天翼云
State Cloud

存储资源盘活系统

Container Storage Interface(CSI) 使用手册

天翼云科技有限公司

修订记录

版本	发布日期	说明	适配的 HBlock 版本
1.4	2025 年 03 月 03 日	1. 支持卷的折叠副本数（仅 HBlock 集群版支持）。 2. 支持调整 PV 的服务端连接位置。	3.5 及以上
1.3.2	2024 年 12 月 11 日	1. 支持 Kubernetes service 域名。 2. 支持指定 HBlock 创建 LUN 的等待时间。 3. 支持设置 HBlock 节点信息，用来控制卷的 iqn 所在区域。	3.5 及以上
1.3	2024 年 08 月 08 日	1. 支持设置卷的存储池和缓存池。 2. 支持上云卷。 3. Target 迁移后支持自动重连。 4. Block Volume 模式支持 RWX、ROX、RWO 访问方式。 5. Filesystem 模式支持 RWO，同时静态 PV 场景下支持 ROX。	3.5 及以上
1.2	2024 年 05 月 30 日	1. 支持同时管理多个 HBlock 集群。 2. Block Volume 支持	3.5 及以上

		RWX、ROX 访问方式。 3. 支持 Helm 安装 CSI 插件。 4. 支持设置卷的最小副本数和 Target 最大连接数。	
1.1	2023 年 11 月 15 日	1. 新增 Block Volume 支持。 2. 创建 Target 时支持指定服务器个数，支持一主多备。 3. 创建卷时支持指定 EC 卷的切片大小。	3.4 及以上
1.0	2023 年 09 月 20 日	第一次发布： 1. 支持静态 PV 和动态 PV 场景。 2. 支持 MPIO，静态 PV 场景下支持一主多备。 3. 支持 Target Portal IP。	

目 录

1 产品定义

HBlock 是中国电信天翼云自主研发的存储资源盘活系统（Storage Resource Reutilization System，简称 SRRS），是一款轻量级存储集群控制器，实现了全用户态的软件定义存储，将通用服务器及其管理的闲置存储资源转换成高可用的虚拟磁盘，通过标准 iSCSI 协议提供分布式块存储服务，挂载给本地服务器（或其他远程服务器）使用，实现对资源的集约利用。同时，产品拥有良好的异构设备兼容性及场景化适配能力，无惧 IT 架构升级带来的挑战，助力企业用户降本增效和绿色转型。

Kubernetes 是一个开源的容器集群管理系统，可以实现容器集群的自动化部署、自动扩缩容、维护等功能。通过 Kubernetes 可以快速部署应用，快速扩展应用，无缝对接新的应用功能，节省资源，优化硬件资源的使用。

企业在使用 Kubernetes 部署容器时，通常需要使用持久化存储。持久卷（Persistent Volume）是独立于 Pod 的存储资源，它的生命周期与 Pod 无关，在存储节点中创建持久卷后，即可将其提供给计算节点中的 Pod 使用，实现了计算资源和存储资源的解耦。Kubernetes CSI 插件可以为有状态应用提供持久化存储能力。Kubernetes 集群中的计算节点挂载持久卷作为磁盘，供节点上运行在 Pod 内的数据库等应用使用。容器中使用的数据库或有状态应用程序，需要依赖于高稳定性、高性能的存储系统，数据需要在硬件或软件重启后仍能继续保留。

Kubernetes 通过 PV、PVC、Storageclass 提供了一种强大的基于插件的存储管理机制，并且引入了容器存储接口 Container Storage Interface（CSI）机制，用于在 Kubernetes 和外部存储系统之间建立一套标准的存储管理接口，通过该接口可以为容器提供存储服务。存储提供方只需要基于标准接口进行存储插件的实现，就能使用 Kubernetes 的原生存储机制为容器提供存储服务。HBlock Container Storage Interface(CSI)插件是一种符合 CSI 规范的驱动程序。

HBlock CSI 插件与 Kubernetes 集成，可以为 Kubernetes 提供高性能、可扩展、高可靠的持久化存储。HBlock CSI 插件架构如下图所示。

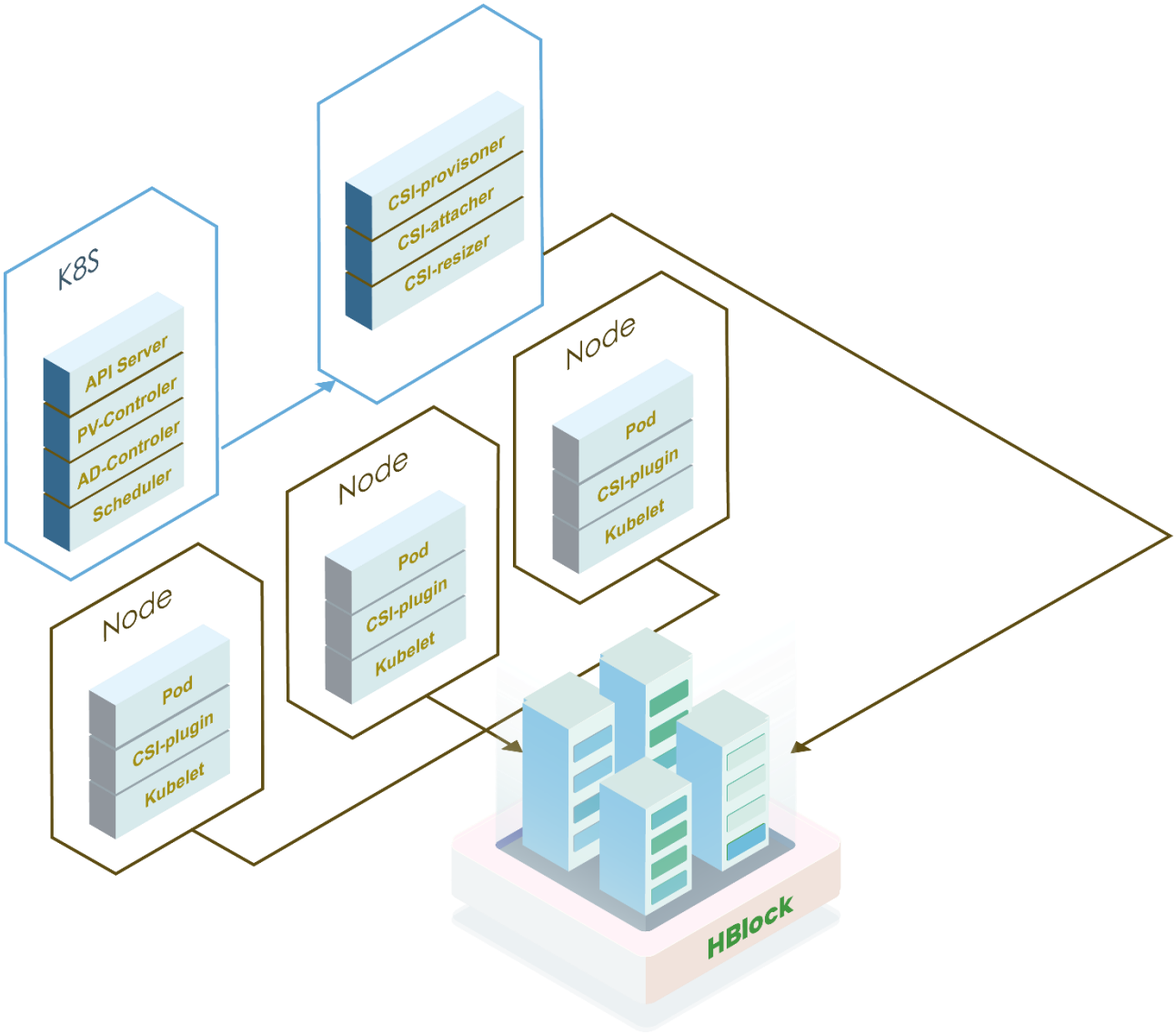

图1.HBlock CSI 插件架构图

HBlock CSI 插件运行在 CSI Pod 中，CSI Pod 通过此插件管理持久卷的生命周期，包括创建、删除、挂载和卸载持久卷等操作。每个持久卷在 HBlock 存储集群中对应一个 iSCSI LUN。当 Kubernetes 中的 Pod 需要使用该卷时，将通过 iSCSI 协议在 Pod 所在的物理节点上挂载该 LUN，Pod 使用 LUN 对应的数据目录进行读写。

HBlock CSI 插件具有以下特点：

- 稳定性强：HBlock CSI插件将存储和计算资源完全解耦，实现了资源的隔离。HBlock CSI插件以容器化方式部署，充分保证了运行的稳定性。
- 安全性高：对于需要配置在HBlock CSI插件中的敏感信息，支持使用加密方式配置，充分保证用户数据安全，不会泄露密码密钥等重要信息。
- 兼容性强：HBlock CSI插件以容器化方式部署，不依赖于操作系统版本的限制，可以运

行在多种操作系统上，消除环境差异带来的不确定性以及其他约束。

- 灵活性强：支持静态PV、动态PV、动态PVC等多种调用方法，用户可根据实际应用需要灵活创建、挂载存储卷。
  - 静态PV：即直接使用YAML文件创建的PV，适用于存储卷较少，并且不会频繁修改配置信息的场景。
  - 动态PV：即通过PVC创建的PV，无需提前创建PV，只要通过StorageClass把存储资源定义好，Kubernetes就会根据PVC动态创建PV。适用于存储卷较多，但存储卷的类型都相同的场景。
  - 动态PVC：通过StatefulSet中指定的StorageClass动态创建PVC，Kubernetes根据StorageClass中配置的信息，自动触发HBlock CSI插件创建HBlock 卷。适用于需动态创建多个Pod，并为其挂载存储卷的场景。

2 部署安装

2.1 环境要求

安装部署 HBlock 3.5.0 或以上版本，详细安装过程参见 HBlock 产品用户手册。

Kubernetes 集群的 node 节点如果需要挂载 HBlock 的卷来为 pod 提供持久化存储，需要卸载 iSCSI initiator 以及 MPIO，并重启 node 节点，才能够正常挂载。

2.2 运行环境

环境项目	版本
Kubernetes	V1.18 及以上
docker	19.03 及以上

2.3 安装驱动

HBlock CSI 可以使用脚本方式进行安装使用（**脚本方式使用指南**），也可以使用 HELM 方式进行安装使用（**HELM 方式使用指南**）。

3 脚本方式使用指南

3.1 安装

HBlock CSI 插件支持以下两种安装方式，用户可以根据情况选择其中一种进行安装：

- 逐台安装：适用于 Kubernetes 集群的节点数量不多的部署场景。
- Docker 私仓方式安装：适用于节点多的部署场景。

3.1.1 逐台安装

执行以下安装步骤（适用于 Kubernetes 集群的节点数量不多的部署场景）：

1. 在 Kubernetes master 和 node 上解压安装包。

```
unzip stor-csi-driver-1.3.0.zip
```

2. 在 Kubernetes master 和 node 上导入插件镜像。

```
cd stor-csi-driver-1.3.0
docker load < stor-csi-driver.tar
```

3. 在 Kubernetes master 节点执行部署脚本，完成插件的安装。

```
cd deploy
./deploy.sh
```

安装完成后，可以看到以下 pod、HBlock 的 plugin 以及 Sidecar 容器正常启动：

```
[root@server ~]# kubectl get pod | grep csi
csi-storplugin-controller-6c789569b9-nb2rm   3/3     Running   0          116s
csi-storplugin-node-9sj6z                    2/2     Running   0          116s
csi-storplugin-node-g8rwn                    2/2     Running   0          116s
```

3.1.2 docker 私仓方式安装

若节点数量较多，可以使用私仓方式安装 HBlock CSI 插件，避免在 Kubernetes 的所有节点上都导入插件。用户可先将插件镜像推送到私仓中，修改插件 YAML 文件的 image 地址为私仓中镜像地址，执行安装脚本即可完成安装。

在集群中任意一台服务器上执行下面的操作：

1. 解压安装包。

```
unzip stor-csi-driver-1.3.0.zip
```

2. 导入插件镜像。

```
cd stor-csi-driver-1.3.0
docker load < stor-csi-driver.tar
```

3. 推送镜像到私仓。

```
docker tag stor-csi-driver:1.3.0 xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
docker push xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
```

其中，xxx.xxx.xxx.xxx:5000 为私仓地址。

4. 查看私仓。

```
cat /etc/docker/daemon.json
```

5. 修改 YAML 镜像拉取地址。

修改 deploy/csi-storplugin-node.yaml 文件中 storplugin 对应 image 的 value 为 xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0。

```
  volumeMounts:
  - mountPath: /csi
   name: socket-dir
  - mountPath: /registration
   name: registration-dir
- name: storplugin
 #私仓镜像地址
 image: xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
```

6. 执行部署脚本

```
cd deploy
./deploy.sh
```

3.2 卸载

如果要卸载 HBlock CSI，建议卸载前清除 sc、statefulset、pod、pvc、pv，可以执行下列命令清除对应资源。对于 pod、pvc、pv，必须按照顺序 pod > pvc > pv 执行删除命令。

删除 sc：

```
kubectl delete sc storageclassname
```

删除 statefulset：

```
kubectl delete statefulset statefulsetname  --cascade=true
```

删除 pod：

```
kubectl delete pod podname
```

删除 pvc：

```
kubectl delete pvc pvcname
```

删除 pv：

```
kubectl delete pv pvname
```

执行卸载脚本./undeploy.sh 卸载 CSI。

```
cd deploy
./undeploy.sh
```

3.3 升级

如果要升级 HBlock CSI，保留要继续使用的 sc、statefulset、pod、pvc、pv，执行卸载脚本，再安装最新的 HBlock CSI 即可。

1. 执行卸载脚本./undeploy.sh 卸载 CSI。

```
cd deploy
./undeploy.sh
```

2. 下载最新 CSI 安装包，执行安装，具体安装步骤参照**安装**。

3.4 配置插件

3.4.1 配置 HBlock 访问地址

HBlock CSI 插件通过调用 HBlock 的 HTTP RESTful API 进行存储卷的管理操作，例如创建卷、删除卷、扩容卷等。需要配置插件访问 HBlock RESTful API 的 URL 地址和端口。

说明：支持对接多个 HBlock，包括：HBlock 单机版本、HBlock 集群版。

可以按照下列步骤配置 HBlock 访问地址。

1. 修改配置文件。

   修改 deploy/csi-plugin-conf/csi-configMap.yaml 配置文件，apiEndPointList 配置为 HBlock HTTP RESTful API 地址和端口，storProvider 配置为 HBlock。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-plugin-stor
data:
  config.json: |-
    [
      {
        "clusterID": "cluster1",
        "apiEndPointList": [
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx"
        ],
        "storProvider": "HBlock",
        "csiApiTimeout": csiApiTimeout
      },
      {
        "clusterID": "cluster2",
        "apiEndPointList": [
          "https://xx.xx.xx.xx:xx"
        ],
        "storProvider": "HBlock",
        "csiApiTimeout": csiApiTimeout
```

```
    },
    {
      "clusterID": "cluster3",
      "apiEndPointList": [
        "https://xx.xx.xx.xx:xx",
        "https://xx.xx.xx.xx:xx",
        "https://xx.xx.xx.xx:xx"
      ],
      "storProvider": "HBlock",
      "csiApiTimeout": csiApiTimeout
    }
  ]
```

参数

参数	描述	是否必填
clusterID	指定 HBlock 的标识，在 csi-configMap 中唯一。 取值：字符串形式，长度范围是 1~256，可以包含字母、数字、和短横线（-），字母区分大小写。	是
storProvider	HBlock 产品名称。 取值：HBlock。	是
apiEndPointList	HBlock 的服务器 IP 地址及 API 端口号；或者已经关联了 HBlock IP 和 API 端口号的 Kubernetes service 域名。	是
csiApiTimeout	指定 HBlock 创建 LUN 的等待时间，在等待时间内 LUN 创建失败，会报错，然后重试。 取值：正整数，默认值为 480，单位是秒。 **注意**：建议使用默认值。	否

示例：对接集群版 HBlock 和单机版 HBlock

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: csi-plugin-stor
data:
```

```
  config.json: |-
    [
      {
        "clusterID": "stor1",
        "apiEndPointList": [
          "https://192.168.0.192:1443",
          "https://192.168.0.110:1443",
          "https://192.168.0.102:1443"
        ],
        "storProvider": "HBlock",
        "csiApiTimeout": 480
      },
      {
        "clusterID": "stor2",
        "apiEndPointList": [
          "https://192.168.0.32:1443"
        ],
        "storProvider": "HBlock"
      }
    ]
```

2. 应用配置文件。

```
[root@server csi-plugin-conf]# kubectl apply -f csi-configMap.yaml

configmap/csi-plugin-stor configured
```

3.4.2 配置 HBlock 访问用户名和密码

HBlock CSI 插件调用 HBlock HTTP RESTful API 时，需要提供用户名和密码以进行签名认证，用户名为 HBlock 用户名，密码为 HBlock 的密码。

可以按照下列步骤配置 HBlock 访问用户名和密码。

(一) 修改配置文件

修改 deploy/csi-plugin-conf/csi-secret.yaml 配置文件中的参数。

```
apiVersion: v1
kind: Secret
metadata:
  name: csi-plugin-stor-secret
type: Opaque
data:
  userKey: userkey //对接 HBlock 的标识、用户名及密码的字符串的 base64 编码
```

userKey 源码：

```
[
  {
    "clusterID": "cluster1",
    "username": "storuser",
    "password": "YOUR_PASSWORD1"
  },
  {
    "clusterID": "cluster2",
    "username": "storuser",
    "password": "YOUR_PASSWORD2"
  }
]
```

参数

参数	描述	是否必填
userKey	对接 HBlock 的标识、用户名及密码的字符串的 base64 编码。	是
clusterID	csi-configMap.yaml 中配置的 HBlock 的标识。	是
username	HBlock 的管理员用户名。	是

password	HBlock 的管理员密码。	是

示例：

1. userKey 的源码如下：

```
[
      {
        "clusterID": "stor1",
        "username": "storuser",
        "password": "hblock12@"
      },
      {
        "clusterID": "stor2",
        "username": "storuser",
        "password": "hblock12@"
      }
    ]
```

2. 使用 Base64 工具对 userKey 源码进行编码。编码后的 userKey 如下：

```
WwogICAgICB7CiAgICAgICAgImNsdXN0ZXJJRCI6ICJzdG9yMSIsICAKICAgICAgICAidXNlcm5hbWUiOiAic3Rv
cnVzZXIiLAogICAgICAgICJwYXNzd29yZCI6ICJoYmxvY2sxMkAiCiAgICAgIH0sCiAgICAgIHsKICAgICAgICAi
Y2x1c3RlcklEIjogInN0b3IyIiwgIAogICAgICAgICJ1c2VybmFtZSI6ICJzdG9ydXNlciIsCiAgICAgICAgInBh
c3N3b3JkIjogImhibG9jazEyQCIKICAgICAgfQogICAgXQo=
```

3. 修改配置文件 deploy/csi-plugin-conf/csi-secret.yaml 配置文件中的参数。

```
apiVersion: v1
kind: Secret
metadata:
  name: csi-plugin-stor-secret
type: Opaque
data:
  userKey: WwogICAgICB7CiAgICAgICAgImNsdXN0ZXJJRCI6ICJzdG9yMSIsICAKICAgICAgICAidXNlcm5h
bWUiOiAic3RvcnVzZXIiLAogICAgICAgICJwYXNzd29yZCI6ICJoYmxvY2sxMkAiCiAgICAgIH0sCiAgICAgIHs
KICAgICAgICAiY2x1c3RlcklEIjogInN0b3IyIiwgIAogICAgICAgICJ1c2VybmFtZSI6ICJzdG9ydXNlciIsCi
AgICAgICAgInBhc3N3b3JkIjogImhibG9jazEyQCIKICAgICAgfQogICAgXQo=
```

(二) 应用配置文件。

```
[root@server csi-plugin-conf]# kubectl apply -f csi-secret.yaml
secret/csi-plugin-stor-secret configured
```

3.4.3 配置加密模式

对于配置文件中的一些敏感信息，可以采用加密模式进行配置。当使用的卷在 HBlock 中配置了质询握手认证协议（Challenge-Handshake Authentication Protocol，CHAP），则需要配置 CHAP 认证的用户名和密码。为了确保数据安全，建议采用加密模式来配置 CHAP 密码。

在使用 CHAP 方式时，输入的 chapUser 和 chapPassword 为加密字符串。加密字符串可以使用下列方法生成：使用 deploy/csi-plugin-conf/csi-secret-decrypt.yaml 中的 DecryptData 配置的密钥对原来的字符串进行 AES（ECP、paddingcs7）加密，加密后的结果进行 base64 编码。

可以按照下列步骤配置加密模式：

(一) 配置加密密钥。

修改配置文件 deploy/csi-plugin-conf/csi-secret-decrypt.yaml 中的 decryptFlag 和 decryptData 参数。

```
apiVersion: v1
kind: Secret
metadata:
  name: csi-plugin-stor-secret-decrypt
type: Opaque
data:
  #是否启用加密，配置为 true（启用）或 false（不启用）的 base64 编码字符串
  decryptFlag: decryptFlag
  #密钥的 base64 编码字符串，密钥长度必须为 16 位
  decryptData: decryptData    #密钥的 base64 编码
```

参数

参数	描述	是否必填
decryptFlag	是否启用加密模式，配置为 true（启用）或 false（不启用）的 Base64 编码字符串。 取值： ● dHJ1ZQ==：启用加密，true 的 Base64 编码。 ● ZmFsc2U=：不启用加密，false 的 Base64 编码。	是
decryptData	加密的密钥。 **说明**：启用加密，此参数必填。	条件

	取值：源码为 16 位的字符串。需要对源码进行 Base64 编码。	

示例：

1. 启用加密模式。例如 decryptFlag 的源码为 true，decryptData 的源码为 stor012345678901。
2. 使用 Base64 工具对 decryptFlag、decryptData 的源码进行编码。

   对 decryptFlag 源码进行 Base64 编码，编码后的 decryptFlag 如下：

```
dHJ1ZQ==
```

   对 decryptData 源码进行 Base64 编码，编码后的 decryptData 如下：

```
c3RvcjAxMjM0NTY3ODkwMQ==
```

3. 修改配置文件 deploy/csi-plugin-conf/csi-secret-decrypt.yaml 配置文件中的参数。

```
apiVersion: v1
kind: Secret
metadata:
  name: csi-plugin-stor-secret-decrypt
type: Opaque
data:
  decryptFlag: dHJ1ZQ==
  decryptData: c3RvcjAxMjM0NTY3ODkwMQ==
```

(二) 应用配置文件 csi-secret-decrypt.yaml。

```
[root@server csi-plugin-conf]# kubectl apply -f csi-secret-decrypt.yaml
secret/csi-plugin-stor-secret-decrypt configured
```

(三) 加密配置项。

以静态 PV 中 PV YAML 配置文件（csi-pv-local.yaml）为例，加密 CHAP 认证的密码，采用 AES 128 位加密，填充 PKCS7 的 Padding 加密 CHAP 密码，并对加密后的结果进行 Base64 编码。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: csi-pv-test
  labels:
```

```
    app: stor-pv-test
spec:
  capacity:
     storage: 1Gi
  volumeMode: Filesystem #卷模式，支持 Filesystem 和 Block 模式，默认为 Filesystem 模式。
  accessModes: # 访问模式
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
   driver: stor.csi.k8s.io
   volumeHandle: stor:lun1
   volumeAttributes:
     fsType: xfs
     readOnly: "false"
     #是否使用 CHAP 验证
     chapEnable: "true"
     chapUser: username
     chapPassword: password
```

chapUser 和 chapPassword 加密步骤：

（1）使用 AES 加密工具。“工作模式”为“ECB”，“填充模式”为“PKCS7”，“密钥长度”为“AES-128”。

（2）“加密密钥”使用 DecryptData 配置的密钥字符串(对应步骤 1 中的 decryptData 的源码，如步骤 1 举例中使用的源码为 stor012345678901)。

（3）输入加密文本（chapUser 或 chapPassword 的源码），点击“开始 AES 加密文本”。下图中为 chapUser=csi123 的加密方式，chapPassword 按下图方式加密即可。

图2.chapUser 加密

3.4.4 配置 Multipath

HBlock 采用分布式多控架构，单 LUN 可以通过多个 Target IQN 进行连接。

确保文件 deploy/csi-plugin-conf/multipath.conf 中的 vendor 为 CTYUN。

```
# This is a basic configuration file with some examples, for device mapper
# multipath.
#
# For a complete list of the default configuration values, run either
# multipath -t
# or
# multipathd show config
#
# For a list of configuration options with descriptions, see the multipath.conf
# man page

## Use user friendly names, instead of using WWIDs as names.
defaults {
        user_friendly_names yes
        find_multipaths yes
        uid_attribute "ID_WWN"
        #verbosity   6
}
devices {
    device {
        vendor                  "CTYUN"
        product                 "iSCSI LUN Device"
        path_grouping_policy    failover
        path_checker            tur
        path_selector           "round-robin 0"
        hardware_handler        "1 alua"
        rr_weight               priorities
        no_path_retry           queue
        prio                    alua
    }
}
```

3.5 调用方式

examples 目录中包含了使用 HBlock CSI 插件的 YAML 文件示例，用户可以根据卷模式（filesystem 或 block）修改其中的参数，即可在 Kubernetes 集群中使用 HBlock 创建的卷。

各举例路径如下：

```
[root@server stor-csi-driver-1.3.0]# cd examples
[root@server examples]# tree
.
├── block-volumes
│   ├── dynamic-pv
│   │   ├── csi-app-local-pvc-block.yaml
│   │   ├── csi-pvc-local-block.yaml
│   │   └── csi-storageclass-local.yaml
│   ├── statefulset
│   │   ├── csi-app-stateful-local-block.yaml
│   │   └── csi-storageclass-local-stateful-block.yaml
│   └── static-pv
│       ├── csi-app-local-pv-block.yaml
│       ├── csi-pvc-local-nocreate-block.yaml
│       └── csi-pv-local-block.yaml
└── filesystem-volumes
    ├── dynamic-pv
    │   ├── local
    │   │   ├── csi-app-local-pvc.yaml
    │   │   ├── csi-pvc-local.yaml
    │   │   └── csi-storageclass-local.yaml
    │   ├── local-chap
    │   │   ├── csi-app-local-pvc-chap.yaml
    │   │   ├── csi-pvc-local-chap.yaml
    │   │   └── csi-storageclass-local-chap.yaml
    │   └── local-chap-decrypt
    │       ├── csi-app-decrypt-local-pvc-chap.yaml
    │       ├── csi-pvc-local-chap-decrypt.yaml
    │       └── csi-storageclass-local-chap-decrypt.yaml
    ├── statefulset
    │   ├── csi-app-stateful-local.yaml
```

```
    |    └──── csi-storageclass-local-stateful.yaml
    └──── static-pv
        ├──── csi-app-local-pv.yaml
        ├──── csi-pvc-local-nocreate.yaml
        └──── csi-pv-local.yaml
```

3.5.1 静态 PV

静态 PV 是指直接使用 YAML 文件创建的 PV，适用于存储卷较少，并且不会频繁修改配置信息的场景。使用静态 PV 时，需要在 HBlock 中先创建好卷，然后创建 PV、PVC 和 Pod，HBlock CSI 插件可自动格式化，挂载 HBlock 的卷。

使用静态 PV 的主要流程如下：

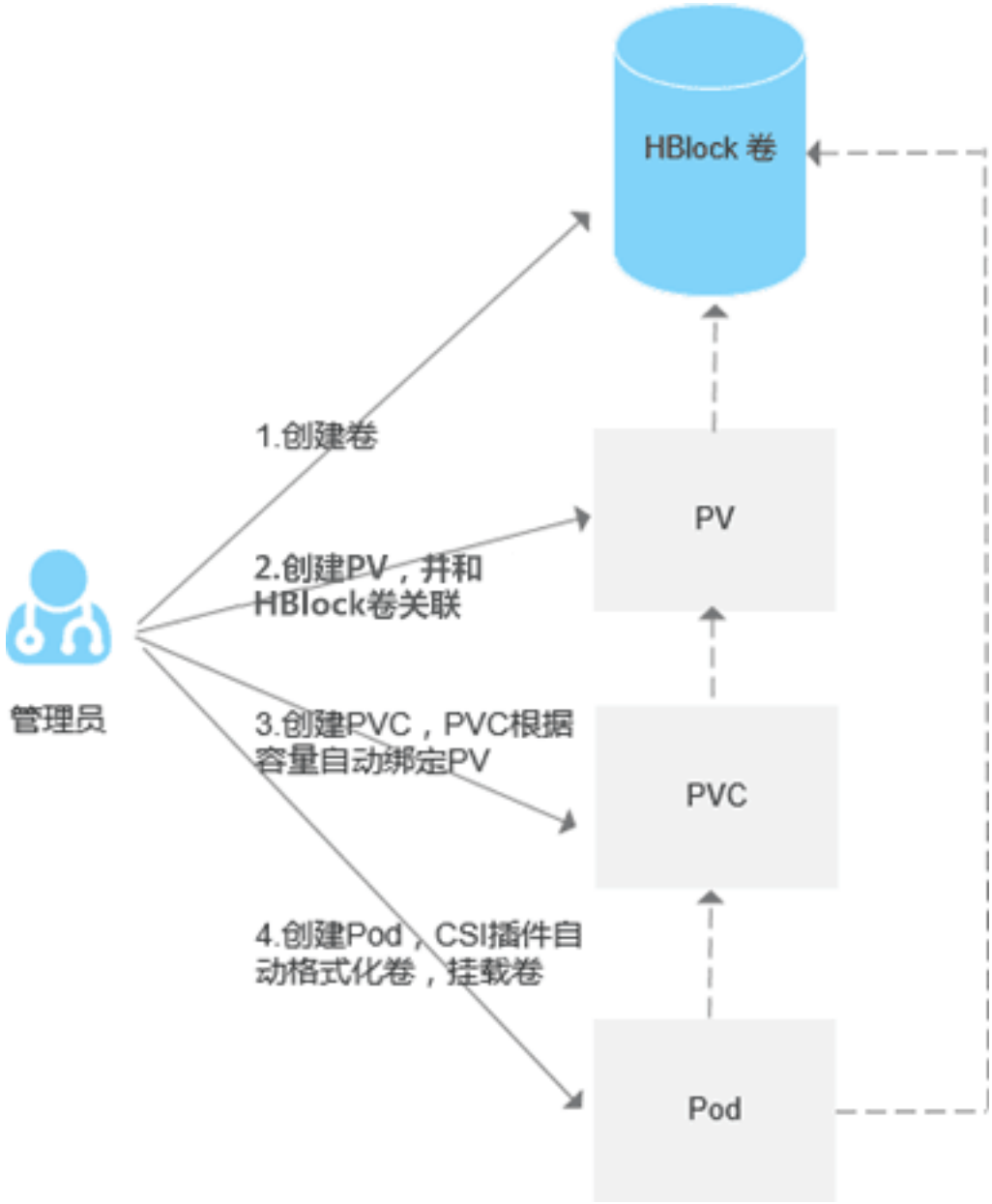

图3.静态 PV 流程图

1. 创建卷：创建 HBlock 卷的步骤请参考 HBlock 用户手册。

2. 创建 PV。

（1）新建 PV 的 YAML 配置文件

- 卷模式为 filesystem 的示例，创建 PV csi-pv-local-stor1lun06a。参考 examples\filesystem-volumes\static-pv\csi-pv-local.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolume
metadata:  #元数据
  name: csi-pv-local-stor1lun06a  #PV 的名称
  labels: #标签
    app: stor-pv-nocreate-stor1lun06a #PV 的标识
spec:
  capacity:
     storage: 66Gi  #卷容量，单位为 GiB，此处配置的卷容量应等于 HBlock 中创建的卷容量
  volumeMode: Filesystem  #卷模式，支持 Filesystem 和 Block 模式，默认为 Filesystem 模式。
  accessModes:
# 访问模式，Filesystem 模式的卷支持 ReadWriteOnce，ReadOnlyMany（卷需要提前格式化）
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain  #持久化卷回收策略，支持 Retain 和 Delete
  csi:
    driver: stor.csi.k8s.io
    volumeHandle: "stor1:lun06a" #指定 HBlock 卷
    volumeAttributes:
      fsType: xfs   # 卷被挂载到容器的文件系统类型，支持 xfs，ext4
      readOnly: "false"
      isMultipath: "true"   # 是否使用 multipath
      chapEnable: "false"  #是否使用 CHAP 认证
      chapUser: "username"
      chapPassword: "password"
```

- 卷模式为 Block，创建 PV csi-pv-local-block-stor2lunb1。参考 examples\block-volumes\static-pv\csi-pv-local-block.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolume
metadata:  #元数据
  name: csi-pv-local-block-stor2lunb1    #PV 的名称
  labels: #标签
```

```
    app: csi-pv-local-block-stor2lunb1
spec:
  capacity:
     storage: 22Gi     #卷容量，单位为 GiB，此处配置的卷容量应等于 HBlock 中创建的卷容量
  volumeMode: Block #卷模式，支持 Filesystem 和 Block 模式，默认为 Filesystem 模式。
  accessModes: # 访问模式，Block 模式的卷支持 ReadWriteOnce、ReadOnlyMany、ReadWriteMany
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain # 持久化卷回收策略，支持 Retain 和 Delete
  csi:
   driver: stor.csi.k8s.io  #HBlock CSI 插件
   volumeHandle: "stor2:lunb1"   #指定 HBlock 卷
   volumeAttributes:
     readOnly: "false"   #是否以只读方式挂载卷
     isMultipath: "false"   # 是否使用 Multipath，HBlock 单机版时，取值 false
     chapEnable: "false"  #是否使用 CHAP 认证
```

PV 的 YAML 配置文件参数描述

参数	描述	是否必填
driver	HBlock CSI 插件。取值：stor.csi.k8s.io。	是
accessModes	访问模式。 取值： ● ReadWriteOnce：卷可以被一个节点以读写的方式挂载。 ● ReadOnlyMany：卷可以被多个节点以只读方式挂载。filesystem 模式下，卷需要提前格式化。 ● ReadWriteMany：卷可以被多个节点以读写方式挂载。仅 Block 模式的卷支持。	是
volumeHandle	指定具体的 HBlock 卷名称，格式为 *clusterID*:*lunName*。 *clusterID*：指定 HBlock 的标识，在 csi-	是

	configMap.yaml 中唯一。详见**配置 HBlock 访问地址**。 *lunName*：HBlock 中创建的卷名称。	
volumeAttributes.fsType	卷被挂载到容器的文件系统类型，支持 xfs，ext4。 **说明**：卷模式为 filesystem 时必填。	条件
volumeAttributes.readOnly	是否以只读方式挂载卷。 取值： ● "true"。 ● "false"。 默认值为"false"。 **注意**：这里需要输入字符串，即"true"或"false"。	否
volumeAttributes.isMultipath	是否使用 Multipath。 取值： ● "true"。 ● "false"。 默认值为"true"。 **注意**： ● 这里需要输入字符串，即"true"或"false"。 ● 如果 HBlock 集群版使用的 HBlock 卷没有启用高可用模式，即 highAvailability 为 Disabled，此处需要设置为"false"，否则会导致 pod 启动失败。 ● 如果是 HBlock 单机版，此处需要设置为"false"。	条件
volumeAttributes.chapEnable	是否使用 CHAP 认证，取值： ● "true"。 ● "false"。 默认值为"false"。	否

	注意：这里需要输入字符串，即"true"或"false"。	
volumeAttributes.chapUser	CHAP 认证的用户名。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的用户名使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 3~64，只能由字母、数字、句点(.)、短横线(-)、下划线(_)、冒号(:)组成，字母区分大小写，且仅支持以字母或数字开头。	否
volumeAttributes.chapPassword	CHAP 认证的密码。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的密码使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 12~16，必须包含大写字母、小写字母、数字、下划线(_)中的至少两种字符，字母区分大小写。	否

注意：

- 如要使用加密方式配置 CHAP 密码，请参考**配置加密模式**。
- PV 的回收策略告诉集群，在 PV 被释放之后集群应该如何处理该 PV。当前，PV 可以被 Retain（保留）、 Recycle（再利用）或者 Delete（删除）。HBlock CSI 插件目前仅支持 Retain 和 Delete，不支持 Recycle。
  - 回收策略 Retain 使得用户可以手动回收资源。当 PersistentVolumeClaim 对象被删除时，PersistentVolume 卷仍然存在，对应的数据卷被视为“已释放（released）”。由于卷上仍然存在着前一申领人的数据，该卷还不能用于其他申领。管理员需要手动回收该卷。

- 对于回收策略为 Delete 的卷配置，删除动作会将 PersistentVolume 对象从 Kubernetes 中移除，同时也会从外部基础设施中移除所关联的存储资产。动态供应的卷会继承其 StorageClass 中设置的回收策略，该策略默认为 Delete。管理员需要根据用户的期望来配置 StorageClass。注意，如果 PV 中使用的是 HBlock 已经提前创建的卷，则不能通过 Kubernetes 来删除该卷。

（2）应用配置文件（以 csi-pv-local-stor1lun06a.yaml 为例）。

```
[root@server test]# kubectl apply -f csi-pv-local-stor1lun06a.yaml
persistentvolume/csi-pv-local-stor1lun06a created
```

（3）验证创建的 PV（以 csi-pv-local.yaml 为例）。

```
[root@server test]# kubectl get pv
NAME                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM    STORAGECLASS   REASON   AGE
csi-pv-local-stor1lun06a   66Gi       RWO            Retain           Available                                    20s
```

3. 创建 PVC

（1） 新建 PVC 的 YAML 配置文件。

- 卷模式为 filesystem，新建 PVC csi-pvc-local-nocreate-stor1lun06a 的 YAML 配置文件。参考 examples\filesystem-volumes\static-pv\csi-pvc-local-nocreate.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-local-nocreate-stor1lun06a  # PVC 的名字
spec:
  accessModes:  #访问模式
  - ReadWriteOnce
  resources:
    requests:
      storage: 66Gi   #卷的容量，单位 GiB，Kubernetes 会尝试绑定大于等于该容量的 PV
  selector:
    matchLabels:
```

```
      app: stor-pv-nocreate-stor1lun06a
```

- 卷模式为 Block，新建 PVC csi-pvc-local-nocreate-block-stor2lunb1 的配置文件。参考 examples\block-volumes\static-pv\csi-pvc-local-nocreate-block.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-local-nocreate-block-stor2lunb1
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 22Gi
  selector:
    matchLabels:
      app: csi-pv-local-block-stor2lunb1
```

（2） 应用配置文件（以 PVC csi-pvc-local-nocreate-stor1lun06a.yaml 为例）。

```
[root@server test]# kubectl apply -f csi-pvc-local-nocreate-stor1lun06a.yaml
persistentvolumeclaim/csi-pvc-local-nocreate-stor1lun06a created
```

（3） 验证已经创建的 PVC（以 csi-pvc-local-nocreate-stor1lun06a 为例）。

```
[root@server test]# kubectl get pvc
NAME                                 STATUS        VOLUME                      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-local-nocreate-stor1lun06a   Bound         csi-pv-local-stor1lun06a    66Gi       RWO                           10m
```

注意：

- PVC 通过容量自动匹配 PV，当 PV 的容量大于等于 PVC 的容量时，Kubernetes 会将 PVC 和 PV 进行绑定。如果有多个 PV 都满足条件，会选择第一个 PV 进行匹配。
- PV 和 PVC 配置文件中如有 selector 字段，那么 app 必须配置一致，否则无法绑定。如没有 selector 字段，PVC 会根据容量匹配 PV。
- PVC 和 PV 绑定后，PV 无法删除，如需删除 PV，需要先删除绑定的 PVC。

4. 创建 Pod。

创建 Pod，并和 PVC 关联。HBlock CSI 插件将自动完成格式化卷（如果未格式化），挂载卷。

（1） 新建 Pod 的 YAML 配置文件。

- 卷模式为 filesystem，创建 Pod my-csi-app-local-pv-stor1lun06a 的配置文件，参考 examples\filesystem-volumes\static-pv\csi-app-local-pv.yaml 中的示例。

```
kind: Pod
apiVersion: v1
metadata:
  name: my-csi-app-local-pv-stor1lun06a     #创建的 Pod 名称
spec:
  containers:
    - name: my-frontend    #容器名称
      image: busybox  #请替换为容器使用的镜像
      imagePullPolicy: "IfNotPresent"  #容器镜像的拉取策略
      volumeMounts:
      - mountPath: "/test6a"       # 卷挂载到容器的目标路径
        name: lun06a     # 对应 volumes 标签下的资源名
      command: [ "sleep", "1000000" ]
  volumes:
    - name: lun06a    # volumes 资源名，可以在 volumeMounts 下挂载
      persistentVolumeClaim:
        claimName: csi-pvc-local-nocreate-stor1lun06a    # Pod 指定使用的 PVC 名称
```

- 卷模式为 Block，创建 Pod csi-app-local-pv-block-stor2lunb1 的配置文件 csi-app-local-pv-stor1lun06a.yaml，参考 examples\block-volumes\static-pv\csi-app-local-pv-block.yaml 中的示例。

```
kind: Pod
apiVersion: v1
metadata:
  name: csi-app-local-pv-block-stor2lunb1
spec:
  containers:
    - name: lunb1
      image: busybox
```

```
      imagePullPolicy: "IfNotPresent"
      volumeDevices:
      - devicePath: "/dev/testb1"
        name: lunb1
      command: [ "sleep", "1000000" ]
  volumes:
    - name: lunb1
      persistentVolumeClaim:
        claimName: csi-pvc-local-nocreate-block-stor2lunb1
```

（2） 应用配置文件（以 csi-app-local-pv-stor1lun06a.yaml 为例）。

```
[root@server test]# kubectl apply -f csi-app-local-pv-stor1lun06a.yaml
pod/my-csi-app-local-pv-stor1lun06a created
```

（3） 验证 Pod 中挂载的卷。

```
[root@server test]# kubectl get pod|grep my-csi-app-local-pv-stor1lun06a
my-csi-app-local-pv-stor1lun06a              1/1     Running   0          92s
```

可以看到容器中已经挂载了路径/test6a，此路径对应 HBlock 中的卷 lun06a。

```
[root@server ~]# kubectl exec -it my-csi-app-local-pv-stor1lun06a -- /bin/sh
/ # ls
bin     dev     etc     home    lib     lib64   proc    root    sys     test6a  tmp     usr     var
```

3.5.2 动态 PV（静态 PVC）

动态 PV 是通过 PVC 创建的 PV，用户不需要提前创建 PV，只要通过 StorageClass 把存储资源定义好，Kubernetes 就会根据 PVC 动态创建 PV，自动触发 HBlock CSI 插件动态创建 HBlock 的卷。动态 PV 适用于存储卷较多，但存储卷的类型都相同的场景。

使用动态 PV 的主要流程如下：

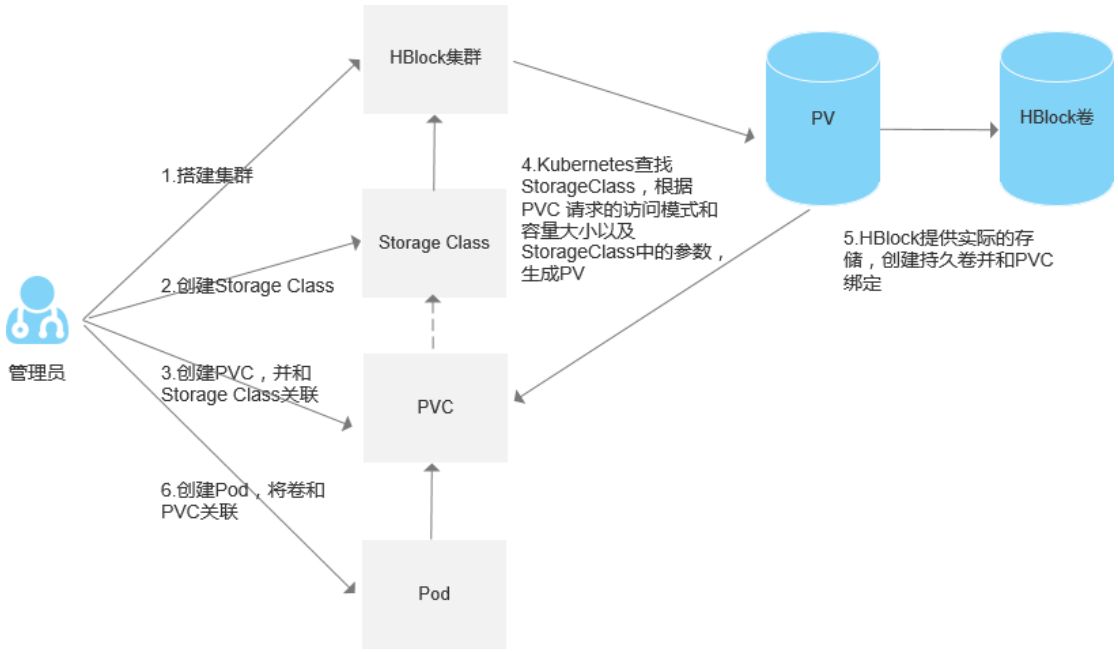

图4.动态 PV 流程图

1. 创建 StorageClass

（1） 新建 StorageClass 的 YAML 配置文件。

卷模式为 filesystem，创建 StorageClass csi-stor-sc-local-stor1lun08 的 YAML 配置文件 csi-storageclass-local-stor1lun08.yaml。可以参考 examples\filesystem-volumes\dynamic-pv\local\csi-storageclass-local.yaml 中的示例。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-stor-sc-local-stor1lun08  #StorageClass 名称
provisioner: stor.csi.k8s.io #HBlock CSI 插件
parameters:
  storageMode: Local  # HBlock 卷的存储类型
  fsType: xfs  # 卷被挂载到容器的文件系统类型，支持 xfs，ext4
  readOnly: "false"  # 是否以只读方式挂载卷
```

```
  sectorSize: "4096"   # HBlock 中卷的扇区大小，支持 512,4096，单位字节
  localStorageClass: "EC 2+1"   # HBlock 卷冗余模式
  minReplica: "2" # 最小副本数（仅 HBlock 集群版支持）。
  highAvailability: "ActiveStandby"   # HBlock 卷高可用模式。
  writePolicy: "WriteBack"  # HBlock 卷写策略
  isMultipath: "true"   # 是否启动多控
  maxSessions: "1" # iSCSI Target 允许建立的最大会话数。
  ECfragmentSize: "16" #纠删码模式分片大小。卷冗余模式为 EC 模式时，此设置才生效，否则忽略。
  serverNumbers: "2" # HBlock Target 所在的服务器数量（仅集群版支持）。
  clusterID: "stor1"  #HBlock 标识
reclaimPolicy: Delete  # PV 的回收策略，支持 Retain 和 Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true  # 是否允许扩展卷
```

卷模式为 Block，创建 StorageClass csi-stor-sc-local-stor1lun09 的 YAML 配置文件 csi-storageclass-local-stor1lun09.yaml。可以参考 examples\block-volumes\dynamic-pv\csi-storageclass-local.yaml 中的示例。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-stor-sc-local-stor1lun09  #StorageClass 名称
provisioner: stor.csi.k8s.io #HBlock CSI 插件
parameters:
  storageMode: Local   #HBlock 卷的存储类型
  readOnly: "false"    #是否以只读方式挂载卷
  sectorSize: "4096"   #HBlock 中卷的扇区大小，支持 512,4096，单位字节
  localStorageClass: "EC 2+1"   #HBlock 卷冗余模式
  minReplica: "2" # 最小副本数（仅 HBlock 集群版支持）。
  highAvailability: "ActiveStandby"   #HBlock 卷高可用模式。
  writePolicy: "WriteBack"    #HBlock 卷写策略
  isMultipath: "true"  #是否启动多控
  maxSessions: "1" # iSCSI Target 允许建立的最大会话数。
  ECfragmentSize: "16" #纠删码模式分片大小。卷冗余模式为 EC 模式时，此设置才生效，否则忽略。
  serverNumbers: "2" # HBlock Target 所在的服务器数量（仅集群版支持）。
  clusterID: "stor1" #HBlock 标识
reclaimPolicy: Delete   #PV 的回收策略，支持 Retain 和 Delete
```

```
volumeBindingMode: Immediate
allowVolumeExpansion: true  #是否允许扩展卷
```

StorageClass 的 YAML 配置文件参数：

参数	描述	是否必填
storageMode	卷的存储类型，支持 Local、Cache、Storage 模式。 Cache、Storage 模式表示上云卷。	是
fsType	卷被挂载到容器的文件系统类型，支持 xfs，ext4。 **说明**：卷模式为 filesystem 时必填。	条件
readOnly	是否以只读模式进行卷挂载。 取值："true"、"false"。默认值为"false"。 **注意**：这里需要输入字符串，即"true"或"false"。	是
cloudBucketName	已存在的 OOS 存储桶的名称。 **注意**：请勿开启 Bucket 的生命周期设定和合规保留。 类型：字符串。	上云卷必填
cloudPrefix	设置 OOS 中的前缀名称，设置前缀名称后，卷数据会存在存储桶以前缀命名的类文件夹中。如果未指定前缀，则直接存储在以卷名称命名的类文件夹中。 类型：字符串。 取值：长度范围是 1~256。	否
cloudAccessKey	OOS AccessKeyID。 类型：字符串。	上云卷必填
cloudSecretKey	OOS SecretAccessKey。 如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 secretKey 源码使用 DecryptData 配置的密钥对进行 AES(ECP、paddingcs7)加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 类型：字符串。	上云卷必填

cloudEndpoint	设置 OOS Endpoint。 类型：字符串。	上云卷必填
cloudObjectSize	数据存储在 OOS 中的大小。 取值：128、256、512、1024、2048、4096、8192，单位是 KiB。默认值为 1024。	否
cloudStorageClass	设置 OOS 的存储类型。 取值： ● STANDARD：标准存储。 ● STANDARD_IA：低频访问存储。 默认值为 STANDARD。	否
cloudCompression	是否压缩数据上传至 OOS。 取值： ● Enabled：是。 ● Disabled：否。 默认值为 Enabled。	否
cloudSignVersion	OOS 的请求签名认证方式 取值： ● v2：v2 签名。 ● v4：v4 签名。 默认值为 v2。	否
cloudRegion	表示 Endpoint 资源池所在区域。 V4 签名时，此项必填。 类型：字符串。	条件
deleteCloudData	删除卷时，是否删除云上的数据。 取值： ● true：删除云上的数据。 ● false：不删除云上的数据。 默认值为 false。	否

sectorSize	扇区大小。根据客户端文件系统 I/O 操作的最小单位设定卷扇区大小。 取值："512"、"4096"，单位为字节。默认值为"4096"。	否
localStorageClass	本地存储冗余模式。单机版不能设置此参数。 取值： ● single-copy：单副本； ● 2-copy：两副本； ● 3-copy：三副本； ● EC *N*+*M*：纠删码模式。其中 N、M 为正整数，N>M，且 N+M<=128。表示将数据分割成 N 个片段，并生成 M 个校验数据。 默认值为 EC 2+1。	否
minReplica	最小副本数（仅集群版支持）。 对于副本模式的卷，假设卷副本数为 X，最小副本数为 Y（Y 必须≤X），该卷每次写入时，至少 Y 份数据写入成功，才视为本次写入成功。对于 EC *N*+*M* 模式的卷，假设该卷最小副本数设置为 Y（必须满足 N≤Y≤N+M），必须满足总和至少为 Y 的数据块和校验块写入成功，才视为本次写入成功。 取值：整数。对于副本卷，取值范围是[1, N]，N 为副本模式卷的副本数，默认值为 1。对于 EC 卷，取值范围是[N, N+M]，默认值为 N。	否
redundancyOverlap	卷的折叠副本数（仅集群版支持）。在数据冗余模式下，同一份数据的不同副本/分片默认分布在不同的故障域，当故障域损坏时，允许根据卷的冗余折叠原则，将多份数据副本放在同一个故障域中，但是分布在不同的 path 上。 **注意**：如果存储池故障域级别为 path，此参数不生效。 取值：对副本模式，取值范围是[1，副本数]，默认值为 1；	否

	对于 EC 模式，取值范围是[1，M+N]，默认值为 1。	
ECfragmentSize	纠删码模式分片大小。卷冗余模式为 EC 模式时，此设置才生效，否则忽略。 取值：1、2、4、8、16、32、64、128、256、512、1024、2048、4096，单位是 KiB。默认值为 16。	否
highAvailability	是否选择高可用模式。单机版不能设置此参数。 取值： ● ActiveStandby：启动主备，该卷关联对应 Target 下的所有 IQN。 ● Disabled：禁用高可用模式，该卷关联对应 Target 下的一个 IQN。 默认值为 ActiveStandby。	否
writePolicy	卷的写策略。 取值： ● WriteBack：回写，即数据写入到内存后，立刻返回给客户端写成功，之后再异步写入磁盘。适用于对性能要求较高，稳定性要求不高的场景。 ● WriteThrough：透写，即数据同时写入内存和磁盘，并在两处都写成功后，再返回客户端写成功。适用于稳定性要求较高，写性能要求不高，且最近写入的数据会较快被读取的场景。 ● WriteAround：绕写，即数据写入磁盘后即释放相应内存，写入磁盘成功后，立刻返回客户端写成功。适用于稳定性要求较高，性能要求不高，且写多读少的场景。 默认值为 WriteBack。	否
isMultipath	是否使用 Msultipath。 取值： ● "true"。	条件

	● "false"。 默认值为"true"。 **注意：** ● 这里需要输入字符串，即"true"或"false"。 ● 如果 HBlock 集群版使用的 HBlock 卷没有启用高可用模式，即 highAvailability 为 Disabled，此处需要设置为"false"，否则会导致 pod 启动失败。 ● 如果是 HBlock 单机版，此处需要设置为"false"。	
path	指定存储卷数据的数据目录（仅单机版支持）。 如果创建卷时不指定数据目录，使用服务器设置的默认数据目录。	否
pool	存储池名称，表示最终存储池，卷数据最终落在该存储池内（仅集群版支持）。 取值：长度范围是 1~16，只能由字母、数字和短横线（-）、下划线（_）组成，字母区分大小写，且仅支持以字母和数字开头。 默认使用基础存储池。	否
cachePool	存储池名称，表示高速缓存存储池，卷数据首先写入该存储池内（仅集群版支持）。 取值：长度范围是 1~16，只能由字母、数字和短横线（-）、下划线（_）组成，字母区分大小写，且仅支持以字母和数字开头。 如果不填写则代表不设置高速缓存存储池。 **注意：**存储池与缓存存储池不能是同一个存储池。	否
maxSessions	iSCSI Target 允许建立的最大会话数。 取值：整数，取值范围是[1, 1024]，默认值为 1。	否
serverNumbers	Target 所在的服务器数量（仅集群版支持）。 整数形式，取值为[2, *n*]，n 为集群内服务器的数量。默认值	否

	为 2。	
faultDomains	卷的服务端连接位置信息。根据存储池的故障域，创建 Target 所在服务器的列表（仅集群版支持），以便创建 LUN 时，LUN 关联的 Target 优先从该服务器列表中选择所在服务器。例如存储池为 rack 级别，其拓扑图涵盖 rack1、rack2、rack3、rack4 中的节点，且 faultDomains 指定 rack1、rack2，那么创建 LUN 时，LUN 关联的 Target 优先从 rack1、rack2 所包含的此存储池的服务器列表里进行选择，每个节点限选一个服务器。 **注意：** ● 存储池的故障域为 path 级别时，不能设置该参数。 ● 如果 LUN 指定了高速缓存池和最终存储池，则从高速缓存池池中选择节点列表。如果 LUN 只指定了最终存储池，则从最终存储池中选择节点列表。 **取值：**以节点的形式添加，节点的级别可以到 room、rack、server。可以指定多个节点，但是节点的个数要小于等于 serverNumbers。节点的级别可以不同，但节点名必须唯一。节点名称可以使用在集群拓扑中的全路径，格式为：*name*:*name*:*name*，从根节点开始逐级指定；也可以使用部分路径，但该路径需在集群拓扑中唯一。例如 default:room4:hblock_4、room4:hblock_4、hblock_4 在集群拓扑中均指向同一节点，且节点名在集群拓扑中唯一，则任选其一即可。	否
chapEnable	是否使用 CHAP 认证，取值： ● "true"。 ● "false"。 默认值为"false"。 注意：这里需要输入字符串，即"true"或"false"。	否

chapUser	CHAP 认证的用户名。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的用户名使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 3~64，只能由字母、数字、句点(.)、短横线(-)、下划线(_)、冒号(:)组成，字母区分大小写，且仅支持以字母或数字开头。	否
chapPassword	CHAP 认证的密码。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的密码使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 12~16，必须包含大写字母、小写字母、数字、下划线(_)中的至少两种字符，字母区分大小写。	否
clusterID	HBlock 的标识，在 csi-configMap.yaml 中唯一。详见**配置 HBlock 访问地址**。	是
reclaimPolicy	持久化卷回收策略。 取值： ● Retain：保留。 ● Delete：删除。 默认值为 Delete。	否
volumeBindingMode	立即绑定还是等待 Pod 调度时绑定。 取值： ● Immediate：立即绑定。 ● WaitForFirstConsumer：延迟绑定。 默认值为 Immediate。	否

allowVolumeExpansion	允许卷扩展。 取值： ● true：允许卷扩展。 ● false：不允许卷扩展。 默认值为 false。	否

（2） 应用配置文件（以 csi-storageclass-local-stor1lun08.yaml 为例）。

```
[root@server dynamic-pv]# kubectl apply -f  csi-storageclass-local-stor1lun08.yaml
storageclass.storage.k8s.io/csi-stor-sc-local-stor1lun08 created
```

（3） 验证创建的 StorageClass（以 csi-storageclass-local-lun09.yaml 为例）。

```
[root@server dynamic-pv]# kubectl get storageclass
NAME                          PROVISIONER       RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
csi-stor-sc-local-stor1lun08   stor.csi.k8s.io   Delete          Immediate           true                   24s
```

注意：如需加密配置 CHAP 用户名和密码，则 deploy/csi-plugin-conf/csi-secret-decrypt.yaml 文件中 decryptFlag 字段需配置为 true，且 CHAP 用户名和密码字段必须配置为 AES 密文的 base64 格式。具体可以参见**配置加密模式**。

2. 创建 PVC

创建 PVC，并和 StorageClass 关联。

（1） 新建 PVC 的 yaml 配置文件：

卷模式为 filesystem，创建 PVC csi-pvc-local-stor1lun08 的配置文件 csi-pvc-local-stor1lun08.yaml。参考 examples\filesystem-volumes\dynamic-pv\local\csi-pvc-local.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-local-stor1lun08  # PVC 的名字
spec:
  accessModes:  #访问模式，filesystem 模式的卷支持 ReadWriteOnce
  - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 77Gi  #卷容量，单位为 GiB
  storageClassName: csi-stor-sc-local-stor1lun08  #PVC 使用的 StorageClass 的名字
```

卷模式为 Block，创建 PVC csi-pvc-local-block-stor1lun09 的配置文件 csi-pvc-local-stor1lun09.yaml。参考 examples\block-volumes\dynamic-pv\csi-pvc-local-block.yaml 中的示例。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-local-block-stor1lun09   # PVC 的名字
spec:
  accessModes: # 访问模式，Block 模式的卷支持 ReadWriteOnce、ReadOnlyMany、ReadWriteMany
  - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 88Gi   #卷容量，单位为 GiB
  storageClassName: csi-stor-sc-local-stor1lun09 #PVC 使用的 StorageClass 的名字
```

（2） 应用配置文件（以 csi-pvc-local-stor1lun08.yaml 为例）。

```
[root@server dynamic-pv]# kubectl apply -f csi-pvc-local-stor1lun08.yaml
persistentvolumeclaim/csi-pvc-local-stor1lun08 created
```

（3） 验证已经创建的 PVC。

```
[root@server dynamic-pv]# kubectl get pvc
NAME                       STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS                AGE
csi-pvc-local-stor1lun08   Bound    pvc-84b9a4b9-403d-41a5-96bc-8d1e08c7ab15   77Gi       RWO            csi-stor-sc-local-stor1lun08   27s
```

3. 创建 Pod。

创建 Pod，并和 PVC 关联。HBlock CSI 插件将自动创建、格式化、挂载 HBlock 的卷。

（1） 新建 Pod 的 YAML 配置文件，

卷模式为 filesystem，创建 Pod my-csi-app-local-stor1lun08 的配置文件 csi-app-local-pvc-stor1lun08.yaml。可以参考 examples\filesystem-volumes\dynamic-pv\local\csi-app-local-pvc.yaml 中的示例。

```
kind: Pod
apiVersion: v1
metadata:
  name: my-csi-app-local-stor1lun08
spec:
  containers:
    - name: my-frontend
      image: busybox    # 镜像地址
      imagePullPolicy: "IfNotPresent"     # 拉取镜像策略
      volumeMounts:
      - mountPath: "/test8"    # 挂载到容器的目标路径
        name: lun08
      command: [ "sleep", "1000000" ]
  volumes:
    - name: lun08  # 对应 volumeMounts 的挂载项目
      persistentVolumeClaim:
        claimName: csi-pvc-local-stor1lun08    # 调用 pvc 的名字
```

卷模式为 Block，创建 Pod my-csi-app-local-block-stor1lun09 的配置文件 csi-app-local-pvc-stor1lun09.yaml。可以参考 examples\block-volumes\dynamic-pv\csi-app-local-pvc-block.yaml 中的示例。

```
kind: Pod
apiVersion: v1
metadata:
  name: my-csi-app-local-block-stor1lun09  #Pod 的名称
spec:
  containers:
    - name: my-frontend
      image: busybox # 镜像地址
      imagePullPolicy: "IfNotPresent" # 拉取镜像策略
      volumeDevices:
      - devicePath: "/dev/test9" # 挂载到容器的目标路径
        name: lun09    # 对应 volumes 中的 name
```

```
      command: [ "sleep", "1000000" ]
  volumes:
    - name: lun09  # 对应 volumeMounts 的挂载项目
      persistentVolumeClaim:
        claimName: csi-pvc-local-block-stor1lun09  # 调用 pvc 的名字
```

（2） 应用配置文件（以 csi-app-local-pvc-stor1lun08.yaml 为例）。

```
[root@server dynamic-pv]# kubectl apply -f csi-app-local-pvc-stor1lun08.yaml
pod/my-csi-app-local-stor1lun08 created
```

（3） 验证创建的 Pod（以 csi-app-local-pvc-storlun08.yaml 为例）。

```
[root@server test]# kubectl get pod|grep my-csi-app-local-stor1lun08
my-csi-app-local-stor1lun08                  1/1     Running   0          27s
```

可以看到容器中已经挂载了路径/test8。

```
[root@server ~]# kubectl exec -it my-csi-app-local-stor1lun08 -- /bin/sh
/ # ls
bin    dev    etc    home   lib    lib64  proc   root   sys    test8  tmp    usr    var
```

注意：HBlock CSI 插件根据用户的配置，在 HBlock 集群中自动创建卷和 Target，卷、Target、PV 是一一对应的关系。

3.5.3 动态 PVC

通过 StatefulSet 中指定的 StorageClass 动态创建 PVC，Kubernetes 根据 StorageClass 中配置的信息，自动触发 HBlock CSI 插件创建 HBlock 卷。动态 PVC 适用于需动态创建多个 Pod，并为其挂载存储的场景。

使用动态 PVC 的主要流程如下：

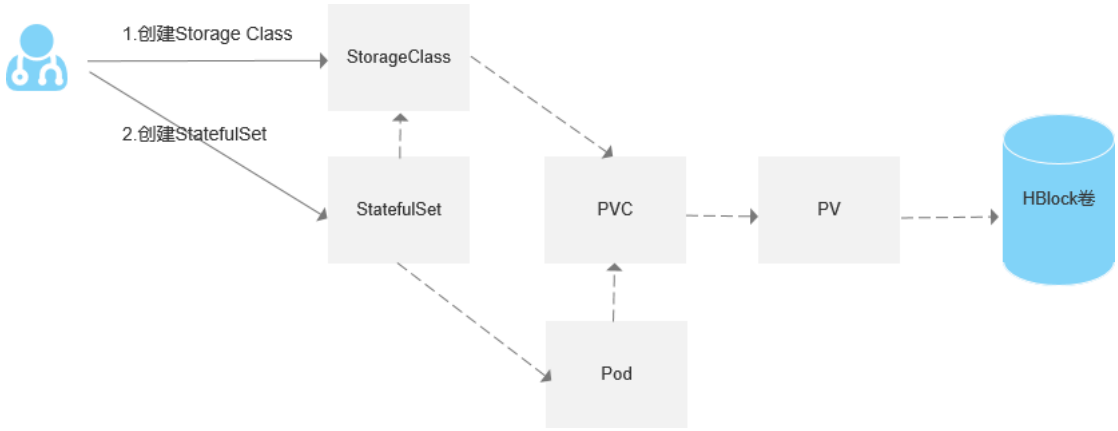

图9.动态 PV 流程图

1. 创建 Storageclass

（1） 新建 StorageClass 的 YAML 配置文件。

- 卷模式为 filesystem，创建 StorageClass csi-storageclass-local-stateful-stor1lun10 的配置文件 csi-storageclass-local-stateful-stor1lun10.yaml。可以参考 examples\filesystem-volumes\statefulset\csi-storageclass-local-stateful.yaml 中的示例。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-storageclass-local-stateful-stor1lun10   #StorageClass 名称
provisioner: stor.csi.k8s.io  #HBlock CSI 插件
parameters:
  storageMode: Local   #HBlock 卷的存储类型
  fsType: xfs   #挂载卷的文件系统的格式，支持 xfs、ext4
  readOnly: "false"   #是否以只读方式挂载卷
  sectorSize: "4096"  # HBlock 中卷的扇区大小，支持 512,4096，单位字节
```

```
  localStorageClass: "EC 2+1" # HBlock 卷冗余模式
  # minReplica 和 localStorageClass "EC N+M"中的 N 默认相等
  minReplica: "2"  # 最小副本数（仅 HBlock 集群版支持）
  highAvailability: "ActiveStandby" # HBlock 卷高可用模式
  writePolicy: "WriteBack"   # HBlock 卷写策略
  isMultipath: "true"  # 是否启动多控
  maxSessions: "1"  # iSCSI Target 允许建立的最大会话数。
  ECfragmentSize: "16"  #纠删码模式分片大小。卷冗余模式为 EC 模式时，此设置才生效，否则忽略。
  serverNumbers: "2"  # HBlock Target 所在的服务器数量（仅集群版支持）
  clusterID: "stor1"  #HBlock 标识
reclaimPolicy: Delete  #持久化卷回收策略，支持 Retain 和 Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true  #允许卷扩展
```

- 卷模式为 Block，创建 StorageClass csi-storageclass-local-stateful-block-stor1lun11 的配置文件 csi-storageclass-local-stateful-block-stor1lun11.yaml。可以参考 examples\block-volumes\statefulset\csi-storageclass-local-stateful-block.yaml。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-storageclass-local-stateful-block-stor1lun11
provisioner: stor.csi.k8s.io
parameters:
  storageMode: Local
  readOnly: "false"
  sectorSize: "4096"
  localStorageClass: "EC 2+1"
  # minReplica 和 localStorageClass "EC N+M"中的 N 默认相等
  minReplica: "2"
  highAvailability: "ActiveStandby"
  writePolicy: "WriteBack"
  isMultipath: "true"
  maxSessions: "1"
  ECfragmentSize: "16"
```

```
  serverNumbers: "2"
  clusterID: "stor1"
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

StorageClass 的 YAML 配置文件参数：

参数	描述	是否必填
storageMode	卷的存储类型，支持 Local、Cache、Storage 模式。 Cache、Storage 模式表示上云卷。	是
fsType	卷被挂载到容器的文件系统类型，支持 xfs，ext4。 **说明**：卷模式为 filesystem 时必填。	条件
readOnly	是否以只读模式进行卷挂载。 取值："true"、"false"。默认值为"false"。 **注意**：这里需要输入字符串，即"true"或"false"。	是
cloudBucketName	已存在的 OOS 存储桶的名称。 注意：请勿开启 Bucket 的生命周期设定和合规保留。 类型：字符串。	上云卷必填
cloudPrefix	设置 OOS 中的前缀名称，设置前缀名称后，卷数据会存在存储桶以前缀命名的类文件夹中。如果未指定前缀，则直接存储在以卷名称命名的类文件夹中。 类型：字符串。 取值：长度范围是 1~256。	否
cloudAccessKey	OOS AccessKeyID。 类型：字符串。	上云卷必填
cloudSecretKey	OOS SecretAccessKey。 如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 secretKey 源码使用 DecryptData 配置的密钥对进行 AES(ECP、paddingcs7)加密，加密后的结果进	上云卷必填

	行 Base64 编码，具体详见**配置加密模式**。 类型：字符串。	
cloudEndpoint	设置 OOS Endpoint。 类型：字符串。	上云卷必填
cloudObjectSize	数据存储在 OOS 中的大小。 取值：128、256、512、1024、2048、4096、8192，单位是 KiB。默认值为 1024。	否
cloudStorageClass	设置 OOS 的存储类型。 取值： ● STANDARD：标准存储； ● STANDARD_IA：低频访问存储。 默认值为 STANDARD。	否
cloudCompression	是否压缩数据上传至 OOS。 取值： ● Enabled：是； ● Disabled：否。 默认值为 Enabled。	否
cloudSignVersion	OOS 的请求签名认证方式 取值： ● v2：v2 签名。 ● v4：v4 签名。 默认值为 v2。	否
cloudRegion	表示 Endpoint 资源池所在区域。 使用 V4 签名时，此项必填。 类型：字符串。	条件
deleteCloudData	删除卷时，是否删除云上的数据。 取值： ● true：删除云上的数据。	否

	● false：不删除云上的数据。 默认值为 false。	
sectorSize	扇区大小。根据客户端文件系统 I/O 操作的最小单位设定卷扇区大小。 取值："512"、"4096"，单位为字节。默认值为"4096"。	否
localStorageClass	本地存储冗余模式。单机版不能设置此参数。 取值： ● single-copy：单副本。 ● 2-copy：两副本。 ● 3-copy：三副本。 ● EC *N*+*M*：纠删码模式。其中 N、M 为正整数，N>M，且 N+M<=128。表示将数据分割成 N 个片段，并生成 M 个校验数据。 默认值为 EC 2+1。	否
minReplica	最小副本数（仅集群版支持）。 对于副本模式的卷，假设卷副本数为 X，最小副本数为 Y（Y 必须≤X），该卷每次写入时，至少 Y 份数据写入成功，才视为本次写入成功。对于 EC *N*+*M* 模式的卷，假设该卷最小副本数设置为 Y（必须满足 N≤Y≤N+M），必须满足总和至少为 Y 的数据块和校验块写入成功，才视为本次写入成功。 取值：整数。对于副本卷，取值范围是[1, N]，N 为副本模式卷的副本数，默认值为 1。对于 EC 卷，取值范围是[N, N+M]，默认值为 N。	否
redundancyOverlap	卷的折叠副本数（仅集群版支持）。在数据冗余模式下，同一份数据的不同副本/分片默认分布在不同的故障域，当故障域损坏时，允许根据卷的冗余折叠原则，将多份数据副本放在同一个故障域中，但是分布在不同的	否

	path 上。 **注意**：如果存储池故障域级别为 path，此参数不生效。 取值：对副本模式，取值范围是[1，副本数]，默认值为 1；对于 EC 模式，取值范围是[1，M+N]，默认值为 1。	
ECfragmentSize	纠删码模式分片大小。卷冗余模式为 EC 模式时，此设置才生效，否则忽略。 取值：1、2、4、8、16、32、64、128、256、512、1024、2048、4096，单位是 KiB。默认值为 16。	否
highAvailability	是否选择高可用模式。单机版不能设置此参数。 取值： ● ActiveStandby：启动主备，该卷关联对应 Target 下的所有 IQN。 ● Disabled：禁用高可用模式，该卷关联对应 Target 下的一个 IQN。 默认值为 ActiveStandby。	否
writePolicy	卷的写策略。 取值： ● WriteBack：回写，即数据写入到内存后，立刻返回给客户端写成功，之后再异步写入磁盘。适用于对性能要求较高，稳定性要求不高的场景。 ● WriteThrough：透写，即数据同时写入内存和磁盘，并在两处都写成功后，再返回客户端写成功。适用于稳定性要求较高，写性能要求不高，且最近写入的数据会较快被读取的场景。 ● WriteAround：绕写，即数据写入磁盘后即释放相应内存，写入磁盘成功后，立刻返回客户端写成功。适用于稳定性要求较高，性能要求不高，且写多读少的场景。	否

	默认值为 WriteBack。	
isMultipath	是否使用 Multipath。 取值： ● "true"。 ● "false"。 默认值为"true"。 **注意：** ● 这里需要输入字符串，即"true"或"false"。 ● 如果 HBlock 集群版使用的 HBlock 卷没有启用高可用模式，即 highAvailability 为 Disabled，此处需要设置为"false"，否则会导致 pod 启动失败。 ● 如果是 HBlock 单机版，此处需要设置为"false"。	条件
path	指定存储卷数据的数据目录（仅单机版支持）。 如果创建卷时不指定数据目录，使用服务器设置的默认数据目录。	否
pool	存储池名称，表示最终存储池，卷数据最终落在该存储池内（仅集群版支持）。 取值：长度范围是 1~16，只能由字母、数字和短横线（-）、下划线（_）组成，字母区分大小写，且仅支持以字母和数字开头。 默认使用基础存储池。	否
cachePool	存储池名称，表示高速缓存存储池，卷数据首先写入该存储池内（仅集群版支持）。 取值：长度范围是 1~16，只能由字母、数字和短横线（-）、下划线（_）组成，字母区分大小写，且仅支持以字母和数字开头。 如果不填写则代表不设置高速缓存存储池。 **注意：**存储池与缓存存储池不能是同一个存储池。	否

maxSessions	iSCSI Target 允许建立的最大会话数。 取值：整数，取值范围是[1, 1024]，默认值为 1。	否
serverNumbers	Target 所在的服务器数量（仅集群版支持）。 整数形式，取值为[2, *n*]，n 为集群内服务器的数量。默认值为 2。	否
faultDomains	卷的服务端连接位置信息。根据存储池的故障域，创建 Target 所在服务器的列表（仅集群版支持），以便创建 LUN 时，LUN 关联的 Target 优先从该服务器列表中选择所在服务器。例如存储池为 rack 级别，其拓扑图涵盖 rack1、rack2、rack3、rack4 中的节点，且 faultDomains 指定 rack1、rack2，那么创建 LUN 时，LUN 关联的 Target 优先从 rack1、rack2 所包含的此存储池的服务器列表里进行选择，每个节点限选一个服务器。 **注意：** ● 存储池的故障域为 path 级别时，不能设置该参数。 ● 如果 LUN 指定了高速缓存池和最终存储池，则从高速缓存池池中选择节点列表。如果 LUN 只指定了最终存储池，则从最终存储池中选择节点列表。 **取值：**以节点的形式添加，节点的级别可以到 room、rack、server。可以指定多个节点，但是节点的个数要小于等于 serverNumbers。节点的级别可以不同，但节点名必须唯一。节点名称可以使用在集群拓扑中的全路径，格式为：*name*:*name*:*name*，从根节点开始逐级指定；也可以使用部分路径，但该路径需在集群拓扑中唯一。例如 default:room4:hblock_4、room4:hblock_4、hblock_4 在集群拓扑中均指向同一节点，且节点名在集群拓扑中唯一，则任选其一即可。	否
clusterID	HBlock 的标识，在 csi-configMap.yaml 中唯一。详见**配**	是

	置 HBlock 访问地址。	
chapEnable	是否使用 CHAP 认证，取值： ● "true"。 ● "false"。 默认值为"false"。 注意：这里需要输入字符串，即"true"或"false"。	否
chapUser	CHAP 认证的用户名。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的用户名使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 3~64，只能由字母、数字、句点(.)、短横线(-)、下划线(_)、冒号(:) 组成，字母区分大小写，且仅支持以字母或数字开头。	否
chapPassword	CHAP 认证的密码。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的密码使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 12~16，必须包含大写字母、小写字母、数字、下划线(_)中的至少两种字符，字母区分大小写。	否
reclaimPolicy	持久化卷回收策略。 取值： ● Retain：保留。 ● Delete：删除。 默认值为 Delete。	否
volumeBindingMode	立即绑定还是等待 Pod 调度时绑定。	否

	取值： ● Immediate：立即绑定。 ● WaitForFirstConsumer：延迟绑定。 默认值为 Immediate。	
allowVolumeExpansion	允许卷扩展。 取值： ● true：允许卷扩展。 ● false：不允许卷扩展。 默认值为 false。	否

（2）应用配置文件（以 csi-storageclass-local-stateful-stor1lun10.yaml 为例）。

```
[root@server statefulset]# kubectl apply -f csi-storageclass-local-stateful-stor1lun10.yaml
storageclass.storage.k8s.io/csi-storageclass-local-stateful-stor1lun10 created
```

2. 创建 StatefulSet。

（1） 新建 StatefulSet 的 YAML 配置文件。

卷模式为 filesystem，创建 StatefulSet csi-app-stateful-local-stor1lun10 的 YAML 配置文件 csi-app-stateful-local-stor1lun10.yaml。可以参考 examples\filesystem-volumes\statefulset\csi-app-stateful-local.yaml 中的示例。

```
apiVersion: v1
kind: Service
metadata:
  name: csi-app-stateful-local-stor1lun10
  labels:
    app: csi-app-stateful-local-stor1lun10
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
```

```
    app: csi-app-stateful-local-stor1lun10
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: csi-app-stateful-local-stor1lun10
spec:
  selector:
    matchLabels:
      app: csi-app-stateful-local-stor1lun10
  serviceName: "csi-app-stateful-local-stor1lun10"
  replicas: 2
  template:
    metadata:
      labels:
        app: csi-app-stateful-local-stor1lun10
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: csi-app-stateful-local-stor1lun10
        image: busybox
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: lun10
          mountPath: /test10
        command: [ "sleep", "1000000" ]
  volumeClaimTemplates:
  - metadata:
      name: lun10
    spec:
      accessModes: [ "ReadWriteOnce" ] # 访问模式，filesystem 模式的卷支持 ReadWriteOnce
      storageClassName: "csi-storageclass-local-stateful-stor1lun10"
      resources:
        requests:
          storage: 100Gi
```

卷模式为 Block，创建 StatefulSet csi-app-stateful-local-block-lun11 的 YAML 配置文件 csi-app-stateful-local-block-lun11.yaml。参考 examples\block-volumes\statefulset\csi-app-stateful-local-block.yaml 中的示例。

```
apiVersion: v1
kind: Service
metadata:
  name: csi-app-stateful-local-block-lun11
  labels:
    app: csi-app-stateful-local-block-lun11
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: csi-app-stateful-local-block-lun11
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: csi-app-stateful-local-block-lun11
spec:
  selector:
    matchLabels:
      app: csi-app-stateful-local-block-lun11
  serviceName: "csi-app-stateful-local-block-lun11"
  replicas: 2
  template:
    metadata:
      labels:
        app: csi-app-stateful-local-block-lun11
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: csi-app-stateful-local-block-lun11
        image: busybox
```

```
        ports:
        - containerPort: 80
          name: web
        volumeDevices:
        - name: lun11
          devicePath: /dev/test11
        command: [ "sleep", "1000000" ]
  volumeClaimTemplates:
  - metadata:
      name: lun11
    spec:
      accessModes: [ "ReadWriteOnce" ]
#访问模式，Block 模式的卷支持的访问模式：ReadWriteOnce、ReadOnlyMany、ReadWriteMany
      volumeMode: Block #卷模式为 Block
      storageClassName: "csi-storageclass-local-stateful-block-stor1lun11"
      resources:
        requests:
          storage: 101Gi
```

（2）应用配置文件（以 csi-app-stateful-local-stor1lun10.yaml 为例）。

```
[root@server statefulset]# kubectl apply -f csi-app-stateful-local-stor1lun10.yaml
service/csi-app-stateful-local-stor1lun10 created
statefulset.apps/csi-app-stateful-local-stor1lun10 created
```

（3）验证创建的 StatefulSet（以 csi-app-stateful-local-stor1lun10 为例）。

```
[root@server statefulset]# kubectl get statefulset |grep csi-app-stateful-local-stor1lun10
csi-app-stateful-local-stor1lun10   2/2     28m
```

可以看到容器中已经挂载了路径/test10。

```
[root@server ~]# kubectl exec -it csi-app-stateful-local-stor1lun10-0  -- /bin/sh
/ # ls
bin     dev     etc     home    lib     lib64   proc    root    sys     test10  tmp     usr     var
```

3.5.4 调整 PV 的服务端连接位置

通过调整 PV 的服务端连接位置信息，优化 Pod 中计算资源和存储资源的连接信息，一方面可以提升数据的读写性能，另一方面也可以提升连接的可靠性和容错能力。

前提条件：HBlock 的存储池级别为 room，rack，server，才能调整 PV 的服务端连接位置信息。

调整的步骤如下：

1. 停止 PV 关联的所有 Pod。

```
kubectl delete -f pod1.yaml -f pod2.yaml -f podN.yaml
```

2. 调整 PV 的服务端连接位置。

- 如果是第一次调整 PV 的服务端连接位置，使用下列命令：

```
kubectl annotate pv pv-name faultDomains=faultDomain1，faultDomain2，faultDomainN
```

- 如果先前已经调整过 PV 的服务端连接位置，使用下列命令：

```
kubectl annotate pv pv-name faultDomains=faultDomain1，faultDomain2，faultDomainN --overwrite
```

pv-name：需要调整服务端连接位置的 PV 名称。

faultDomains：卷的服务端连接位置信息。根据存储池的故障域，修改 Target 所在服务器的列表（仅集群版支持），LUN 关联的 Target 优先从该服务器列表中选择所在服务器。例如存储池为 rack 级别，其拓扑图涵盖 rack1、rack2、rack3、rack4 中的节点，且 faultDomains 指定 rack1、rack2，那么修改 LUN 的 Target 时，LUN 关联的 Target 优先从 rack1、rack2 所包含的此存储池的服务器列表里进行选择，每个 rack 限选一个服务器。

注意：

- 对于同一 PV，其 faultDomains 的取值必须限定在 HBlock 集群的同一存储池内。
- faultDomains 的取值应低于或等于 HBlock 集群存储池的故障域等级。例如，若存储池 default 的故障域等级为 rack，则 faultDomains 只能配置为 rack 或 server 等级的节点，不得配置高于 rack 等级的节点，如 room1。
- 不得配置重复节点。例如：rack1,rack1, 或者 server1,rack1:server1。

3. 启动 PV 关联的所有 Pod。

```
Kubectl apply -f pod1.yaml -f pod2.yaml -f podN.yaml
```

4 HELM 方式使用指南

4.1 安装

HBlock CSI 插件支持以下两种安装方式，用户可以根据情况选择其中一种进行安装：

- 逐台安装：适用于 Kubernetes 集群的节点数量不多的部署场景。
- Docker 私仓方式安装：适用于节点多的部署场景。

4.1.1 前置条件

已经安装 HELM 3.12 及以上版本。

4.1.2 逐台安装

执行以下安装步骤（适用于 Kubernetes 集群的节点数量不多的部署场景）：

1. 在 Kubernetes master 和 node 上解压安装包。

```
unzip stor-csi-driver-1.3.0.zip
```

2. 在 Kubernetes master 和 node 上导入插件镜像。

```
cd stor-csi-driver-1.3.0
docker load < stor-csi-driver.tar
```

3. 在 Kubernetes master 节点执行部署脚本，进行插件的安装。

```
[root@server stor-csi-driver-1.3.0_x64]# cd charts/csi-driver-stor/
[root@server csi-driver-stor]# helm install stor ./
NAME: stor
LAST DEPLOYED: Mon Apr 29 17:41:08 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

```
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "cluster1",
    "apiEndPointList": [
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx"
    ],
    "storProvider": "xx"
  },
  {
    "clusterID": "cluster2",
    "apiEndPointList": [
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx"
    ],
    "storProvider": "xx"
  }
]
```

安装完成后，可以看到以下 pod、HBlock 的 plugin 以及 Sidecar 容器正常启动：

```
[root@server csi-driver-stor]# kubectl get pod | grep csi
csi-storplugin-controller-6c789569b9-fn77v   3/3     Running   0          29s
csi-storplugin-node-45tvs                    2/2     Running   0          29s
csi-storplugin-node-spmc4                    2/2     Running   0          29s
```

4.1.3 docker 私仓方式安装

若节点数量较多，可以使用私仓方式安装 HBlock CSI 插件，避免在 Kubernetes 的所有节点上都导入插件。用户可先将插件镜像推送到私仓中，修改插件 YAML 文件的 image 地址为私仓中镜像地址，执行安装脚本即可完成安装。

在集群中任意一台服务器上执行下面的操作：

1. 解压安装包。

```
unzip stor-csi-driver-1.3.0.zip
```

2. 导入插件镜像。

```
cd stor-csi-driver-1.3.0
docker load < stor-csi-driver.tar
```

3. 推送镜像到私仓。

```
docker tag stor-csi-driver:1.3.0 xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
docker push xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
```

其中，xxx.xxx.xxx.xxx:5000 为私仓地址。

4. 查看私仓。

```
cat /etc/docker/daemon.json
```

5. 修改 YAML 镜像拉取地址。

修改 charts\csi-driver-stor\values.yaml 文件中 csiStorPlugin 的值为 xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0。

```
images:
  # k8s 官方维护镜像
  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-
registrar:v2.8.0
  # CSI 插件镜像
  csiStorPlugin: xxx.xxx.xxx.xxx:5000/stor-csi-driver:1.3.0
```

6. 在 Kubernetes master 节点执行部署脚本，进行插件的安装。

```
[root@server stor-csi-driver-1.3.0_x64]# cd charts/csi-driver-stor/
[root@server csi-driver-stor]# helm install stor ./
```

```
NAME: stor
LAST DEPLOYED: Mon Apr 29 17:41:08 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "cluster1",
    "apiEndPointList": [
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx"
    ],
    "storProvider": "xx"
  },
  {
    "clusterID": "cluster2",
    "apiEndPointList": [
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx",
      "https://xx.xx.xx.xx:xx"
    ],
    "storProvider": "xx"
  }
]
```

安装完成后，可以看到以下 pod、HBlock 的 plugin 以及 Sidecar 容器正常启动：

```
[root@server csi-driver-stor]# kubectl get pod | grep csi
csi-storplugin-controller-6c789569b9-fn77v   3/3     Running   0          29s
csi-storplugin-node-45tvs                    2/2     Running   0          29s
csi-storplugin-node-spmc4                    2/2     Running   0          29s
```

4.2 卸载

如果要卸载 CSI：卸载插件前，先卸载样例。

1. 卸载前准备：

   编辑 values.yaml 文件，将样例都设置为禁用，并使用命令 helm upgrade stor ./进行更新。执行更新后，如果几分钟后，看到存在没有结束的样例 sc、statefulset、pod、pvc、pv 资源，需要手动进行清除。清除 sc、statefulset、pod、pvc、pv，可以执行下列命令清除对应资源。对于 pod、pvc、pv，必须按照顺序 pod > pvc > pv 执行删除命令。

   - 删除 sc：

```
kubectl delete sc storageclassname
```

   - 删除 statefulset：

```
kubectl delete statefulset statefulsetname  --cascade=true
```

   - 删除 pod：

```
kubectl delete pod podname
```

   - 删除 pvc：

```
kubectl delete pvc pvcname
```

   - 删除 pv：

```
kubectl delete pv pvname
```

2. 执行卸载命令卸载 CSI。

```
helm uninstall stor
```

4.3 升级

下载最新安装包，进入安装包的 charts/csi-driver-stor 目录，使各种配置和升级前保持一致，在 charts/csi-driver-stor 下执行升级命令即可。

```
helm upgrade stor ./
```

4.4 配置插件

4.4.1 设置 HBlock 相关配置

HBlock CSI 插件通过调用 HBlock 的 HTTP RESTful API 进行存储卷的管理操作，例如创建卷、删除卷、扩容卷等。需要配置插件访问 HBlock RESTful API 的 URL 地址和端口。

说明：支持对接多个 HBlock，包括：HBlock 单机版本、HBlock 集群版。

可以按照下列步骤设置 HBlock 访问地址。

(一) 修改配置文件，并应用配置文件。

1. 修改 charts/csi-driver-stor/values.yaml 配置文件，配置 HBlock 访问地址、访问用户名和密码、加密模式。

```
images:


  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0


  csiStorPlugin: stor-csi-driver:1.3.2


storConfig:


  configJson: |-
    [
      {
        "clusterID": "cluster1",
        "apiEndPointList": [
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx"
        ],
        "storProvider": "xx",
        "csiApiTimeout": 480
      },
      {
```

```
        "clusterID": "cluster2",
        "apiEndPointList": [
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx",
          "https://xx.xx.xx.xx:xx"
        ],
        "storProvider": "xx",
        "csiApiTimeout": 480
      }
    ]

  userKey:
IFsKIHsKICJjbHVzdGVySUQiOiAiY2x1c3RlcjEiLCAKICJ1c2VybmFtZSI6ICJhZG1pbiIsCiAicGFzc3dvcmQiOiAic2tza2RuZEQwIgogfSwKIHsKI
CJjbHVzdGVySUQiOiAiY2x1c3RlcjIiLCAKICJ1c2VybmFtZSI6ICJ4eCIsCiAicGFzc3dvcmQiOiAieHgiCiB9Cl0=

chap:
  # Whether to enable chap encryption. true（dHJ1ZQ==）, false（ZmFsc2U=）
  decryptFlag: ZmFsc2U=
  # Encrypt the key with base64
  decryptData: c3RvcjAxMjM0NTY3ODkwMQ==

# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
```

```
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
      # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
      local:
        enable: false
        clusterId: cluster1
        clusterMode: true
      # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
      localChap:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: user
        chapPassword: skskdndD0dkfL
# Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
      localChapDecrypt:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: SFFFzADcpZaUJKsYbPq54A==
        chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
    # Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
    statefulset:
        enable: false
        clusterId: cluster1
        clusterMode: true
    # Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun04
```

参数描述

参数	描述	是否必填
storConfig	HBlock 配置信息。	是
clusterID	指定 HBlock 的标识，在 csi-configMap 中唯一。 取值：字符串形式，长度范围是 1~256，可以包含字母、数字、和短横线（-），字母区分大小写。	是
apiEndPointList	HBlock 的服务器 IP 地址及 API 端口号；或者已经关联了 HBlock IP 和 API 端口号的 Kubernetes service 域名。 填写 HBlock 的服务器 IP 地址及 API 端口号时，如果是集群版，填写集群中所有的 IP 地址及 API 端口号；如果单机版，只填写一个即可。	是
storProvider	HBlock 产品名称。 取值：HBlock。	是
csiApiTimeout	指定 HBlock 创建 LUN 的等待时间，在等待时间内 LUN 创建失败，会报错，然后重试。 取值：正整数，默认值为 480，单位是秒。 **注意**：建议使用默认值。	否
userKey	对接 HBlock 的标识、用户名及密码的字符串的 base64 编码。 userKey 的编码过程，可以参见**配置 HBlock 访问用户名和密码**。	是
chap	加密模式配置。	否
decryptFlag	是否启用加密模式，配置为 true（启用）或 false（不启用）的 Base64 编码字符串。 取值： ● dHJ1ZQ==：启用加密，true 的 Base64 编码。 ● ZmFsc2U=：不启用加密，false 的 Base64 编码。	是
decryptData	加密的密钥。 **说明**：启用加密，此参数必填。	条件

	取值：源码为 16 位的字符串。需要对源码进行 Base64 编码。 decryptData 的编码过程可以参见**配置加密模式**。	
example	此字段是各样例的开启，配置完成 storConfig、userKey、chap 后，用户可以通过开启一个样例开关，来验证 HBlock CSI 是否已经可以正常启用。 **注意**：启动样例时，建议只开启一个样例，避免多样例互相冲突。验证完成后，需要将样例开关变为 false，避免与正式的实例冲突。	否

2. 应用配置文件，在 charts/csi-driver-stor 下执行更新配置命令。

```
helm upgrade stor ./
```

(二) 使用样例验证 HBlock 配置是否成功（可选）。

说明：如果 HBlock CSI 中配置了多个 HBlock，建议使用样例验证多次。例如 HBlock CSI 对应 2 个 HBlock，则需验证 2 个对应 HBlock 的样例。

1. 启用样例

（1） 修改配置文件 charts/csi-driver-stor/values.yaml 中 example 的开关，打开一个样例。例如下列配置中打开 blockVolumes 中的 dynamicPv 开关。

```
images:


  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0


  csiStorPlugin: stor-csi-driver:1.3.2
……


# Samples of pod, pvc and storageclass
example:
  # Mode of block
```

```
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: true
      clusterId: cluster1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
      # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
      local:
        enable: false
        clusterId: cluster1
        clusterMode: true
      # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
      localChap:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: user
        chapPassword: skskdndD0dkfL
# Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
      localChapDecrypt:
        enable: false
        clusterId: cluster1
        clusterMode: true
```

```
    chapUser: SFFFzADcpZaUJKsYbPq54A==
    chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
# Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
statefulset:
    enable: false
    clusterId: cluster1
    clusterMode: true
# Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
staticPv:
  enable: false
  clusterId: cluster1
  clusterMode: true
  lunName: lun04
```

Example 的参数描述

参数	描述
enable	是否开启示例： ● true：开启示例。 ● false：开始示例。 **注意**：启动样例时，建议只开启一个样例，避免多样例互相冲突。验证完成后，需要将样例开关变为 false，避免与正式的实例冲突。
clusterId	指定 HBlock 的标识。
clusterMode	指定 HBlock 是集群版还是单机版： ● true：HBlock 集群版。 ● false：HBlock 单机版。
chapUser	CHAP 认证的用户名。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的用户名使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 3~64，只能由字母、数字、句点(.)、短横线(-)、下划线(_)、冒号(:)组成，字母区分大小写，且仅支持以字母或数字开头。

chapPassword	CHAP 认证的密码。如果配置文件 csi-secret-decrypt.yaml 中的 decryptFlag 为 true，需要对 CHAP 认证的密码使用 DecryptData 配置的密钥对进行 AES（ECP、paddingcs7）加密，加密后的结果进行 Base64 编码，具体详见**配置加密模式**。 加密前取值：字符串形式，长度范围是 12~16，必须包含大写字母、小写字母、数字、下划线(_)中的至少两种字符，字母区分大小写。

（2） 应用配置文件，在 charts/csi-driver-stor 下执行更新配置命令。

```
helm upgrade stor ./
```

（3） 验证 HBlock CSI 是否已经可以正常启用：如果启用的样例成功建立 pod，则表示 HBlock CSI 可以正常启用。（可选）

```
kubectl get pod
```

2. 停用样例：如果启用了样例，为了避免与正式的实例冲突，建议停用样例。

（1） 在 values.yaml 中将样例开关变为 false。

（2） 执行命令 helm upgrade stor ./更新配置。

（3） 执行命令 kubectl get pod 检查样例是否停用。

说明：如果按步骤停用样例后，仍有因样例产生的 sc、statefulset、pod、pvc、pv，建议手动卸载这些资源。

4.4.2 配置示例

应用场景

CSI（以 1.3.0 版本为例）对接两个 HBlock，集群版和单机版。

- clusterID 为 stor1，对应集群版。clusterID 为 stor2，对应单机版。
- 集群版的服务器 IP 和 API 为：192.168.0.192:1443、192.168.0.110:1443、192.168.0.102:1443。单机版的服务器 IP 和 API 端口为 192.168.0.32:1443。
- 集群版的用户名和密码为：storuser、hblock12@。单机版的用户名和密码为：storuser、hblock12@。userKey 源码可以参考**配置 HBlock 访问用户名和密码**示例中的步骤 1、2.
- 集群版和单机版均不使用 CHAP 认证，如若使用，可以参考**配置加密模式**。
- 启用样例 blockVolumes 的动态 PV 样例（dynamicPv），验证 HBlock CSI 是否已经可以正常启用。

操作步骤

(一) 修改配置文件，并应用配置文件。

1. 修改 charts/csi-driver-stor/values.yaml 配置文件，配置 HBlock 访问地址、访问用户名和密码、加密模式。

```
images:


  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0


  csiStorPlugin: stor-csi-driver:1.3.0
# stor 集群信息
storConfig:
  # ClusterId And EndPointList
  configJson: |-
    [
      {
        "clusterID": "stor1",
        "apiEndPointList": [
          "https://192.168.0.192:1443",
          "https://192.168.0.110:1443",
```

```
          "https://192.168.0.102:1443"
        ],
        "storProvider": "HBlock"
      },
      {
        "clusterID": "stor2",
        "apiEndPointList": [
          "https://192.168.0.32:1443"
        ],
        "storProvider": "HBlock"
      }
    ]

  userKey: WwogICAgICB7CiAgICAgICAgImNsdXN0ZXJJRCI6ICJzdG9yMSIsICAKICAgICAgICAidXNlcm5hbWUiOiAic3RvcnVzZXIiLAogICAgICAgI
CJwYXNzd29yZCI6ICJoYmxvY2sxMkAiCiAgICAgIH0sCiAgICAgIHsKICAgICAgICAiY2x1c3RlcklEIjogInN0b3IyIiwgIAogICAgICAgICJ1c2VybmFtZ
SI6ICJzdG9ydXNlciIsCiAgICAgICAgInBhc3N3b3JkIjogImhibG9jazEyQCIKICAgICAgfQogICAgXQo=

chap:
  # Whether to enable chap encryption. true（dHJ1ZQ==）, false（ZmFsc2U=）
  decryptFlag: ZmFsc2U=
  # Encrypt the key with base64
  decryptData: c3RvcjAxMjM0NTY3ODkwMQ==

# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: true
      clusterId: stor1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
```

```
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
      # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
      local:
        enable: false
        clusterId: cluster1
        clusterMode: true
      # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
      localChap:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: user
        chapPassword: skskdndD0dkfL
# Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
      localChapDecrypt:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: SFFFzADcpZaUJKsYbPq54A==
        chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
    # Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
    statefulset:
        enable: false
        clusterId: cluster1
        clusterMode: true
    # Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun04
```

2. 应用配置文件，在 charts/csi-driver-stor 下执行更新配置命令。

```
[root@server csi-driver-stor]# helm upgrade stor ./
Release "stor" has been upgraded. Happy Helming!
NAME: stor
LAST DEPLOYED: Mon May 13 19:20:22 2024
```

```
NAMESPACE: default
STATUS: deployed
REVISION: 13
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "stor1",
    "apiEndPointList": [
      "https://192.168.0.192:1443",
      "https://192.168.0.110:1443",
      "https://192.168.0.102:1443"
    ],
    "storProvider": "HBlock"
  },
  {
    "clusterID": "stor2",
    "apiEndPointList": [
      "https://192.168.0.32:1443"
    ],
    "storProvider": "HBlock"
  }
]
```

(二) 使用样例验证 HBlock 配置是否成功（可选）。

说明：此处 HBlock CSI 中配置了 2 个 HBlock，则需要验证 2 个 HBlock 的样例。

- 验证 clusterID 为 stor1 的 HBlock（集群版）

1. 启用样例。

（1） 修改配置文件 charts/csi-driver-stor/values.yaml 中 example 的开关，打开样例 blockVolumes 中的动态 PV 样例，clusterID 为 stor1，clusterMode 为 true。

```
images:
  # k8s 官方维护镜像
  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0
  # CSI 插件镜像
  csiStorPlugin: stor-csi-driver:1.3.0


……


# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: true
      clusterId: stor1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
```

```
  # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
  local:
    enable: false
    clusterId: cluster1
    clusterMode: true
  # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
  localChap:
    enable: false
    clusterId: cluster1
    clusterMode: true
    chapUser: user
    chapPassword: skskdndD0dkfL
  # Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
  localChapDecrypt:
    enable: false
    clusterId: cluster1
    clusterMode: true
    chapUser: SFFFzADcpZaUJKsYbPq54A==
    chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
# Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
statefulset:
    enable: false
    clusterId: cluster1
    clusterMode: true
# Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
staticPv:
  enable: false
  clusterId: cluster1
  clusterMode: true
  lunName: lun04
```

（2） 应用配置文件，在 charts/csi-driver-stor 下执行更新配置命令。

```
[root@server csi-driver-stor]# helm upgrade stor ./
Release "stor" has been upgraded. Happy Helming!
NAME: stor
LAST DEPLOYED: Tue May 14 08:45:46 2024
NAMESPACE: default
```

```
STATUS: deployed
REVISION: 14
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "stor1",
    "apiEndPointList": [
      "https://192.168.0.192:1443",
      "https://192.168.0.110:1443",
      "https://192.168.0.102:1443"
    ],
    "storProvider": "HBlock"
  },
  {
    "clusterID": "stor2",
    "apiEndPointList": [
      "https://192.168.0.32:1443"
    ],
    "storProvider": "HBlock"
  }
]
```

（3） 验证 HBlock CSI 是否已经可以正常启用：如果启用的样例成功建立 pod，则表示 HBlock CSI 可以正常启用。

```
[root@pm-001 csi-driver-stor]# kubectl get pod
NAME                                        READY   STATUS             RESTARTS        AGE
csi-storplugin-controller-6c789569b9-mfh2q  3/3     Running            0               13d
csi-storplugin-node-fdqq7                   2/2     Running            6 (5d13h ago)   13d
csi-storplugin-node-tj7rf                   2/2     Running            0               13d
my-csi-app-block-dynamic                    1/1     Running            0                6m41s
```

2. 停用刚启动的样例：如果启用了样例，为了避免与正式的实例冲突，建议停用样例。

（1） 在 values.yaml 中，将 dynamicPv 的开关设置为 false。

```
images:
```

```
  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0

  csiStorPlugin: stor-csi-driver:1.3.0
……

# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: false
      clusterId: stor1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
      # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
      local:
        enable: false
        clusterId: cluster1
        clusterMode: true
 # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
      localChap:
        enable: false
        clusterId: cluster1
        clusterMode: true
```

```
        chapUser: user
        chapPassword: skskdndD0dkfL
# Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
      localChapDecrypt:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: SFFFzADcpZaUJKsYbPq54A==
        chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
    # Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
    statefulset:
        enable: false
        clusterId: cluster1
        clusterMode: true
    # Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun04
```

（2） 执行命令 helm upgrade stor ./更新配置。

```
[root@server csi-driver-stor]# helm upgrade stor ./
Release "stor" has been upgraded. Happy Helming!
NAME: stor
LAST DEPLOYED: Tue May 14 09:09:05 2024
NAMESPACE: default
STATUS: deployed
REVISION: 15
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "stor1",
    "apiEndPointList": [
      "https://192.168.0.192:1443",
      "https://192.168.0.110:1443",
      "https://192.168.0.102:1443"
```

```
    ],
    "storProvider": "HBlock"
  },
  {
    "clusterID": "stor2",
    "apiEndPointList": [
      "https://192.168.0.32:1443"
    ],
    "storProvider": "HBlock"
  }
]
```

（3） 执行命令 kubectl get pod 检查样例是否停用，若刚创建的 pod 不存在，即说明样例已停用。

```
[root@server csi-driver-stor]# kubectl get pod
NAME                                          READY   STATUS    RESTARTS        AGE
csi-storplugin-controller-6c789569b9-mfh2q    3/3     Running   0               14d
csi-storplugin-node-fdqq7                     2/2     Running   6 (5d14h ago)   14d
csi-storplugin-node-tj7rf                     2/2     Running   0               14d
```

- 验证 clusterID 为 stor2 的 HBlock（单机版）

1. 启用样例。

（1） 修改配置文件 charts/csi-driver-stor/values.yaml 中 example 的开关，将 clusterMode 的取值修改为 false，clusterId 为 stor2，打开样例 filesystem 中的动态 PV 开关。

```
……


# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: false
      clusterId: stor1
      clusterMode: true
```

```
  # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
  statefulset:
    enable: false
    clusterId: cluster1
    clusterMode: true
  # Sample of static pv. Path is templates/examples/block-volumes/static-pv
  staticPv:
    enable: false
    clusterId: cluster1
    clusterMode: true
    lunName: lun05
# Mode of filesystem
filesystemVolumes:
  dynamicPv:
    # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
    local:
      enable: true
      clusterId: stor2
      clusterMode: false
    # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
    localChap:
      enable: false
      clusterId: cluster1
      clusterMode: true
      chapUser: user
      chapPassword: skskdndD0dkfL
    # Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
    localChapDecrypt:
      enable: false
      clusterId: cluster1
      clusterMode: true
      chapUser: SFFFzADcpZaUJKsYbPq54A==
      chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
  # Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
  statefulset:
      enable: false
      clusterId: cluster1
```

```
    clusterMode: true
# Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
staticPv:
  enable: false
  clusterId: cluster1
  clusterMode: true
  lunName: lun04
```

（2） 应用配置文件，在 charts/csi-driver-stor 下执行更新配置命令。

```
[root@server csi-driver-stor]# helm upgrade stor ./
Release "stor" has been upgraded. Happy Helming!
NAME: stor
LAST DEPLOYED: Tue May 14 10:20:50 2024
NAMESPACE: default
STATUS: deployed
REVISION: 22
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "stor1",
    "apiEndPointList": [
      "https://192.168.0.192:1443",
      "https://192.168.0.110:1443",
      "https://192.168.0.102:1443"
    ],
    "storProvider": "HBlock"
  },
  {
    "clusterID": "stor2",
    "apiEndPointList": [
      "https://192.168.0.32:1443"
    ],
    "storProvider": "HBlock"
  }
]
```

（3） 验证 HBlock CSI 是否已经可以正常启用：如果启用的样例成功建立 pod，则表示 HBlock CSI 可以正常启用。

```
[root@pm-001 csi-driver-stor]# kubectl get pod
NAME                                        READY   STATUS             RESTARTS        AGE
csi-storplugin-controller-6c789569b9-mfh2q  3/3     Running            0               14d
csi-storplugin-node-fdqq7                   2/2     Running            6 (5d15h ago)   14d
csi-storplugin-node-tj7rf                   2/2     Running            0               14d
my-csi-app-filesystem-dynamic-local         1/1     Running            0               4m14s
```

2. 停用刚启动的样例：如果启用了样例，为了避免与正式的实例冲突，建议停用样例。

（1） 在 values.yaml 中，将 clusterMode 的取值修改为 true，关闭样例 filesystem 中的动态 PV 开关。

```
images:

  csiProvisioner: registry.aliyuncs.com/google_containers/csi-provisioner:v3.5.0
  csiAttacher: registry.aliyuncs.com/google_containers/csi-attacher:v4.3.0
  csiResizer: registry.aliyuncs.com/google_containers/csi-resizer:v1.8.0
  csiDriverRegistrar: registry.aliyuncs.com/google_containers/csi-node-driver-registrar:v2.8.0

  csiStorPlugin: stor-csi-driver:1.3.0
……

# Samples of pod, pvc and storageclass
example:
  # Mode of block
  blockVolumes:
    # Sample of dynamic pv. Path is templates/examples/block-volumes/dynamic-pv
    dynamicPv:
      enable: false
      clusterId: stor1
      clusterMode: true
    # Sample of statefulset with dynamic pv. Path is templates/examples/block-volumes/statefulset
    statefulset:
      enable: false
      clusterId: cluster1
      clusterMode: true
```

```
    # Sample of static pv. Path is templates/examples/block-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
      clusterMode: true
      lunName: lun05
  # Mode of filesystem
  filesystemVolumes:
    dynamicPv:
      # Sample of dynamic pv. Path is templates/examples/filesystem-volumes/dynamic-pv/local
      local:
        enable: false
        clusterId: stor2
        clusterMode: false
      # Sample of dynamic pv with chap disabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap
      localChap:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: user
        chapPassword: skskdndD0dkfL
# Sample of dynamic pv with chap enabled. Path is templates/examples/filesystem-volumes/dynamic-pv/local-chap-encrypt
      localChapDecrypt:
        enable: false
        clusterId: cluster1
        clusterMode: true
        chapUser: SFFFzADcpZaUJKsYbPq54A==
        chapPassword: tbHWrBep3R0RhkFaW+f5Fw==
    # Sample of statefulset with dynamic pv. Path is templates/examples/filesystem-volumes/statefulset
    statefulset:
        enable: false
        clusterId: cluster1
        clusterMode: true
    # Sample of static pv. Path is templates/examples/filesystem-volumes/static-pv
    staticPv:
      enable: false
      clusterId: cluster1
```

```
clusterMode: true

lunName: lun04
```

（2） 执行命令 helm upgrade stor ./更新配置。

```
[root@server csi-driver-stor]# helm upgrade stor ./
Release "stor" has been upgraded. Happy Helming!
NAME: stor
LAST DEPLOYED: Tue May 14 10:47:02 2024
NAMESPACE: default
STATUS: deployed
REVISION: 25
TEST SUITE: None
NOTES:
The Stor CSI Plugin has been published.

The plugin image is: stor-csi-driver:1.3.0

The Stor Cluster Info is: [
  {
    "clusterID": "stor1",
    "apiEndPointList": [
      "https://192.168.0.192:1443",
      "https://192.168.0.110:1443",
      "https://192.168.0.102:1443"
    ],
    "storProvider": "HBlock"
  },
  {
    "clusterID": "stor2",
    "apiEndPointList": [
      "https://192.168.0.32:1443"
    ],
    "storProvider": "HBlock"
  }
]
```

（3） 执行命令 kubectl get pod 检查样例是否停用，若刚创建的 pod 不存在，即说明样例已停用。

```
[root@server csi-driver-stor]# kubectl get pod
NAME                                        READY   STATUS    RESTARTS        AGE
```

```
csi-storplugin-controller-6c789569b9-mfh2q   3/3     Running   0               14d
csi-storplugin-node-fdqq7                    2/2     Running   6 (5d15h ago)   14d
csi-storplugin-node-tj7rf                    2/2     Running   0               14d
```

4.5 调用方式

使用 HELM 方式安装、配置 HBlock CSI 成功后，可以按照下列方法创建静态 PV、动态 PV、动态 PVC：

- 按照脚本调用方法创建静态 PV、动态 PV、动态 PVC（参见**调用方式**）。
- 参照 charts\csi-driver-stor\templates\examples 下的样例，创建属于自己的静态 PV、动态 PV、动态 PVC，使用 HELM 进行管理。各样例的参数可以参照脚本**调用方式**中的参数解释。

  注意：用户自己的 PV，PVC，POD 资源配置文件不可放在插件的 charts 中，需要用户自行管理。

  charts\csi-driver-stor\templates\examples 各举例路径如下：

```
[root@server stor-csi-driver-1.3.0_x64]# cd charts/
[root@server charts]# cd csi-driver-stor/
[root@server csi-driver-stor]# cd templates/
[root@server templates]# cd examples/
[root@server examples]# tree
.
├── block-volumes
│   ├── dynamic-pv
│   │   ├── csi-app-local-pvc-block.yaml
│   │   ├── csi-pvc-local-block.yaml
│   │   └── csi-storageclass-local.yaml
│   ├── statefulset
│   │   ├── csi-app-stateful-local-block.yaml
│   │   └── csi-storageclass-local-stateful-block.yaml
│   └── static-pv
│       ├── csi-app-local-pv-block.yaml
│       ├── csi-pvc-local-nocreate-block.yaml
│       └── csi-pv-local-block.yaml
└── filesystem-volumes
    ├── dynamic-pv
    │   ├── local
    │   │   ├── csi-app-local-pvc.yaml
    │   │   ├── csi-pvc-local.yaml
```

```
    |   |   └── csi-storageclass-local.yaml
    |   ├── local-chap
    |   |   ├── csi-app-local-pvc-chap.yaml
    |   |   ├── csi-pvc-local-chap.yaml
    |   |   └── csi-storageclass-local-chap.yaml
    |   └── local-chap-decrypt
    |       ├── csi-app-decrypt-local-pvc-chap.yaml
    |       ├── csi-pvc-local-chap-decrypt.yaml
    |       └── csi-storageclass-local-chap-decrypt.yaml
    ├── statefulset
    |   ├── csi-app-stateful-local.yaml
    |   └── csi-storageclass-local-stateful.yaml
    └── static-pv
        ├── csi-app-local-pv.yaml
        ├── csi-pvc-local-nocreate.yaml
        └── csi-pv-local.yaml

11 directories, 22 files
```

5 常见问题

Q：在创建 Pod 时，由于 iSCSI driver 问题出现报错 FailedMount，如何处理？

A：可能有以下两个原因：

- 报错信息为：MountVolume.Setup failed for volume “xxxx”: Kubernetes.io/csi: mounter.SetupAt failed: rpc error: code = Internal desc = iscsiadm error: iscsiadm: Could not login to [iface: default, target: xxxx, portal: 192.168.0.1,3260].iscsiadm: initiator repoeted error (12 - iSCSI driver not found. Please make sure it is loaded, and retry the operation)iscsiadm: Could not login to all portalsLogging in to [iface: default, target: xxxx, portal: 192.168.0.1,3260] (multiple) (exit status 12)

  出现此报错的原因是 Kubernetes 的 node 节点上已安装了 iscsi-initiator-utils。CSI 插件会自动部署 iscsi-initiator-utils，用于挂载 HBlock 的卷，node 节点上无需重复安装，如已安装，需要卸载 node 节点上的 iscsi-initiator-utils。建议卸载之后重启节点。

- 报错信息为：MountVolume.Setup failed for volume “xxxx”: rpc error: code = Internal desc = exit status 1

  出现此报错的可能原因 node 采用了 Multipath 的方式来连接到 HBlock 集群中的两个 Target，但是 node 节点之前安装过 MPIO，因此需要卸载节点上的 MPIO（如 device-mapper-multipath device-mapper-multipath-libs）。建议卸载之后重启节点。

Q：HBlock 集群模式下，在动态创建 PV 的场景中，创建出来的 LUN 对应多少个 Target IQN？

A：动态 PV 对应的是根据需求在 HBlock 中动态创建 LUN 的场景，在集群模式下，如果设置了 highAvailability: "ActiveStandby"，LUN 关联对应 Target 下的所有 IQN，如果 highAvailability 设置为 Disabled，LUN 关联对应 Target 下的一个 IQN。

Q：对于 HBlock 集群版，使用动态 PV 或者动态 PVC 创建卷时，创建卷成功，但写入数据失败，可能是什么原因导致？

A：可能原因：

- 可用故障域个数小于最小副本数。对于 HBlock 集群版，故障域个数大于等于卷的最小副本数，数据才能写入成功。
- 该卷在 HBlock 服务端被禁用。卷禁用后，无法读写卷数据。
- 网络问题，请检查客户端与 HBlock 端的网络连接情况。

如果排除以上原因还未解决，请联系天翼云工作人员。

6 附录

6.1 术语解释

6.1.1 Pod

Pod 是 Kubernetes 中创建和管理的最小单元，是一个或多个容器的组合，Pod 中的容器共享存储和网络资源，以及运行容器的规范。

6.1.2 Volume

Volume 是 Pod 内部的共享存储资源，生命周期和 Pod 相同，与容器无关，即使 Pod 上的容器停止或者重启，Volume 也不会受到影响。但如果 Pod 终止，那么 Volume 的生命周期也会结束。

6.1.3 Persistent Volume（PV）

Volume 中的数据无法持久保留，不能满足有状态服务的需求，因此需要 Persistent Volume。PV 是 Kubernetes 中的持久存储资源，是一种网络存储，它的生命周期和 Pod 无关。如果在 Kubernetes 中运行有状态服务，比如数据库 MySQL，MongoDB 或者中间件 Redis，RabbitMQ 等，那么就需要使用 PV，这样即使 Pod 终止也不会丢失数据。

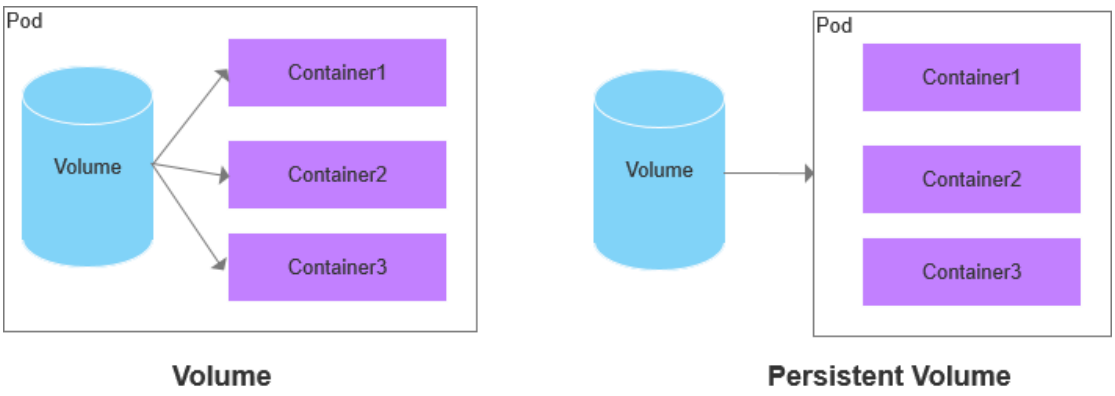

图10. PV

6.1.4 Persistent Volume Claim（PVC）

Persistent Volume Claim 是 PV 的声明。在 Kubernetes 中，直接使用 PV 作为存储时，需要集群管理员提前创建好 PV，使用上不灵活。而 PVC 可以将 Pod 和 PV 解耦，即 Pod 不直接使用 PV，而是通过 PVC 来使用 PV。这样，无需提前创建 PV，只要通过 StorageClass 把存储资源定义好，Kubernetes 就会根据使用需要，动态创建 PV，这种方式称为动态供应。

6.1.5 StorageClass

StorageClass 用于描述不同的存储类型。当通过 PVC 动态创建 HBlock 的卷时，需要在 StorageClass 中配置创建 HBlock 卷的参数，如卷冗余模式、扇区大小、写策略等信息。

6.1.6 Container Storage Interface（CSI）

Container Storage Interface（CSI）是通用存储接口，旨在实现容器编排器和存储提供商之间的互操作。通过 CSI，容器编排器能够使用任何存储提供商的存储服务，存储提供商也可以为任何容器编排器提供存储服务。

6.1.7 DaemonSet

DaemonSet 确保集群中所有（或部分）节点运行一个 Pod 副本，当节点加入到集群时，Pod 就被添加到节点上。当节点从集群中移除时，Pod 就被垃圾回收。HBlock CSI 插件使用的是 DaemonSet 类型的 Pod 控制器，会在每台节点上启动插件。

6.1.8 StatefulSet

StatefulSet 是有状态服务的 Pod 控制器。StatefulSet 用于管理一组 Pod 的部署和扩展，可以保证 Pod 的有序启动和停止等。