私，且一如既往地严密关注产品和解决方案的安全性，为您提供更满意的服务。在您正式使用本存储系统之前，请先阅读以下安全声明。

1. 为了保护您的数据隐私，在调整存储产品用途或淘汰存储设备时，请您将存储系统软件恢复固件出厂设置、删除信息、清除日志。 同时，建议采用第三方安全擦除工具对存储系统软件所在的系统盘进行全面安全擦除。
2. 您购买的存储产品业务运营或故障定位的过程中可能会获取或使用用户的某些个人数据（如告警邮件接收地址、IP 地址）。因此，您有义务根据所适用国家或地区的法律法规制定必要的用户隐私政策，并采取足够的措施以确保用户的个人数据受到充分的保护。
3. 如需获取存储系统开源软件声明，请直接联系浪潮客户服务人员。
4. 存储系统的某些安全特性需要您自行配置，如认证、传输加密、存储数据加密等，这些配置操作可能会对存储系统的性能和使用方便性造成一定影响。您可以根据应用环境，权衡是否进行安全特性配置。
5. 存储系统自带了部分用于生产、装备、返厂检测维修的接口、命令及定位故障的高级命令，如使用不当，可能会导致设备异常或者业务中断，不建议您自行使用。如需使用，请联系我们的客户服务人员。
6. 我们已全面建立产品安全漏洞应急和处理机制，确保第一时间处理产品安全问题。若您在存储产品使用过程中发现任何安全问题，或者寻求有关产品安全漏洞的必要支持，请直接联系我们的客户服务人员。

以上声明中，“我们”指代浪潮电子信息产业股份有限公司；浪潮电子信息产业股份有限公司拥有对以上声明的最终解释权。

安全细则

在使用本存储系统时，若操作不当，可能会危及您的人身安全。为避免发生意外，在正式使用本存储系统之前，请务必认真阅读以下安全细则，严格按照要求进行操作。

1. 本存储系统中的电源设备可能会产生高电压和危险电能，从而导致人身伤害。请勿自行卸下主机盖以拆装、更换系统内部的任何组件。除非另外得到我们的通知，否则只有经过我们培训的维修技术人员才有权拆开主机盖及拆装、更换内部组件。
2. 请将设备连接到适当的电源，仅可使用额定输入标签上指明的外部电源为设备供电。为保护您的设备免受电压瞬间升高或降低所导致的损坏，请使用相关的稳压设备或不间断电源设备。
3. 如果必须使用延长线缆，请使用配有正确接地插头的三芯线缆，并查看延长线缆的额定值，确保插入延长线缆的所有产品的额定电流总和不超过延长线缆额定电流限制的百分之八十。
4. 请务必使用随机配备的供电组件，如电源线、电源插座（如果随机配备）等。为了本存储系统及使用者的安全，切勿随意更换电源线缆或插头。
5. 为防止因系统漏电而造成电击危险，请务必将本存储系统和外围设备的电源电缆插入已正确接地的电源插座。在未安装接地导线及不确定是否已有适当接地保护的情况下，请勿操作和使用本存储系统，并及时与电工联系。
6. 切勿将任何物体塞入本存储系统的开孔处，否则，可能会导致内部组件短路而引起火灾或电击。
7. 请将本存储系统置于远离散热片和有热源的地方，切勿堵塞通风孔。
8. 切勿在高潮湿、高灰尘的环境中使用本存储系统，切勿让食物或液体散落在系统内部或其它组件上。
9. 使用错误型号的电池会有爆炸的危险，需要更换电池时，请先向制造商咨询并使用与制造商推荐型号相同或相近的电池。切勿拆开、挤压、刺戳电池或使其外部接点短路。不要将其丢入火中或水中，也不要暴露在温度超过 60 摄氏度的环境中。请勿尝试打开或维修电池，务必合理处置用完的电池，不要将用完的电池及可能包含电池的电路板及其它组件与其它废品放在一起。有关电池回收政策请与当地废品回收处理机构联系。

以上内容中，“我们”指代浪潮电子信息产业股份有限公司；浪潮电子信息产业股份有限公司拥有对以上内容的最终解释权。

目 录

1 功能描述

1.1 基本介绍

浪潮 K8sCSIPlugin 插件使得浪潮存储可以为 Kubernetes 集群中的应用自动的提供持久化存储。K8sCSIPlugin 插件功能主要包含卷的创建、删除、挂载、卸载、克隆，快照的创建、删除。

当 Kubernetes 集群中的一个服务需要使用浪潮存储提供的卷时，用户会首先在 Kubernetes 上创建一个 PVC（Persistent Volume Claim）资源信息，Kubernetes 根据 PVC 中的信息，通过接口，将消息传递给 instorage-csi，instorage-csi 收到消息后会自动的在存储上创建满足要求的卷，并在 Kubernetes 上生成一个 PV（Persistent Volume）信息，之后服务就可以使用该卷。当 Kubernetes 删除 PV 信息时，消息也将传递给 instorage-csi，instorage-csi 根据里面的具体信息，自动的将存储上与之对应的卷删除。

当 Kubernetes 集群中的一个使用持久化存储的服务启动时，首先需要将服务所使用的持久化存储映射到某个主机的指定目录上，然后服务才可以使用这些持久化存储。在这个过程中，Kubernetes 会调用与相应 PV 信息对应的卷挂载/卸载插件，K8sCSIPlugin 插件中也实现了该部分功能。当 Kubernetes 需要挂载一个卷时，instorage-csi 会自动的在存储上完成卷与主机信息的绑定，并在主机端将对应的卷扫描出来，根据需要，会自动对卷进行格式化，然后将卷上的文件系统挂载到 Kubernetes 指定的目录上，随后，Kubernetes 便可以将该目录映射到服务的容器中，供服务使用。

当前插件实现的详细功能如下：

表 1-1K8sCSIPlugin 插件实现的功能

序号	功能模块	操作
1	卷-主机操作	挂载卷到主机

2		mount 卷上的文件系统到主机
3		从主机上 unmount 卷上的文件系统
4		从主机上卸载卷
5	存储接口协议	iSCSI
6		FC
7	多路径	支持多路径设备
8	卷操作	创建卷（支持普通卷、镜像卷、双活卷）
9		删除卷
10		克隆卷（支持基于卷克隆和基于快照克隆）
11		卷在线扩容
12	快照操作	创建快照
13		删除快照

1.2 约束与限制

表 1-2 约束与限制

支持 Kubernetes 版本	1.10 及以上
支持 Linux 内核版本	4.4 及以上
支持功能	1. 卷的创建、删除。 2. 卷的挂载、卸载（支持 FC/iSCSI，支持多路径）。 3. 卷的在线扩容。（需要 Kubernetes 版本支持）。 4. 快照的创建、删除。

适用存储产品类型

AS2150G2&AS2200G2&AS2600G2&AS5300G2&AS5500G2&AS5600G2&AS5800G2&AS6800G2

HF5500

AS2600G2-F&AS5300G2-F&AS5500G2-F&AS5600G2-F&AS5800G2-F&AS6800G2-F

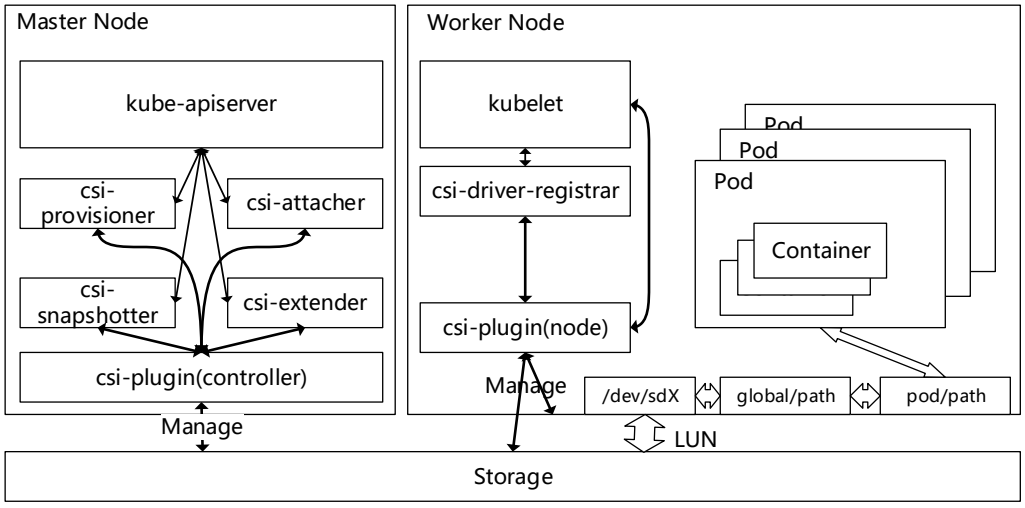
AS5300G5&AS5500G5&AS5600G5&AS5800G5&HF5000G5&HF6000G5

1.3 应用场景

通过浪潮 K8sCSIPlugin 插件，在使用 Kubernetes 时，可以直接的利用 Kubernetes 的管理命令来自动的在浪潮的 InStorage 系列存储上创建卷、删除卷、克隆卷、挂载卷、卸载卷、创建快照、删除快照，并可以将存储上的卷提供给应用服务使用，用于服务的持久化数据存储。

K8sCSIPlugin 插件，主要包括 Identity、Controller、Node 三个服务。Identity 服务负责插件信息、能力、状态的获取，Controller 服务负责卷的创建、删除、克隆和快照的创建、删除，Node 服务负责卷的挂载、卸载。K8sCSIPlugin 插件拥有三种工作模式，all-in-one、controller、nodeworker。all-in-one 模式包含上述三个服务，部署在 Kubernetes 的 Master 节点。controller 模式包含 Identity 和 Controller 两个服务，部署在 Kubernetes 的 Master 节点。nodeworker 模式包含 Identity 和 Node 两个服务，部署在 Kubernetes 的所有节点。拓扑图如图 1-1 所示。

图 1-1 K8sCSIPlugin 插件应用拓扑图



Kubernetes 集群通过 sidecar 容器与 CSI 插件进行对接，sidecar 容器服务是为了方便插件与 Kubernetes 集群进行对接开发的桥接服务，本质上属于 Kubernetes CSI 接口支持的一部分。Sidecar 容器按照 Kubernetes 的实现机制与 Kubernetes 进行交

互，并将消息按照 CSI 接口定义的规范调用 CSI 插件。

图中 controller 模式的 CSI 插件通过 csi-provisioner、csi-attacher、csi-snapshotter、csi-extender 等 sidecar 服务接收 kubernetes 集群的任务，并与存储交互完成管理任务。

nodeworker 模式的 CSI 插件通过 csi-driver-registrar 这个 sidecar 服务注册到 kubernetes 集群工作节点的 kubelet 服务中，后续 kubelet 服务直接调用 nodeworker 模式的 CSI 插件来完成工作节点卷挂载/卸载相关任务。在挂载设备时，nodeworker 模式的 CSI 插件会根据主机信息在存储上映射卷，并在主机上扫描，挂载设备；在卸载设备时，nodeworker 模式的 CSI 插件实现卸载，删除设备，并在存储上解除卷映射。

2 安装与部署

为了在 Kubernetes 集群中使用浪潮存储，需要提前完成存储的初始化，并将浪潮存储插件部署在集群中，all-in-one 模式和 controller 模式的插件部署在 Kubernetes 的 Master 节点。nodeworker 模式的插件部署在 Kubernetes 的所有节点。

在部署插件前，需要确保 Kubernetes 集群状态正常。

说明：部署插件时，无需重启 Kubernetes 服务。升级插件时，删除插件容器对应的 pods，修改部署文件，再重新创建 pods。

2.1 浪潮 K8sCSIPlugin 插件安装

具体过程如下：

1. 打开浪潮存储随机光盘中的 K8sCSIPlugin 插件安装包。

   解压并查看浪潮 K8sCSIPlugin 插件安装包中包含的文件：

```
root@lab:~/workspace$ ls
K8sPlugin_V2.1.0.Build20200114_amd64.tar.gz
root@lab:~/ K8sPlugin_V2.1.0.Build20200114_amd64# tree
├──── csiplugin
│    └── csiplugin
│    └── Dockerfile
│    └── deploy
│         └──── configMap.yaml
│         └──── csi-deploy.yaml
│         └──── csi-rbac.yaml
│         └──── centos-7.6.1810.base-for-csiplugin.docker-image
│         └──── README.md
│    └── demo
│         └──── csi-pvc.yaml
│         └──── csi-storageClass.yaml
```

   可以看到安装包中包含两个文件和两个目录，文件./csiplugin/csiplugin 是 CSI 插件，文件./csiplugin/Dockerfile 是制作 CSI 插件镜像的 Dockerfile，目录./csiplugin/deploy 放置部署文件，目录./csiplugin/demo 放置示例文件。

2. 部署过程如下：

a. 拉取镜像。

docker pull quay.io/k8scsi/csi-snapshotter:v1.2.2

docker pull quay.io/k8scsi/csi-resizer:canary

docker pull quay.io/k8scsi/csi-provisioner:canary

docker pull quay.io/k8scsi/csi-attacher:canary

docker pull quay.io/k8scsi/csi-node-driver-registrar:canary

拉取完成后，可通过命令 docker images | grep csi 查看已拉取的镜像。

b. 加载镜像。

docker load -i centos-7.6.1810.base-for-csiplugin.docker-image

加载完成后，可通过命令 docker images | grep centos 查看已加载的镜像。

c. 制作镜像。

chmod -R 777 csiplugin

docker build -t csiplugin:2.1.0 -f Dockerfile .

制作完成后，可通过命令 docker images | grep csiplugin 查看已制作的镜像。

d. 增加服务 kube-apiserver 的启动参数。

文件/etc/kubernetes/manifests/kube-apiserver.yaml 中增加

--feature-gates=VolumeSnapshotDataSource=true

--feature-gates=VolumePVCDataSource=true

--feature-gates=ExpandInUsePersistentVolumes=true

--feature-gates=ExpandCSIVolumes=true

增加完成后，服务 kube-apiserver 会自动重启。通过命令

ps aux | grep kube-apiserver 查看是否生效。

e. 增加服务 kube-controller-manager 的启动参数。

文件/etc/kubernetes/manifests/kube-controller-manager.yaml 中增加

--feature-gates=ExpandInUsePersistentVolumes=true

--feature-gates=ExpandCSIVolumes=true

增加完成后，服务 kube-controller-manager 会自动重启。通过命令 ps aux | grep kube-controller-manager 查看是否生效。

f. 增加服务 kubelet 的启动参数。

文件/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf 中增加

--feature-gates=ExpandInUsePersistentVolumes=true

--feature-gates=ExpandCSIVolumes=true

增加完成后，服务 kubelet 需要手动重启。可通过命令 systemctl daemon-reload 和 systemctl restart kubelet 完成重启。通过命令 ps aux | grep kubelet 查看是否生效。

g. 创建 socket 目录。

在目录/var/lib/kubelet/plugins/下创建文件夹 csi-instorage。

h. 创建 RBAC 权限信息。

通过命令 kubectl create -f csi-rbac.yaml 创建 RBAC 权限信息。

i. 创建存储配置信息。

修改 configMap.yaml 中存储信息（参考 3.3 节）。通过命令 kubectl create –f configMap.yaml 创建存储配置信息。

j. 部署 CSI 插件。

通过命令 kubectl create -f csi-deploy.yaml 部署 CSI 插件。通过命令 kubectl get pods 查看插件部署是否成功。

说明：需要用户修改 csi-deploy.yaml 文件中的 instorage-csi 部分的镜像名称（参考步骤 c 的镜像命名）。

图 2-1 查看部署结果

```
[root@node1 csiplugin]# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
instorage-csi-controller-85b897477c-f9trg   5/5     Running   6          7d23h
instorage-csi-node-kvpgw              2/2     Running   2          7d23h
nginx-01                              1/1     Running   1          7d23h
[root@node1 csiplugin]#
```

k. 至此，浪潮 K8sCSIPlugin 插件部署完毕，后续可参考功能配置与管理部分。

说明：插件是以容器方式部署的，可通过命令 kubectl logs ${pod_name} –c instorage-csi 查

看日志。

2.2 Kubernetes 集群搭建

Kubernetes 是一个开源的容器编排管理平台。Kubernetes 集群的搭建过程，使用方法，请参考官方网站 https://kubernetes.io/中的介绍。同时作为一个开源系统，各厂商可以基于 Kubernetes 发行自己的版本，针对厂商版本的使用方法，请参考各厂商提供的使用手册及相关文档。

3 功能配置与管理

为了在 Kubernetes 集群中使用浪潮存储，首先需要确保浪潮存储本身已经完成了初始化，并且完成存储池的创建。存储可以在集群中的各节点被访问，存储数据层通道（iSCSI、FC）可以正常使用，然后根据集群的具体信息对配置文件进行修改。

3.1 网络配置

管理 IP

驱动需要访问浪潮存储的管理接口，驱动使用 SSH 的方式与管理接口通信。驱动需要配置浪潮存储系统的 IP、SSH 端口。

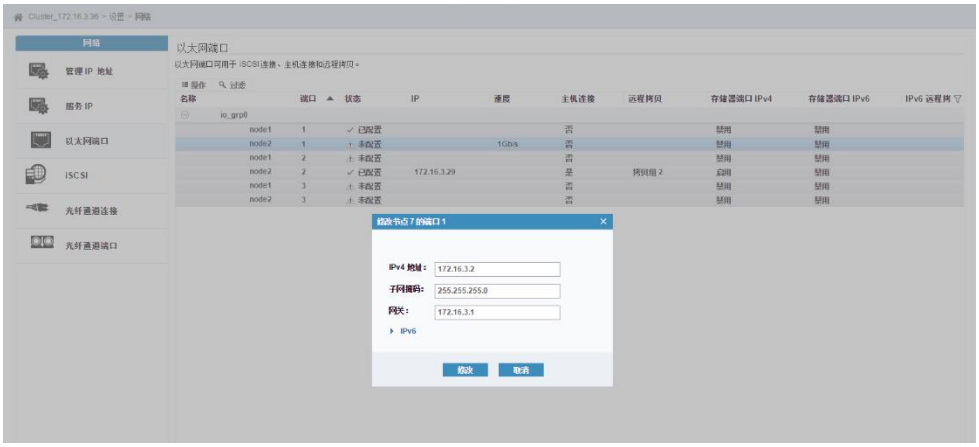
注意

- 确保插件所在节点具有存储系统的 SSH 访问权限。
- 浪潮存储设备必须配置有 iSCSI、FC，两者至少有一种，或者两者兼有。

iSCSI 网络配置

如果使用 iSCSI，则需要每个浪潮存储节点至少有一个 iSCSI 的 IP 地址。插件会直接从存储系统中获取 iSCSI IP，用户不需要给插件单独提供 iSCSI IP。虽然不需要单独为驱动分配 iSCSI IP，但是要在存储管理系统中设定 iSCSI 端口的 IP，在“设置 > 网络 > 以太网端口”中，选择已经连通的端口，单击鼠标右键，选择“修改 IP”，填写有效的 IP、子网掩码、网关。如图 3-1 所示。

图 3-1 添加 iSCSI 端口 IP



注意

如果使用 iSCSI，确保各工作节点与存储系统有畅通的 iSCSI 网络可供访问。

FC 网络配置

如果使用 FC，则需要每个浪潮存储节点至少配置有一个 WWPN 端口。插件会使用所有可用的 WWPN 端口将卷挂载目标主机。插件将直接从存储系统中获取 WWPN，用户无需为驱动单独提供 WWPN。

注意

如果使用 FC，确保各工作节点与存储系统有 FC 连接。

3.2 启用多路径

为了提升 SAN 存储卷的可靠性，在生产环境中，通常会启用多路径。

在 Kubernetes 环境中，如果多路径相关配置设置不恰当，在实际使用中有可能会使用到单路径设备，如果该路径出现损坏，则可能会产生 I/O 错误。

在 Linux 环境下，浪潮存储利用 Linux 系统自带的 device-mapper-multipath 服务进行多路径聚合，如果需要启用多路径，首先需要保证 Kubernetes 中各工作节点均按照要求部署安装 multipathd 服务。然后对 Kubernetes 环境中的各工作节点多路径相关的配置，请参考下面的描述，以正确使用多路径。

1. 在 multipath.conf 配置中正确设置设备黑名单。

在 Kubernetes 环境中，一个工作节点上，会同时挂载非常多的卷，每个卷又会有多条路径，从而使得节点上的 sdX 设备会非常多。多路径的配置文件中 blacklist 配置组中的参数会过滤符合 blacklist 条件的路径。如果该配置过滤的设备不恰当，可能会导致部分路径无法进行多路径聚合。

例如，如果 devnode 参数配置成类似“^sda”时，会导致 sdaa，sdab 等路径无法进行路径聚合，以致路径缺失，正确的应该是“^sda$”，即仅将 sda 设备屏蔽掉。

针对 multipath.conf 配置文件的具体用法，Linux 用户可以通过 man 5 multipath.conf 获取帮助。请确保不会将浪潮存储提供的设备路径加入设备黑名单。

注意

multipath.conf 配置文件的具体路径，请咨询 Kubernetes 容器平台提供商，linux 系统下该文件默认路径为/etc/multipath.conf。

2. 在 multipath.conf 配置文件中，正确设置浪潮存储推荐的设备配置参数。

目前浪潮存储的推荐多路径配置已经合入到多路径工具的社区版本中，针对使用旧版本多路径工具的场景，需要在配置文件中加入浪潮推荐的多路径配置。即在 devices 配置组中增加浪潮存储的 device 配置内容，devices 配置组中可能会存在多个存储厂商的 device 配置。通常配置信息如下：

```
device{
        vendor "INSPUR"
        product "MCS"
        path_grouping_policy group_by_prio
        path_selector "round-robin 0"
        features “1 queue_if_no_path”
        prio alua
        path_checker tur
        failback immediate
        no_path_retry “60”
        rr_min_io 1
        dev_loss_tmo 120
```

```
        fast_io_fail_tmo 5
}
```

3.3 完善 K8sCSIPlugin 插件存储配置文件

浪潮 K8sPlugin CSI 插件存储配置文件为插件目录中的 deploy/configMap.yaml 文件。文件为 yaml 格式。

具体配置如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: inspur-instorage-01
data:
  instorage.yaml: |
    log:
      enabled: false
      logdir: log
      level: ""
      logrotatemaxsize: 0
    host:
      link: iscsi
      forceUseMultipath: false
      scsiScanRetryTimes: 3
      scsiScanWaitInterval: 1
      iscsiPathCheckRetryTimes: 3
      iscsiPathCheckWaitInterval: 1
      multipathSearchRetryTimes: 3
      multipathSearchWaitInterval: 1
      multipathResizeDelay: 1
    storage:
    - name: storage-01
      type: AS18000
      host: 10.0.0.1:22
      username: username
      password: password
      shadow: <shadow of the password, can be generated by instorage flexvolume driver. like
```

```
'./instorage ext-encrypt-password [password]'>
  barrierPath: ""
```

配置说明如下：

表 3-1 配置说明

配置名称	说明
log.enabled	是否打开插件日志。 true 打开，false 不打开。
log.logdir	日志输出目录。
log.level	日志输出级别。debug/info/warning/error。
Log.logrotatemaxsize	日志文件滚动大小阈值
host.link	数据通道连接类型。 iscsi 使用 iSCSI 连接方式。 fc 使用 FC 连接方式。
host. forceUseMultipath	是否强制使用多路径。 true 强制，false 不强制。
host. scsiScanRetryTimes	SCSI 设备扫描尝试次数。
host. scsiScanWaitInterval	SCSI 设备扫描失败后等待间隔。单位为秒。
host. iscsiPathCheckRetryTimes	iSCSI 路径烧苗检查尝试次数。
host. iscsiPathCheckWaitInterval	iSCSI 路径扫描失败后等待间隔，单位为秒。
host. multipathSearchRetryTimes	多路径设备查找重试次数。
host. multipathSearchWaitInterval	多路径设备查找失败后等待间隔，单位为秒。
host. multipathResizeDelay	在线扩容时，多路径 resize 命令延迟时间。单位为秒。
host. attachExtendFileLockPath	挂载卸载扩容操作并发控制文件锁路径。默认使用配置文件。
storage[].name	存储名称。配置文件范围内唯一，区分多个存储。当前只支持一个存储。
Storage[].type	存储类型，必须配置。类型为 AS18000。

storage[].host	存储 SSH 访问路径。 格式为 IP:Port，如 10.0.0.1:22。
storage[].username	存储 SSH 访问时的用户名。
storage[].password	存储 SSH 访问时的密码明文。
storage[].shadow	存储 SSH 访问时的密码明文加密后的密文。当设置了 password 时，优先使用 password 设置的密码明文，当 password 未设置时，使用 shadow 设置的密码密文。密码密文可以通过执行 ./instorage ext-encrypt-password [password] 获得。
storage[].barrierPath	双活卷扩容目前由插件组合调用存储端命令完成，多个命令调用中间出错以后，无法回滚，且存储端不能再执行对应主机上的卷挂载/卸载/双活卷扩容操作。因此失败后，会在该目录生成屏障文件，阻止后续任务执行。默认为插件程序所在目录。

4 Kubernetes 中使用存储

通过浪潮 K8sCSIPlugin 插件，可以实现在 Kubernetes 集群中使用浪潮存储。Kubernetes 中使用持久化存储的方式包括通过 PVC（Persistent Volume Claim）使用卷和直接使用卷两种方式。

4.1 通过新建卷创建 POD

通过新建卷创建 POD 时，可以利用 K8sCSIPlugin 插件自动的在存储上创建满足要求的卷，并在对应 PV 删除时，自动删除存储上对应的卷。当然卷的创建、删除，以及集群上对应 PV 的创建、删除也可以通过手动的方式来处理。

在使用前，首先需要创建 StorageClass 资源信息，该资源信息用于表示一种卷的类型；一方面作为 PV 和 PVC 之间互相关联的纽带，另一方面，StorageClass 信息中可以标识插件的名称和插件创建卷过程中使用的参数信息。

之后用户就可以通过创建一个 PVC 资源信息来请求一个具体的卷，当插件完成存储上的卷创建以及 Kubernetes 中 PV 的创建后，PVC 会与对应的 PV 进行关联，然后用户就可以创建 Pod 来使用这个 PVC 了。具体过程如下：

1. 在 Kubernetes 集群中创建 StorageClass 资源信息。

```
k8s@dev:~/k8s$ cat csi-sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: csi-sc
provisioner: csi-instorage
parameters:
   volPoolName: Pool0
   volThin: "true"
   volThinResize: "2"
   volThinGrainSize: "256"
   volThinWarning: "20"
reclaimPolicy: Delete
k8s@dev:~/k8s$ kubectl create -f csi-sc.yaml
```

```
storageclass.storage.k8s.io "csi-sc" created
k8s@dev:~/k8s$ kubectl get sc
NAME      PROVISIONER    AGE
csi-sc        csi-instorage        33s
```

a. 通过 StorageClass 资源信息中的 provisioner 属性来标识插件的名称，使用浪潮 K8sCSIPlugin 插件时，该属性需要设置为 csi-instorage。

b. 通过 parameters 属性来指定创建卷时的参数。参数见 4.8 章节。

2. 在 Kubernetes 集群中创建 PVC，来声明使用存储资源。

```
k8s@dev:~/k8s$ cat csi-pvc-dynamic.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: csi-pvc-01
spec:
   accessModes:
      - ReadWriteOnce
   volumeMode: Filesystem
   resources:
      requests:
         storage: 1Gi
   storageClassName: csi-sc
k8s@dev:~/k8s$ kubectl apply -f csi-pvc-dynamic.yaml
persistentvolumeclaim " csi-pvc-01" created
k8s@dev:~/k8s$ kubectl get pvc
NAME          STATUS      VOLUME                                     CAPACITY
ACCESS MODES     STORAGECLASS     AGE
csi-pvc-01          Bound          pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd          1Gi
RWO                csi-sc          9s
k8s@dev:~/k8s$ kubectl get pv
NAME                                        CAPACITY     ACCESS MODES
RECLAIM POLICY     STATUS     CLAIM                          STORAGECLASS
REASON     AGE
pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd     1Gi          RWO                Delete
Bound      default/csi-pvc-01          csi-sc                      21h
```

示例中我们定义了一个 PVC，是 csi-pvc-01，插件收到 PVC 创建的消息后，会根据 StorageClass 的参数信息，在存储上创建对应的卷，并在 Kubernetes 集群中创建对应的 PV 资源信息，之后 PVC 会与 PV 互相绑定，创建的具体信息如下表：

表 4-1 对应的 PV 资源信息

PVC	对应 PV 名称	SC 名称
csi-pvc-001	pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd	csi-sc

通过查看存储上的卷信息，可以发现 PV 对应的卷已经创建：

图 4-1 查看卷信息

3. 当 PVC 与 PV 互相绑定之后，用户就可以利用 PVC 来为 Pod 中的服务提供持久化存储了。以下示例创建 Pod，并在 Pod 中使用 PVC。

```
k8s@dev:~/k8s$ cat pod-use-dynamic-pvc.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-01
spec:
  containers:
    - name: nginx
      image: nginx:v1
      ports:
        - containerPort: 80
      volumeMounts:
        - name: k8s-test-01
          mountPath: /mnt
  volumes:
  - name: k8s-test-01
    persistentVolumeClaim:
      claimName: csi-pvc-01
k8s@dev:~/k8s$ kubectl create -f pod-use-dynamic-pvc.yaml
pod/nginx-01 created
k8s@dev:~/k8s$ kubectl get pods
NAME                                                READY     STATUS
RESTARTS     AGE
nginx-01                                            1/1       Running     0
46m
```

示例中创建了名称为 nginx-01 的 Pod，Pod 中通过使用 csi-pvc-01 这个 PVC 所

声明的卷，映射到 Pod 中的 nginx 容器的/mnt 目录。

4. 查看工作节点，可以看到卷被正确的映射和挂载。

图 4-2 查看卷信息

```
[root@node1 dynamic]# ls -l /dev/disk/by-path | grep 241
lrwxrwxrwx 1 root root  9 12月 17 16:33 ip-100.7.46.117:3260-iscsi-iqn.2004-12.com.inspur:mcs.cluster241.node1-l
un-0 -> ../../sde
lrwxrwxrwx 1 root root  9 12月 17 16:33 ip-100.7.46.118:3260-iscsi-iqn.2004-12.com.inspur:mcs.cluster241.node2-l
un-0 -> ../../sdf
[root@node1 dynamic]# mount |grep pvc
/dev/mapper/36005076000d189c0d0000000000001e7 on /var/lib/kubelet/pods/87fdc933-2780-4b78-b7ee-2a22c95d43f1/volu
mes/kubernetes.io~csi/pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd/mount type ext4 (rw,relatime,stripe=8,data=ordere
d)
```

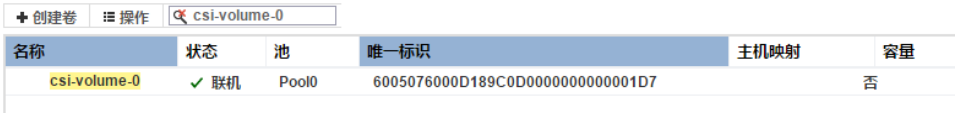
5. 当不再需要这个 Pod 时，删除该 Pod 即可。
6. Pod 删除后，其所引用的 PVC 提供的卷会被对应的服务所释放。根据需要可以删除 PVC。PVC 删除后，对应的 PV 会根据自身创建时设置的 reclaim 策略确定是否需要删除，如果为 delete，则自动删除，当然 PV 也可以通过手动的方式删除。
7. 当 PV 删除后，PV 删除的事件消息会被 K8sPlugin 中的 instorage-csi 捕获到，instorage-csi 会根据 PV 的信息中的属性，将存储上对应的卷删除。

4.2 通过已有卷创建 POD

通过已有卷创建 PV，然后在创建 PVC 时绑定该 PV，最后基于 PVC 创建 POD。过程如下：

1. 确保存储端已创建待使用的卷。

图 4-3 卷信息



2. 在 Kubernetes 集群中创建 SC，参考 4.1。
3. 在 Kubernetes 集群中创建 PV。

```
k8s@dev:~/k8s$ cat csi-pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
   name: pv-02
```

```
spec:
   storageClassName: csi-sc
   capacity:
      storage: 1Gi
   accessModes:
      - ReadWriteOnce
   csi:
      driver: csi-instorage
      fsType: ext4
      volumeHandle: csi-volume-0
k8s@dev:~/k8s$ kubectl create -f csi-pv.yaml
```

注意

spec.csi.volumeHandle 所使用的卷名称需要与存储上的卷名称保持一致。

4. 在 Kubernetes 集群中创建 PVC。

```
k8s@dev:~/k8s $ cat csi-pvc-static.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: csi-pvc-02
spec:
   accessModes:
      - ReadWriteOnce
   volumeMode: Filesystem
   resources:
      requests:
         storage: 1Gi
   storageClassName: csi-sc
k8s@dev:~/k8s$ kubectl create -f csi-pvc-static.yaml
```

5. 在 Kubernetes 集群中创建 Pod。

```
k8s@dev:~/k8s $ cat pod-use-static-pvc.yaml
apiVersion: v1
kind: Pod
metadata:
   name: nginx-02
spec:
   containers:
      - name: nginx
         image: nginx:v1
         ports:
            - containerPort: 80
         volumeMounts:
```

```
        - name: k8s-test-02
          mountPath: /mnt
  volumes:
  - name: k8s-test-02
    persistentVolumeClaim:
      claimName: csi-pvc-02
k8s@dev:~/k8s$ kubectl create -f pod-use-static-pvc.yaml
```

4.3 通过 PVC 创建快照

创建快照的过程类似创建 PVC 过程，这里涉及三个扩展资源，分别是 VolumeSnapshotClass 、 VolumeSnapshot 、 VolumeSnapshotContent 。 VolumeSnapshotClass 相当于 SC， VolumeSnapshot 相当于 PVC， VolumeSnapshotContent 相当于 PV。

在使用前，首先需要创建 VolumeSnapshotClass 资源信息，该资源信息用于表示一种快照类型，一方面作为 VolumeSnapshot 和 VolumeSnapshotContent 之间互相关联的纽带，另一方面，VolumeSnapshotClass 信息中可以标识插件的名称和插件创建快照过程中使用的参数信息。

具体过程如下：

1. 在 Kubernetes 集群中创建 VolumeSnapshotClass 资源信息。

```
k8s@dev:~/k8s$ cat vsc.yaml
kind: VolumeSnapshotClass
apiVersion: snapshot.storage.k8s.io/v1alpha1
metadata:
  name: csi-vsc
snapshotter: csi-instorage
k8s@dev:~/k8s$ kubectl create -f vsc.yaml
volumesnapshotclass.snapshot.storage.k8s.io/csi-vsc created
k8s@dev:~/k8s$ kubectl get volumesnapshotclass
NAME          AGE
csi-vsc       12d
```

通过 VolumeSnapshotClass 资源信息中的 snapshotter 属性来标识插件的名称，使用浪潮 K8sCSIPlugin 插件时，该属性需要设置为 csi-instorage。

2. 在 Kubernetes 集群中创建 VolumeSnapshot。

```
k8s@dev:~/k8s$ cat vs.yaml
apiVersion: snapshot.storage.k8s.io/v1alpha1
```

```
kind: VolumeSnapshot
metadata:
  name: csi-vs-01
spec:
  snapshotClassName: csi-vsc
  source:
    name: csi-pvc-01
    kind: PersistentVolumeClaim
k8s@dev:~/k8s$ kubectl create -f vs.yaml
volumesnapshot.snapshot.storage.k8s.io/csi-vs-01 created
k8s@dev:~/k8s$ kubectl get volumesnapshot
NAME                AGE
csi-vs-01           16s
k8s@dev:~/k8s$ kubectl get volumesnapshotcontent
NAME                                                AGE
snapcontent-002bf754-261a-4ee9-86b8-6ab0796749c2    16s
```

示例中我们定义了一个 VolumeSnapshot，是 csi-vs-01，插件收到 VolumeSnapshot 创建的消息后，会根据 VolumeSnapshotClass 的参数信息和 Source 信息，在存储上创建对应的快照，并在 Kubernetes 集群中创建对应的 VolumeSnapshotContent 资源信息，之后 VolumeSnapshot 会与 VolumeSnapshotContent 互相绑定。

4.4 通过 PVC 克隆 PVC

通过 PVC 方式克隆 PVC 就是通过卷克隆卷，具体过程如下：

1. 在 Kubernetes 集群中创建 PVC，参考 4.1。
2. 在 Kubernetes 集群中通过 PVC 方式克隆 PVC。

```
k8s@dev:~/k8s$ cat pvc-from-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-sc
  resources:
    requests:
      storage: 1Gi
```

```
  volumeMode: Filesystem
  dataSource:
    name: csi-pvc-01
    kind: PersistentVolumeClaim
k8s@dev:~/k8s$ kubectl create -f pvc-from-pvc.yaml
persistentvolumeclaim " pvc-from-pvc " created
k8s@dev:~/k8s$ kubectl get pvc
NAME                                                STATUS              VOLUME
CAPACITY     ACCESS MODES     STORAGECLASS     AGE
csi-pvc-01              Bound          pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd
1Gi              RWO              csi-sc              41m
pvc-from-pvc          Bound          pvc-ad8006a7-8e1c-44cc-81ba-28062b20c0f7
1Gi              RWO              csi-sc              8s
k8s@dev:~/k8s$ kubectl get pv
NAME                                                CAPACITY     ACCESS
MODES              RECLAIM POLICY          STATUS              CLAIM
STORAGECLASS     REASON     AGE
pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd          1Gi                RWO
Delete                Bound          default/csi-pvc-01              csi-sc
42m
pvc-ad8006a7-8e1c-44cc-81ba-28062b20c0f7          1Gi                RWO
Delete                Bound          default/pvc-from-pvc          csi-sc
29s
```

示例中通过旧 PVC csi-pvc-01 克隆新的 PVC pvc-from-pvc。

4.5 通过快照克隆 PVC

通过 VolumeSnapshot 方式克隆 PVC 就是通过快照克隆卷，具体过程如下：

1. 在 Kubernetes 集群中创建 VolumeSnapshot，参考 4.3。
2. 在 Kubernetes 集群中通过快照方式克隆 PVC。

```
k8s@dev:~/k8s$ cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-sc
  resources:
    requests:
```

```
        storage: 1Gi
  dataSource:
    name: csi-vs-01
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
k8s@dev:~/k8s$ kubectl create -f pvc-from-snap.yaml
persistentvolumeclaim " pvc-from-snap " created
k8s@dev:~/k8s$ kubectl get pvc
NAME                                        STATUS          VOLUME
CAPACITY    ACCESS MODES    STORAGECLASS    AGE
csi-pvc-01          Bound       pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd
1Gi         RWO             csi-sc          49m
pvc-from-snap       Bound       pvc-89db5dc8-147f-4168-82e7-c7c7c32e6d3d
1Gi         RWO             csi-sc          2m46s
k8s@dev:~/k8s$ kubectl get pv
NAME                                        CAPACITY    ACCESS
MODES           RECLAIM POLICY          STATUS          CLAIM
STORAGECLASS    REASON      AGE
pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd    1Gi             RWO
Delete              Bound       default/csi-pvc-01          csi-sc
49m
pvc-89db5dc8-147f-4168-82e7-c7c7c32e6d3d    1Gi             RWO
Delete              Bound       default/pvc-from-snap       csi-sc
2m55s
```

示例中通过快照 csi-vs-01 克隆新的 PVC pvc-from-snap。

4.6 离线扩容

扩容是指对卷的扩容，在 Kubernetes 集群中体现在 PVC 上。具体过程如下：

1. 在 Kubernetes 集群中创建 StorageClass 资源信息。

```
k8s@dev:~/k8s$ cat csi-sc-resizer.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-sc-resizer
provisioner: csi-instorage
parameters:
  volPoolName: Pool0
  volThin: "true"
  volThinResize: "2"
  volThinGrainSize: "256"
```

```
  volThinWarning: "20"
reclaimPolicy: Delete
allowVolumeExpansion: true
k8s@dev:~/k8s$ kubectl create -f csi-sc-resizer.yaml
storageclass.storage.k8s.io/csi-sc-resizer created
k8s@dev:~/k8s$ kubectl get sc
NAME              PROVISIONER          AGE
csi-sc-resizer    csi-instorage        33s
```

属性 allowVolumeExpansion 必须设置 true。

2. 在 Kubernetes 集群中创建 PVC，来声明使用存储资源。

```
k8s@dev:~/k8s$ cat csi-pvc-resizer.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-resizer-01
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 2Gi
  storageClassName: csi-sc-resizer
k8s@dev:~/k8s$ kubectl create -f pvc.yaml
persistentvolumeclaim/csi-pvc-resizer-01 created
k8s@dev:~/k8s$ kubectl get pvc
NAME                                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS      AGE
csi-pvc-01                                          Bound    pvc-6789682a-7d0a-4243-a5c2-
329cb28ba8cd   1Gi        RWO            csi-sc            16h
csi-pvc-resizer-01   Bound    pvc-05491ede-5087-438d-b254-b6f3b89fda02
2Gi        RWO            csi-sc-resizer    4s
k8s@dev:~/k8s$ kubectl get pv
NAME                                                CAPACITY   ACCESS
MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS      REASON   AGE
pvc-05491ede-5087-438d-b254-b6f3b89fda02   2Gi        RWO
Delete          Bound    default/csi-pvc-resizer-01   csi-sc-resizer
52s
pvc-6789682a-7d0a-4243-a5c2-329cb28ba8cd   1Gi        RWO
Delete          Bound    default/csi-pvc-01           csi-sc
16h
```

3. 在 Kubernetes 集群中编辑 PVC。

```
k8s@dev:~/k8s$ kubectl edit pvc csi-pvc-resizer-01
属性 spec.resources.requests.storage 的值从 2Gi 改成 3Gi。
persistentvolumeclaim/csi-pvc-resizer-01 edited
k8s@dev:~/k8s$ kubectl get pvc csi-pvc-resizer-01
NAME                  STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-resizer-01    Bound    pvc-05491ede-5087-438d-b254-b6f3b89fda02
2Gi        RWO            csi-sc-resizer   5m33s
```

csi-pvc-resizer-01 的大小仍然是 2Gi。

4. 在 Kubernetes 集群中创建 POD。

```
k8s@dev:~/k8s$ cat pod-use-resizer-pvc.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-03
spec:
  containers:
    - name: nginx
      image: nginx:v1
      ports:
        - containerPort: 80
      volumeMounts:
        - name: k8s-test-03
          mountPath: /mnt
  volumes:
  - name: k8s-test-03
    persistentVolumeClaim:
      claimName: csi-pvc-resizer-01
k8s@dev:~/k8s$ kubectl create -f pod-use-resizer-pvc.yaml
pod/nginx-03 created
k8s@dev:~/k8s$ kubectl get pod
NAME                                  READY   STATUS
RESTARTS   AGE
instorage-csi-controller-85b897477c-jgp5n   5/5     Running   0
22h
instorage-csi-node-7dpdd                    2/2     Running   0
22h
```

```
nginx-03                                    1/1     Running   0
5m32s k8s@dev:~/k8s$ kubectl get pvc
NAME                                                    STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-01                                              Bound    pvc-6789682a-7d0a-4243-a5c2-
329cb28ba8cd   1Gi        RWO            csi-sc         16h
csi-pvc-resizer-01   Bound    pvc-05491ede-5087-438d-b254-b6f3b89fda02
3Gi        RWO            csi-sc-resizer   17m
```

csi-pvc-resizer-01 的大小已变成 3Gi。

4.7 在线扩容

在 4.6 节的基础上操作。具体过程如下：

1. 在 Kubernetes 集群中编辑 PVC。

```
k8s@dev:~/k8s$ kubectl edit pvc csi-pvc-resizer-01
属性 spec.resources.requests.storage 的值从 3Gi 改成 5Gi。
persistentvolumeclaim/csi-pvc-resizer-01 edited
k8s@dev:~/k8s$ kubectl get pvc csi-pvc-resizer-01
NAME                 STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-resizer-01   Bound    pvc-05491ede-5087-438d-b254-b6f3b89fda02
5Gi        RWO            csi-sc-resizer   22m
```

csi-pvc-resizer-01 的大小已变成 5Gi。

4.8 StorageClass 资源存储插件参数说明

表 4-2StorageClass 资源存储插件参数说明

名称	类型	说明	是否必须
volPoolName	string	创建卷时所在的池的名称。	必须
volAuxPoolName	string	创建卷时辅助卷所在的池的名称。	双活/镜像卷时必须
volIOGrp	string	创建卷时所在的 IO 组的 ID 号。	双活卷时必须

volAuxIOGrp	string	创建卷时辅助卷所在的 IO 组的 IO 号。	双活卷时必须
volThin	string	是否创建精简卷。值为 true 或 false 字符串。开启压缩时，自动开启精简卷。	默认 false
volCompress	string	是否创建压缩卷。值为 true 或 false 字符串。开启压缩时，自动开启精简卷。	默认 false
volInTier	string	是否开启分层。值为 true 或 false 字符串。	默认 false
volLevel	string	卷的等级类型。普通卷为 basic，镜像卷为 mirror，双活卷为 aa。	默认 basic，非必须。
volThinResize	string	创建精简卷时，初始的卷大小占实际容量的百分比。	精简卷时必须
volThinGrainSize	string	精简卷增长时的块大小，单位为 KiB。可选 32, 64, 128，256 等值。	精简卷时有效，非必须。
volThinWarning	string	精简卷告警阈值。警告时容量占实际容量的百分比。	精简卷时有效，非必须。
volAutoExpand	string	是否开启自动扩张。值为 true 或 false 字符串。	默认 false

5 故障分析与解决

1. **网络问题**。

   Kubernetes 集群中所有部署了浪潮存储 K8sCSIPlugin 插件的节点都需要与存储的管理网络互通，插件是通过 SSH 连接到存储上执行存储端的相关命令来完成卷管理相关的操作。

2. **SAN 网络异常**。

   存储上的卷被容器业务使用时，通过 SAN 网络将卷挂载到主机，如果 SAN 网络异常，则主机端无法正常使用存储上的卷。

3. **认证失败**。

   无法连接到浪潮存储设备，请检查用户名、密码是否配置正确。

4. 容器使用的设备是单路经设备，检查多路径相关的配置是否正确。

5. 卷无法创建/删除，可通过命令 kubectl logs ${controller-pod-name} -c ${container-name}查看插件是否被调用。

6. 卷无法挂载/卸载，可通过命令 kubectl logs ${node-pod-name} -c ${container-name}查看插件是否被调用

7. 存储上的卷无法在工作节点上发现，检查工作节点与存储之间的数据网络是否互通。

8. 插件创建的卷默认是没有进行文件系统格式化的，因此第一次被容器使用时，需要使用读写模式挂载，系统才会自动格式化该卷。如果第一次使用卷是只读模式，卷是不会自动格式化文件系统的，因此就没有办法挂载并映射给容器使用。

6 术语&缩略语

A		
API	Application Program Interface	应用程序接口
D		
Deployment	-	Kubernetes 集群中部署业务的一种形式。
F		
FC	Fiber Channel	光纤通道
I		
iSCSI	Internet Small Computer System Interface	互联网小型计算机接口
InStorage	Inspur Storage	浪潮存储
K		
-	Kubernetes	Kubernetes 容器管理平台
K8s	Kubernetes	Kubernetes 缩写
K8sPlugin	Kubernetes Plugin	浪潮存储 Kubernetes 存储管理平台插件
P		
-	Pod	Kubernetes 集群中的最小业务单元。
PV	Persistent Volume	Kubernetes 集群中持久卷资源
PVC	Persistent Volume Claim	Kubernetes 集群中持久卷声明资源。
S		

SC	StorageClass	Kubernetes 集群中的存储类型资源
-	SideCar	Pod 中的辅助容器，用于完成一些独立于主进程（主容器）之外的工作。
SSH	Secure Shell Protocol	一种安全协议
W		
WWPN	World Wide Port Name	全球端口名称

附录一 Kubernetes 链接

浪潮所开发的产品属于 Kubernetes 存储相关的插件。利用该插件，可以通过 Kubernetes 来创建卷、删除卷等，并根据业务的调度自动将卷挂载到指定的节点供业务容器使用，并当业务容器结束后，自动将卷从相应主机上卸载下来。

Kubernetes 中的相关概念，部署配置方法，使用方法等知识，请参考 Kubernetes 社区中的相关资料，或通过 Kubernetes 提供商获取相关使用资料。以下提供了部分社区公开资料：

1. Kubernetes 社区官方网站：

   https://kubernetes.io/

2. Kubernetes 社区部署安装资料：

   https://kubernetes.io/docs/setup/

3. Kubernetes 社区存储相关资料：

   https://kubernetes.io/docs/concepts/storage/volumes/

4. Kubernetes 开发设计等过程及资料均托管在 GitHub 上：

   https://github.com/kubernetes/

5. Kubernetes 存储相关开发信息

   https://github.com/kubernetes/community/tree/master/sig-storage

6. CSI 接口文档

   https://github.com/container-storage-interface/spec

附录二 双活卷扩容说明

当前存储端本身不支持双活卷的扩容操作，K8sPlugiCSI 插件为了实现对双活卷扩容的支持，通过组合存储管理命令来完成双活卷的扩容。在扩容过程中，会依次执行以下命令来完成存储端双活卷的扩容。

1. 删除双活卷主卷-辅助卷之间的远程复制关系。

```
mcsop rmrcrelationship <rc-id>
```

2. 扩容主卷。

```
mcsop expandvdisksize -size <add-size> -unit gb <master-name>
```

3. 扩容辅助卷。

```
mcsop expandvdisksize -size <add-size> -unit gb <aux-name>
```

4. 扩容主卷对应的变更卷。

```
mcsop expandvdisksize -size <add-size> -unit gb <master-change-volume>
```

5. 扩容辅助卷对应的变更卷。

```
mcsop expandvdisksize -size <add-size> -unit gb <aux-change-volume>
```

6. 重建主卷辅助卷远程复制关系。

```
mcsop mkrcrelationship -master <master-name> -aux <aux-name> -cluster
<cluster-name> -sync -activeactive
```

7. 增加主卷在远端站点的访问能力。

```
mcsop addvdiskaccess -iogrp <new-iogrp> <master-name>
```

8. 关联主卷和主卷变更卷。

```
mcsop chrcrelationship -masterchange <master-change-volume> <rc-id>
```

9. 关联辅助卷和辅助卷变更卷。

```
mcsop chrcrelationship -auxchange <aux-change-volume> <rc-id>
```

按照以上命令操作过程中，有以下事项需要注意：

1. 双活卷不能有 I/O 下发，插件通过在主机端冻结双活卷挂载到的文件系统来确保过程中没有 I/O 下发。

2. 过程中不能有卷在存储上映射到该主机，或者与该主机解除映射。否则可能造成主机端卷的路径异常，导致系统混乱。插件通过使用文件锁的方式确保插件本身不会同时触发在同一个主机上的双活卷扩容，卷挂载到主机，卷从主机卸载操作。
3. 由于存储端双活卷扩容是由多个命令组合完成的，当一条命令出错后，插件不进行回滚操作，失败后，会在该主机上指定目录生成屏障文件，当有屏障文件存在时，插件在该主机上的双活卷扩容/卷挂载/卷卸载命令处理均会失败，避免对处于失败的双活卷产生新的影响。屏障文件名称为“aa-extend-failure.barrier”。双活卷扩容失败后，可以参考日志文件，确定扩容过程中已成功的步骤，失败的步骤及未处理的步骤，并在存储端完成修复操作，修复后可删除屏障文件。屏障文件删除后，插件才可以继续执行双活卷扩容/卷挂载/卷卸载命令。
4. 如果由于执行存储命令组合时失败导致双活卷扩容失败，需要尽快停止使用对应双活卷的业务，避免继续写产生问题。然后再确定具体从哪一步开始失败，可以根据日志里面的信息，尝试访问存储，通过手动执行命令，完成失败命令以及后续命令的执行。
5. 双活卷扩容命令组合失败，通常不会对主卷造成影响，数据本身不会有异常，但是双活卷中主卷，辅助卷之间的同步关系通常已经不能满足双活卷的要求，需要进行修复才可以满足双活卷的要求。