使用 **Trident**

Trident

NetApp
July 01, 2026

目录

使用 Trident

准备工作节点

Kubernetes 集群中的所有工作节点都必须能够装载您为 Pod 配置的卷。要准备工作节点，必须根据您的驱动程序选择安装 NFS、iSCSI、NVMe/TCP 或 FC 工具。

选择合适的工具

如果要组合使用驱动程序，应为驱动程序安装所有必需的工具。最新版本的 Red Hat Enterprise Linux CoreOS (RHCOS) 默认安装了这些工具。

NFS 工具

"安装 NFS 工具" 如果您使用的是： `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup` 或 `azure-netapp-files`。

iSCSI 工具

"安装 iSCSI 工具" 如果您使用的是： `ontap-san`， `ontap-san-economy`， `solidfire-san`。

NVMe 工具

"安装 NVMe 工具" 如果您使用 `ontap-san`进行基于 TCP 的非易失性内存快速 (NVMe) over TCP (NVMe/TCP) 协议。

NetApp 建议 NVMe/TCP 使用 ONTAP 9.12 或更高版本。

基于 **FC** 的 **SCSI** 工具

有关配置 FC 和 FC-NVMe SAN 主机的详细信息，请参见 "配置 FC FC-NVMe SAN 主机的方法"。

"安装 FC 工具" 如果您使用的是 `ontap-san` 与 sanType `fcp`（SCSI over FC）。

注意事项：* OpenShift 和 KubeVirt 环境支持 SCSI over FC。* Docker 不支持 SCSI over FC。* iSCSI 自我修复不适用于 SCSI over FC。

SMB 工具

"准备配置 SMB 卷" 如果您使用： `ontap-nas` 来配置 SMB 卷。

节点服务发现

Trident 尝试自动检测节点是否可以运行 iSCSI 或 NFS 服务。

节点服务发现可识别发现的服务，但不保证服务配置正确。相反，缺少发现的服务并不能保证卷挂载会失败。

审阅事件

Trident 创建事件以供节点标识发现的服务。要查看这些事件，请运行：

```
kubectl get event -A --field-selector involvedObject.name=<Kubernetes node name>
```

查看已发现的服务

Trident 标识在 Trident 节点 CR 上为每个节点启用的服务。要查看发现的服务，请运行：

```
tridentctl get node -o wide -n <Trident namespace>
```

NFS 卷

使用操作系统的命令安装 NFS 工具。确保 NFS 服务在启动时启动。

RHEL 8+

```
sudo yum install -y nfs-utils
```

Ubuntu

```
sudo apt-get install -y nfs-common
```

安装 NFS 工具后重新启动工作节点，以防止将卷附加到容器时出现故障。

iSCSI 卷

Trident 可以自动建立 iSCSI 会话、扫描 LUN、发现多路径设备、对其进行格式化并将其挂载到 pod。

iSCSI 自我修复功能

对于 ONTAP 系统，Trident 每五分钟运行一次 iSCSI 自我修复，以：

1. 识别 所需的 iSCSI 会话状态和当前 iSCSI 会话状态。
2. 将*所需状态与当前状态进行*比较，以确定所需的维修。Trident 决定维修优先级以及何时抢占维修。
3. 执行所需的维修，将当前 iSCSI 会话状态恢复到所需的 iSCSI 会话状态。

自我修复活动的日志位于相应 Daemonset Pod 的 `trident-main`容器中。要查看日志，必须在 Trident 安装期间将 `debug`设置为"true"。

Trident iSCSI 自我修复功能可以帮助防止：

- 网络连接问题后可能发生的陈旧或不正常的 iSCSI 会话。对于过时的会话，Trident 等待 7 分钟后退出，以重新建立与门户的连接。

例如，如果 CHAP 机密在存储控制器上循环，并且网络失去连接，则旧的（*stale*）CHAP 机密可能会持续存在。自我修复可以识别这一点，并自动重新建立会话以应用更新的 CHAP 机密。

- 缺少 iSCSI 会话
- 缺少 LUN

升级 **Trident** 前需要考虑的要点

- 如果仅使用每个节点 igroup（在 23.04+ 中引入），iSCSI 自我修复将为 SCSI 总线中的所有设备启动 SCSI 重新扫描。
- 如果仅使用后端范围的 igroup（自 23.04 起已弃用），iSCSI 自我修复将启动 SCSI 重新扫描 SCSI 总线中的确切 LUN ID。
- 如果使用每节点 igroup 和后端作用域 igroup 的组合，iSCSI 自我修复将启动 SCSI 重新扫描 SCSI 总线中的确切 LUN ID。

安装 **iSCSI** 工具

使用操作系统命令安装 iSCSI 工具。

开始之前

- Kubernetes 集群中的每个节点必须具有唯一的 IQN。这是必要的先决条件。
- 如果使用 RHCOS 4.5 或更高版本，或其他与 RHEL 兼容的 Linux 发行版，以及 `solidfire-san`驱动程序和 Element OS 12.5 或更低版本，请确保 CHAP 身份验证算法在 `/etc/iscsi/iscsid.conf`中设置为 MD5 。Element 12.7 提供了符合 FIPS 的安全 CHAP 算法 SHA1、SHA-256 和 SHA3-256。

```
sudo sed -i 's/^\(node.session.auth.chap_algs\).*/\1 = MD5/'
/etc/iscsi/iscsid.conf
```

- 当使用运行 RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 的工作节点并配合 iSCSI PV 时，请在 StorageClass 中指定 `discard` mountOption，以执行内联空间回收。请参阅 "Red Hat 文档"。
- 确保已升级到最新版本的 `multipath-tools`。

RHEL 8+

1. 安装以下系统软件包：

```
sudo yum install -y lsscsi iscsi-initiator-utils device-mapper-
multipath
```

2. 检查 iscsi-initiator-utils 版本是否为 6.2.0.874-2.el7 或更高版本：

```
rpm -q iscsi-initiator-utils
```

3. 将扫描设置为手动：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
```

4. 启用多路径：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> 请确保 `/etc/multipath.conf`包含 `find_multipaths no`下的 `defaults`。

5. 确保 `iscsid` 和 `multipathd` 正在运行：

```
sudo systemctl enable --now iscsid multipathd
```

6. 启用并启动 `iscsi`：

```
sudo systemctl enable --now iscsi
```

Ubuntu

1. 安装以下系统软件包：

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools
scsitools
```

2. 检查 open-iscsi 版本是否为 2.0.874-5ubuntu2.10 或更高版本（用于 bionic）或 2.0.874-7.1ubuntu6.1 或更高版本（用于 focal）：

```
dpkg -l open-iscsi
```

3. 将扫描设置为手动：

```
sudo sed -i 's/^\(node.session.scan\).*/\1 = manual/'
/etc/iscsi/iscsid.conf
```

4. 启用多路径：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> 请确保 `/etc/multipath.conf`包含 `find_multipaths no`下的 `defaults。

5. 确保 `open-iscsi`和 `multipath-tools`已启用并正在运行：

```
sudo systemctl status multipath-tools
sudo systemctl enable --now open-iscsi.service
sudo systemctl status open-iscsi
```

> 对于 Ubuntu 18.04，您必须在启动 `open-iscsi` 之前使用 `iscsiadm` 发现目标端口，才能启动 iSCSI 守护进程。或者，您也可以修改 `iscsi` 服务，使其自动启动 `iscsid`。

配置或禁用 **iSCSI** 自我修复

您可以配置以下 Trident iSCSI 自我修复设置，以修复过时的会话：

- **iSCSI** 自我修复间隔：确定调用 iSCSI 自我修复的频率（默认值：5 分钟）。您可以通过设置较小的数字将其配置为更频繁地运行，也可以通过设置较大的数字将其配置为较少频繁地运行。

> 将 iSCSI 自我修复间隔设置为 0 可完全停止 iSCSI 自我修复。我们不建议禁用 iSCSI 自我修复；仅当 iSCSI 自我修复无法按预期工作或出于调试目的时，才应在某些情况下禁用 iSCSI 自我修复。

- **iSCSI Self-Healing Wait Time**：确定 iSCSI 自我修复在从不正常会话注销并尝试重新登录之前等待的持续时间（默认值：7 分钟）。您可以将其配置为更大的数值，以便识别为不正常的会话必须等待更长时间才能注销，然后尝试重新登录，或者配置为更小的数值以便更早注销并登录。

Helm

要配置或更改 iSCSI 自我修复设置，请在 helm 安装或 helm 更新期间传递 `iscsiSelfHealingInterval`和 `iscsiSelfHealingWaitTime`参数。

以下示例将 iSCSI 自我修复间隔设置为 3 分钟，将自我修复等待时间设置为 6 分钟：

```
helm install trident trident-operator-100.2602.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

tridentctl

要配置或更改 iSCSI 自我修复设置，请在 tridentctl 安装或更新期间传递 `iscsi-self-healing-interval`和 `iscsi-self-healing-wait-time`参数。

以下示例将 iSCSI 自我修复间隔设置为 3 分钟，将自我修复等待时间设置为 6 分钟：

```
tridentctl install --iscsi-self-healing-interval=3m0s --iscsi-self
-healing-wait-time=6m0s -n trident
```

NVMe/TCP 卷

使用操作系统命令安装 NVMe 工具。

- NVMe 需要 RHEL 9 或更高版本。
- 如果 Kubernetes 节点的内核版本太旧，或者 NVMe 包不适用于您的内核版本，则可能需要将节点的内核版本更新为使用 NVMe 包的内核版本。

RHEL 9

```
sudo yum install nvme-cli
sudo yum install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

Ubuntu

```
sudo apt install nvme-cli
sudo apt -y install linux-modules-extra-$(uname -r)
sudo modprobe nvme-tcp
```

验证安装

安装后，使用以下命令验证 Kubernetes 集群中的每个节点都有唯一的 NQN：

```
cat /etc/nvme/hostnqn
```

Trident 会修改该 `ctrl_device_tmo` 值，以确保 NVMe 在出现故障时不会放弃该路径。请勿更改此设置。

基于 FC 卷的 SCSI

现在，您可以使用光纤通道 (FC) 协议与 Trident 来配置和管理 ONTAP 系统上的存储资源。

前提条件

配置 FC 所需的网络和节点设置。

网络设置

1. 获取目标接口的 WWPN。请参阅 "network interface show"以获取更多信息。
2. 获取启动器（主机）上接口的 WWPN。

   请参见相应的主机操作系统实用程序。

3. 使用主机和目标的 WWPN 在 FC 交换机上配置分区。

   有关信息，请参见相应的交换机供应商文档。

   有关详细信息，请参阅以下 ONTAP 文档：

   - "Fibre Channel 和 FCoE 分区概述"
   - "配置 FC FC-NVMe SAN 主机的方法"

安装 **FC** 工具

使用操作系统的命令安装 FC 工具。

- 当使用运行 RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 的工作节点并配合 FC PVs 时，请在 `discard` StorageClass 中指定 mountOption 以执行内联空间回收。请参阅 "Red Hat 文档"。

RHEL 8+

1. 安装以下系统软件包：

```
sudo yum install -y lsscsi device-mapper-multipath
```

2. 启用多路径：

```
sudo mpathconf --enable --with_multipathd y --find_multipaths n
```

> 请确保 `/etc/multipath.conf`包含 `find_multipaths no`下的 `defaults。

3. 请确保 `multipathd` 正在运行：

```
sudo systemctl enable --now multipathd
```

Ubuntu

1. 安装以下系统软件包：

```
sudo apt-get install -y lsscsi sg3-utils multipath-tools scsitools
```

2. 启用多路径：

```
sudo tee /etc/multipath.conf <<-EOF
defaults {
    user_friendly_names yes
    find_multipaths no
}
EOF
sudo systemctl enable --now multipath-tools.service
sudo service multipath-tools restart
```

> 请确保 `/etc/multipath.conf`包含 `find_multipaths no`下的 `defaults。

3. 确保 `multipath-tools` 已启用并正在运行：

```
sudo systemctl status multipath-tools
```

准备配置 SMB 卷

您可以使用 `ontap-nas` 驱动程序配置 SMB 卷。

必须在 SVM 上配置 NFS 和 SMB/CIFS 协议，才能为 ONTAP 本地群集创建 `ontap-nas-economy` SMB 卷。无法配置这些协议中的任何一个都将导致 SMB 卷创建失败。

`autoExportPolicy` 不支持 SMB 卷。

开始之前

在设置 SMB 卷之前，必须具有下列内容。

- 具有 Linux 控制器节点和至少一个运行 Windows Server 2022 的 Windows worker 节点的 Kubernetes 集群。Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。
- 至少有一个包含您的 Active Directory 凭据的 Trident 密码。要生成密钥 `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- 配置为 Windows 服务的 CSI 代理。要配置 `csi-proxy`，请参阅"GitHub：CSI Proxy"或"GitHub：适用于 Windows 的 CSI 代理"了解在 Windows 上运行的 Kubernetes 节点。

步骤

1. 对于本地 ONTAP，您可以选择创建 SMB 共享，或者 Trident 可以为您创建一个共享。

   Amazon FSx for ONTAP 需要 SMB 共享。

   您可以使用 "Microsoft 管理控制台" 共享文件夹管理单元或使用 ONTAP CLI 以两种方式之一创建 SMB 管理共享。要使用 ONTAP CLI 创建 SMB 共享：

   a. 如有必要，请为共享创建目录路径结构。

      此 `vserver cifs share create` 命令在共享创建期间检查 -path 选项中指定的路径。如果指定的路径不存在，则命令失败。

   b. 创建与指定 SVM 关联的 SMB 共享：

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

   c. 验证是否已创建此共享：

```
vserver cifs share show -share-name share_name
```

有关详细信息，请参见 "创建 SMB 共享"。

2. 创建后端时，必须配置以下内容以指定 SMB 卷。对于所有 FSx for ONTAP 后端配置选项，请参阅 "FSx for ONTAP 配置选项和示例"。

参数	说明	示例
`smbShare`	可以指定以下选项之一：使用 Microsoft Management Console 或 ONTAP CLI 创建的 SMB 共享的名称；允许 Trident 创建 SMB 共享的名称；或者可以将参数留空以阻止对卷的公共共享访问。此参数对于本地 ONTAP 是可选的。此参数是 Amazon FSx for ONTAP 后端所必需的，不能为空。	`smb-share`
`nasType`	*必须设置为 `smb`。*如果为 null，则默认为 `nfs`。	`smb`
`securityStyle`	新卷的安全样式。对于 **SMB** 卷，必须设置为 **`ntfs`** 或 **`mixed`**。	`ntfs` 或 `mixed` 用于 SMB 卷
`unixPermissions`	新卷的模式。对于 **SMB** 卷，必须留空。	""

配置和管理后端

配置后端

后端定义 Trident 与存储系统之间的关系。它告诉 Trident 如何与该存储系统进行通信，以及 Trident 应如何从中配置卷。

Trident 自动从后端提供与存储类定义的要求相匹配的存储池。了解如何为您的存储系统配置后端。

- "配置 Azure NetApp Files 后端"
- "配置 Google Cloud NetApp Volumes 后端"
- "配置 NetApp HCI 或 SolidFire 后端"
- "使用 ONTAP 或 Cloud Volumes ONTAP NAS 驱动程序配置后端"
- "使用 ONTAP 或 Cloud Volumes ONTAP SAN 驱动程序配置后端"
- "将 Trident 与 Amazon FSx for NetApp ONTAP 结合使用"

Azure NetApp Files

配置 **Azure NetApp Files** 后端

使用 Azure NetApp Files 作为 Trident 的后端。此后端支持 NFS 和 SMB 卷。Trident 支持 Azure Kubernetes Service (AKS) 集群的托管身份和工作负载身份。

支持的 **Azure** 云环境

Trident 支持多个 Azure 云环境中的 Azure NetApp Files 后端。

支持的 Azure 云包括：

- Azure 商业版
- Azure Government（Azure Government / MAG）

部署 Trident 或配置 Azure NetApp Files 后端时，请确保 Azure Resource Manager 和身份验证终结点与 Azure 云环境匹配。

查看 **Azure NetApp Files** 驱动程序支持

Trident 提供了以下 Azure NetApp Files 存储驱动程序。

支持的访问模式包括 *ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX) 和 *ReadWriteOncePod* (RWOP)。

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`azure-netapp-files`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	`nfs`, `smb`

查看考虑事项

- Azure NetApp Files 不支持小于 50 GiB 的卷。当请求较小的卷时，Trident 会创建 50-GiB 卷。
- Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。
- 非商业 Azure 云中的 Azure NetApp Files 部署需要特定于云的 Azure Resource Manager 和身份验证端点。确保 Trident 和任何后端配置使用适合您的 Azure 云环境的端点。

对 **AKS** 使用托管标识

Trident 支持 "托管标识" AKS 集群。

如果使用 `tridentctl` 来创建或管理 Azure NetApp Files 后端，请确保为正确的 Azure 云环境配置它。

要使用托管身份，必须具有：

- 使用 AKS 部署的 Kubernetes 集群
- 在 AKS Kubernetes 集群上配置的托管身份
- Trident 已安装， `cloudProvider` 设置为 `"Azure"`

Trident 操作员

编辑 `tridentorchestrator_cr.yaml` 并设置 `cloudProvider` 为 "Azure"。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
```

Helm

以下示例安装 Trident 并通过环境变量 `cloudProvider`进行设置 `$CP`：

```
helm install trident trident-operator-100.2602.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

以下示例安装 Trident 并将 `cloud-provider` 标志设置为 `Azure`：

```
tridentctl install --cloud-provider="Azure" -n trident
```

使用 **AKS** 的工作负载标识

工作负载标识使 Kubernetes Pod 能够通过作为工作负载标识进行身份验证来访问 Azure 资源。

如果使用 `tridentctl` 来创建或管理 Azure NetApp Files 后端，请确保为正确的 Azure 云环境配置它。

要使用工作负载标识，必须具有：

- 使用 AKS 部署的 Kubernetes 集群
- 在 AKS Kubernetes 集群上配置的工作负载标识和 oidc-issuer
- 已安装 Trident，其中 `cloudProvider`设置为 `"Azure"`，`cloudIdentity`设置为工作负载身份值

Trident 操作员

编辑 `tridentorchestrator_cr.yaml` 并设置 `cloudProvider` 为 `"Azure"`。设置 `cloudIdentity` 为 `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
  cloudIdentity: 'azure.workload.identity/client-id: xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxx' # Edit
```

Helm

使用以下环境变量设置 **cloud-provider (CP)** 和 **cloud-identity (CI)** 标志的值：

```
export CP="Azure"
export CI="'azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx'"
```

以下示例安装 Trident，并使用 `cloudProvider`进行设置，使用 `$CP，并使用 `cloudIdentity`进行设置，使用 `$CI：

```
helm install trident trident-operator-100.6.0.tgz --set
cloudProvider=$CP --set cloudIdentity="$CI"
```

<code>tridentctl</code>

使用以下环境变量设置 **cloud provider** 和 **cloud identity** 标志的值：

```
export CP="Azure"
export CI="azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx"
```

以下示例安装 Trident 并将 `cloud-provider`设置为 `$CP`并将 `cloud-identity`设置为 `$CI：

```
tridentctl install --cloud-provider=$CP --cloud-identity="$CI" -n
trident
```

准备配置 **Azure NetApp Files** 后端

在配置 Azure NetApp Files 后端之前，需要确保满足以下要求。

支持的 **Azure** 云环境

Trident 支持多个 Azure 云环境中的 Azure NetApp Files 后端。

支持的 Azure 云包括：

- Azure 商业版
- Azure Government（Azure Government / MAG）

在准备环境时，请确保在相应的 Azure 云环境中创建 Azure 订阅、标识配置和 Azure NetApp Files 资源。

NFS 和 **SMB** 卷的先决条件

如果是首次使用 Azure NetApp Files 或在新位置使用，则需要一些初始配置才能设置 Azure NetApp Files 并创建 NFS 卷。请参阅 "Azure：设置 Azure NetApp Files 并创建 NFS 卷"。

要配置和使用 "Azure NetApp Files"后端，您需要以下内容：

> - 在 AKS 集群上使用托管标识时，`subscriptionID`、 `tenantID`、 `clientID`、 `location` 和 `clientSecret` 是可选的。
> - `tenantID`、 `clientID` 和 `clientSecret` 在 AKS 集群上使用云标识时是可选的。
> - 非商业 Azure 云中的 Azure NetApp Files 部署需要特定于云的 Azure Resource Manager 和身份验证端点。确保 Trident 和任何后端配置使用适合您的 Azure 云环境的端点。

- 容量池。请参见 "Microsoft：为 Azure NetApp Files 创建容量池"。
- 委托给 Azure NetApp Files 的子网。请参见 "Microsoft：将子网委托给 Azure NetApp Files"。
- `subscriptionID` 从已启用 Azure NetApp Files 的 Azure 订阅。
- `tenantID`、 `clientID` 和 `clientSecret` 来自 Azure Active Directory 中对 Azure NetApp Files 服务具有足够权限的 "应用注册"。应用程序注册应使用以下任一功能：
  - 所有者或参与者角色"由 Azure 预定义"。
  - 订阅级别的 "自定义 Contributor 角色"(`assignableScopes`）具有以下权限，仅限于 Trident 所需的权限。创建自定义角色后，"使用 Azure 门户分配角色"。

自定义贡献者角色

```
{
  "id": "/subscriptions/<subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-
definition-id>",
  "properties": {
    "roleName": "custom-role-with-limited-perms",
    "description": "custom role providing limited permissions",
    "assignableScopes": [
      "/subscriptions/<subscription-id>"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.NetApp/netAppAccounts/capacityPools/read",
          "Microsoft.NetApp/netAppAccounts/capacityPools/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
read",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
write",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/
delete",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/MountTarge
ts/read",
          "Microsoft.Network/virtualNetworks/read",
          "Microsoft.Network/virtualNetworks/subnets/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
ions/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
```

```
ions/delete",
          "Microsoft.Features/features/read",
          "Microsoft.Features/operations/read",
          "Microsoft.Features/providers/features/read",

"Microsoft.Features/providers/features/register/action",

"Microsoft.Features/providers/features/unregister/action",

"Microsoft.Features/subscriptionFeatureRegistrations/read"
        ],
        "notActions": [],
        "dataActions": [],
        "notDataActions": []
      }
    ]
  }
}
```

- 包含至少一个 `location`的 Azure "委派子网"。从 Trident 22.01 开始， `location`参数是后端配置文件顶层的必填字段。在虚拟池中指定的位置值将被忽略。
- 要使用 `Cloud Identity`，请从 "用户分配的托管标识"获取 `client ID`，并在 `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`中指定该 ID。

SMB 卷的其他要求

要创建 SMB 卷，必须具有：

- Active Directory 已配置并连接到 Azure NetApp Files。请参见 "Microsoft：为 Azure NetApp Files 创建和管理 Active Directory 连接"。
- 具有 Linux 控制器节点和至少一个运行 Windows Server 2022 的 Windows worker 节点的 Kubernetes 集群。Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。
- 至少包含一个包含 Active Directory 凭据的 Trident 密码，以便 Azure NetApp Files 可以向 Active Directory 进行身份验证。要生成密钥 `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- 配置为 Windows 服务的 CSI 代理。要配置 `csi-proxy`，请参阅"GitHub：CSI Proxy"或"GitHub：适用于 Windows 的 CSI 代理"了解在 Windows 上运行的 Kubernetes 节点。

Azure NetApp Files 后端配置选项和示例

了解 Azure NetApp Files 的 NFS 和 SMB 后端配置选项并查看配置示例。

Trident 使用后端配置（子网、虚拟网络、服务级别和位置）在请求位置可用的容量池上创建 Azure NetApp Files 卷，并与请求的服务级别和子网匹配。

Azure NetApp Files 后端提供这些配置选项。

参数	说明	默认
`version`	后端配置版本。	始终为 1
`storageDriverName`	存储驱动程序的名称	"azure-netapp-files"
`backendName`	存储后端的自定义名称	驱动程序名称 + "_" + 随机字符
`subscriptionID`	来自您的 Azure 订阅的订阅 ID，在 AKS 集群上启用托管标识时为可选。	
`tenantID`	当在 AKS 集群上使用托管标识或云标识时，来自应用注册的租户 ID 可选。	
`clientID`	当在 AKS 集群上使用托管标识或云标识时，来自应用注册的客户端 ID 可选。	
`clientSecret`	当在 AKS 集群上使用托管标识或云标识时，来自应用注册的客户端密钥可选。	
`serviceLevel`	`Standard`、 `Premium`或 `Ultra`之一	""（随机）
`location`	将在其中创建新卷的 Azure 位置的名称在 AKS 群集上启用托管标识时可选。	
`resourceGroups`	用于筛选已发现资源的资源组列表	"[]"（无过滤器）
`netappAccounts`	用于筛选发现的资源的 NetApp 帐户列表	"[]"（无过滤器）
`capacityPools`	用于筛选发现资源的容量池列表	"[]"（无过滤器，随机）
`virtualNetwork`	具有委派子网的虚拟网络的名称	""
`subnet`	委托给 `Microsoft.Netapp/volumes`的子网的名称	""
`networkFeatures`	卷的 VNet 功能集，可以是 `Basic`或 `Standard`。并非所有地区都提供 Network Features，可能必须在订阅中启用。在未启用此功能时指定 `networkFeatures`会导致卷配置失败。	""

参数	说明	默认
`nfsMountOptions`	NFS 挂载选项的精细控制。SMB 卷将被忽略。要使用 NFS 版本 4.1 挂载卷，请在逗号分隔的挂载选项列表中包含 `nfsvers=4`以选择 NFS v4.1。存储类定义中设置的挂载选项会覆盖后端配置中设置的挂载选项。	"nfsvers=3"
`limitVolumeSize`	如果请求的卷大小高于此值，则配置失败	"（默认情况下不强制执行）
`debugTraceFlags`	故障排除时使用的调试标志。示例，`\{"api": false, "method": true, "discovery": true}`。除非正在进行故障排除并需要详细的日志转储，否则不要使用此选项。	空
`nasType`	配置 NFS 或 SMB 卷创建。选项为 `nfs`、 `smb`或 null。设置为 null 默认为 NFS 卷。	`nfs`
`supportedTopologies`	表示此后端支持的区域和可用区列表。有关详细信息，请参阅 "使用 CSI 拓扑"。	
`qosType`	表示 QoS 类型：自动或手动。	自动
`maxThroughput`	设置允许的最大吞吐量（以 MiB/秒为单位）。仅支持手动 QoS 容量池。	`4 MiB/sec`

有关网络功能的更多信息，请参阅 "为 Azure NetApp Files 卷配置网络功能"。

考虑 **Azure** 云环境（**26.02**）

从 26.02 版本开始，Trident 支持在多个 Azure 云环境中创建和管理 Azure NetApp Files 后端。

支持的 Azure 云包括：

- Azure 商业版
- Azure Government（Azure Government / MAG）

部署 Trident 或创建 Azure NetApp Files 后端时，请确保 Azure Resource Manager 和身份验证终结点与 Azure 云环境匹配。如果端点不匹配， `tridentctl`无法进行身份验证，并且后端创建失败。

所需权限和资源

如果在创建 PVC 时收到"未找到容量池"错误，则您的应用注册可能没有关联所需的权限和资源（子网、虚拟网络、容量池）。如果启用调试，Trident 会记录创建后端时发现的 Azure 资源。验证是否正在使用适当的角色。

`resourceGroups`、 `netappAccounts`、 `capacityPools`、 `virtualNetwork` 和 `subnet`
的值可以使用短名称或完全限定名称指定。在大多数情况下，建议使用完全限定名称，因为短名称可以与多个同名资源匹配。

如果 vNet 位于与 Azure NetApp Files (ANF) 存储帐户不同的资源组中，请在为后端配置 resourceGroups 列表时为虚拟网络指定资源组。

`resourceGroups`、 `netappAccounts` 和 `capacityPools`
值是将发现的资源集限制为此存储后端可用的资源的筛选器，可以以任何组合指定。完全限定的名称遵循以下格式：

类型	格式
资源组	<resource group>
NetApp 帐户	<resource group>/<netapp account>
容量池	<resource group>/<netapp account>/<capacity pool>
虚拟网络	<resource group>/<virtual network>
子网	<resource group>/<virtual network>/<subnet>

卷配置

您可以通过在配置文件的特殊部分中指定以下选项来控制默认卷配置。有关详细信息，请参见 [示例配置]。

参数	说明	默认
`exportRule`	新卷的导出规则。`exportRule` 必须是以 CIDR 表示法表示的 IPv4 地址或 IPv4 子网任意组合的逗号分隔列表。SMB 卷将被忽略。	"0.0.0.0/0"
`snapshotDir`	访问 `.snapshot` 目录	`true`, `false` (显式设置)。
`size`	新卷的默认大小	"100G"
`unixPermissions`	新卷的 unix 权限（4 位八进制数字）。SMB 卷将被忽略。	" （预览功能，需要在订阅中列入白名单）

示例配置

以下示例显示了将大多数参数保留为默认值的基本配置。这是定义后端的最简单方法。

最小配置

这是绝对最小后端配置。通过此配置，Trident 发现配置位置中委托给 Azure NetApp Files 的所有 NetApp 帐户、容量池和子网，并在其中一个池和子网上随机放置新卷。由于 `nasType`被省略， `nfs`默认值适用，后端将为 NFS 卷进行配置。

当您刚刚开始使用 Azure NetApp Files 并尝试各种功能时，此配置非常理想，但在实践中，您需要为所配置的卷提供额外的范围。

```
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
  tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
  clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
  clientSecret: SECRET
  location: eastus
```

AKS 的受管身份

此后端配置省略了 `subscriptionID`、 `tenantID`、 `clientID`和 `clientSecret`，这些在使用托管身份时是可选的。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - resource-group-1/netapp-account-1/ultra-pool
  resourceGroups:
    - resource-group-1
  netappAccounts:
    - resource-group-1/netapp-account-1
  virtualNetwork: resource-group-1/eastus-prod-vnet
  subnet: resource-group-1/eastus-prod-vnet/eastus-anf-subnet
```

适用于 **AKS** 的云标识

此后端配置省略 `tenantID`、 `clientID`和 `clientSecret`，这些在使用云标识时是可选的。

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-anf-1
  namespace: trident
spec:
  version: 1
  storageDriverName: azure-netapp-files
  capacityPools:
    - ultra-pool
  resourceGroups:
    - aks-ami-eastus-rg
  netappAccounts:
    - smb-na
  virtualNetwork: eastus-prod-vnet
  subnet: eastus-anf-subnet
  location: eastus
  subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
```

具有容量池筛选器的特定服务级别配置

此后端配置将卷放置在 Azure 的 `eastus`位置中的 `Ultra`容量池中。Trident 自动发现该位置中委托给 Azure NetApp Files 的所有子网，并随机在其中一个子网上放置新卷。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
```

具有手动 **QoS** 容量池的后端示例

此后端配置使用手动 QoS 容量池将卷放置在 Azure `eastus` 位置。

```
---
version: 1
storageDriverName: azure-netapp-files
backendName: anf1
location: eastus
labels:
  clusterName: test-cluster-1
  cloud: anf
  nasType: nfs
defaults:
  qosType: Manual
storage:
  - serviceLevel: Ultra
    labels:
      performance: gold
    defaults:
      maxThroughput: 10
  - serviceLevel: Premium
    labels:
      performance: silver
    defaults:
      maxThroughput: 5
  - serviceLevel: Standard
    labels:
      performance: bronze
    defaults:
      maxThroughput: 3
```

高级配置

此后端配置进一步将卷放置范围缩小到单个子网，并修改了一些卷配置默认值。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
virtualNetwork: application-group-1/eastus-prod-vnet
subnet: application-group-1/eastus-prod-vnet/my-subnet
networkFeatures: Standard
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 500Gi
defaults:
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  snapshotDir: "true"
  size: 200Gi
  unixPermissions: "0777"
```

虚拟池配置

此后端配置在单个文件中定义多个存储池。当您有多个支持不同服务级别的容量池，并且希望在 Kubernetes 中创建表示这些级别的存储类时，这非常有用。虚拟池标签用于根据 `performance`来区分池。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
resourceGroups:
  - application-group-1
networkFeatures: Basic
nfsMountOptions: vers=3,proto=tcp,timeo=600
labels:
  cloud: azure
storage:
  - labels:
      performance: gold
    serviceLevel: Ultra
    capacityPools:
      - application-group-1/netapp-account-1/ultra-1
      - application-group-1/netapp-account-1/ultra-2
    networkFeatures: Standard
  - labels:
      performance: silver
    serviceLevel: Premium
    capacityPools:
      - application-group-1/netapp-account-1/premium-1
  - labels:
      performance: bronze
    serviceLevel: Standard
    capacityPools:
      - application-group-1/netapp-account-1/standard-1
      - application-group-1/netapp-account-1/standard-2
```

支持的拓扑配置

Trident 便于根据区域和可用区为工作负载调配卷。此后端配置中的 `supportedTopologies`块用于为每个后端提供区域和区域的列表。此处指定的区域和区域值必须与每个 Kubernetes 集群节点上标签的区域和区域值匹配。这些区域和区域表示可以在存储类中提供的允许值列表。对于包含后端提供的区域和区域子集的存储类，Trident 会在上述区域和区域中创建卷。有关详细信息，请参阅 "使用 CSI 拓扑"。

```
---
version: 1
storageDriverName: azure-netapp-files
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: eastus
serviceLevel: Ultra
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
supportedTopologies:
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-1
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-2
```

存储类定义

以下 `StorageClass`定义参考了上述存储池。

使用 `parameter.selector` 字段的定义示例

使用 `parameter.selector`，您可以为每个 `StorageClass`指定用于托管卷的虚拟池。卷将具有所选池中定义的方面。

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: silver
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
allowVolumeExpansion: true

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: bronze
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze
allowVolumeExpansion: true
```

SMB 卷的定义示例

使用 `nasType`、 `node-stage-secret-name`和 `node-stage-secret-namespace`，可以指定 SMB 卷并提供所需的 Active Directory 凭据。

默认命名空间的基本配置

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

每个命名空间使用不同的机密

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

每个卷使用不同的密钥

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: anf-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "azure-netapp-files"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` 支持 SMB 卷的池的筛选器。
`nasType: nfs` 或 `nasType: null` NFS 池的筛选器。

创建后端

创建后端配置文件后，运行以下命令：

```
tridentctl create backend -f <backend-file>
```

如果使用非商业 Azure 云，请确保 `tridentctl`已配置为使用 Azure 资源管理器和 Azure 云环境的身份验证终结点。如果后端创建失败，请检查后端配置并查看日志以确定原因：

```
tridentctl logs
```

在识别并更正配置文件的问题后，您可以再次运行 create 命令。

Google Cloud NetApp Volumes

配置 **Google Cloud NetApp Volumes**

您可以将 Google Cloud NetApp Volumes 配置为 Trident 的后端，以便为 Kubernetes 工作负载配置存储。

概述

Trident 支持用于 NAS (NFS 和 SMB) 和块 (iSCSI) 工作负载的 Google Cloud NetApp Volumes。

- NAS 工作负载使用 `google-cloud-netapp-volumes` 后端
- 块 (iSCSI) 工作负载使用 `google-cloud-netapp-volumes-san` 后端

NAS 卷提供基于文件的存储，并使用 NFS 或 SMB 协议进行访问。这些卷支持跨多个 pod 或节点的共享访问。

块卷提供原始块存储，并作为连接到 Kubernetes 节点的 iSCSI 设备进行访问。当应用程序需要块级访问时，将使用这些卷。

这适用于以下环境：

- Trident 26.02 及更高版本
- Google Kubernetes Engine (GKE) 或 Red Hat OpenShift
- Google Cloud NetApp Volumes 存储池

要配置块 (iSCSI) 存储，请参见 "配置块存储 (iSCSI)"。

准备配置

云身份使 Kubernetes 工作负载能够通过作为工作负载身份进行身份验证而不是使用静态凭据来访问 Google

Cloud 资源。

要在 Google Cloud NetApp Volumes 中使用云标识，必须具有：

- 使用 Google Kubernetes Engine (GKE) 部署的 Kubernetes 集群
- 已在 GKE 群集上启用工作负载标识，已在节点池上启用元数据服务器
- 具有 Google Cloud NetApp Volumes Admin 角色(`roles/netapp.admin`）的 Google Cloud 服务帐户或同等自定义角色
- 安装 Trident 时，云提供商设置为 `GCP`并配置了云身份注释

Trident 操作员

要使用 Trident 操作员安装 Trident，请编辑 `tridentorchestrator_cr.yaml`：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  cloudProvider: "GCP"
  cloudIdentity: "iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com"
```

Helm

使用 Helm 安装 Trident 时设置云提供商和云标识：

```
helm install trident trident-operator-100.6.0.tgz \
  --set cloudProvider=GCP \
  --set cloudIdentity="iam.gke.io/gcp-service-account: cloudvolumes-
admin-sa@mygcpproject.iam.gserviceaccount.com"
```

tridentctl

通过指定云提供商和云身份来安装 Trident：

```
tridentctl install \
  --cloud-provider=GCP \
  --cloud-identity="iam.gke.io/gcp-service-account: cloudvolumes-admin-
sa@mygcpproject.iam.gserviceaccount.com" \
  -n trident
```

对于 Google Cloud NetApp Volumes UNIFIED 存储池，Trident 在卷操作期间应用 UNIFIED 特定的命名和验证规则。

定位卷时，Trident 可以评估多个兼容的卷名变体（例如，连字符和下划线格式），以提高导入和发现的可靠性。

驱动程序详细信息

Trident 提供 `google-cloud-netapp-volumes` 驱动程序，用于从 Google Cloud NetApp Volumes 配置 NAS 存储。

驱动程序支持以下访问模式：

- ReadWriteOnce（RWO）
- ReadOnlyMany（ROX）
- ReadWriteMany（RWX）
- ReadWriteOncePod（RWOP）

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`google-cloud-netapp-volumes`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	`nfs`, `smb`

配置 **Trident NAS** 后端

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
    network: "<vpc-network>"
```

配置 NAS 卷

NAS 卷使用 `google-cloud-netapp-volumes`后端进行调配，并支持 NFS 和 SMB 协议。

StorageClass 适用于 NFS 卷

要配置 NFS 卷，请将 `nasType`设置为 `nfs`。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "nfs"
allowVolumeExpansion: true
```

StorageClass 适用于 SMB 卷

要配置 SMB 卷，请将 `nasType`设置为 `smb`并提供凭据。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
allowVolumeExpansion: true
```

PersistentVolumeClaim 示例 (RWX)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwx
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

PersistentVolumeClaim 示例 (RWO)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-nas-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs
```

NAS 卷使用 `volumeMode: Filesystem`。

为 **SAN** 工作负载配置 **Google Cloud NetApp Volumes**

您可以将 Trident 配置为使用 iSCSI 协议从 Google Cloud NetApp Volumes 配置块存储卷。SAN 卷使用 `google-cloud-netapp-volumes-san`存储驱动程序从 Flex Unified 存储池进行配置。

此驱动程序专用于块工作负载，不支持 NAS 协议。

`google-cloud-netapp-volumes-san` 后端需要配置 iSCSI 块卷。 `google-cloud-netapp-volumes` 后端仅支持 NAS 协议，不能用于 SAN 工作负载。

概述

Trident 支持 Google Cloud NetApp Volumes SAN (iSCSI) 工作负载，使用 `google-cloud-netapp-volumes-san` 驱动程序。

SAN 卷从 Flex Unified 存储池进行调配，并作为 iSCSI 块设备呈现给 Kubernetes 节点。

这适用于以下环境：

- Trident 26.02 及更高版本
- Google Kubernetes Engine (GKE) 或 Red Hat OpenShift
- Google Cloud NetApp Volumes Flex 统一存储池
- 基于 iSCSI 的工作负载

Flex Unified 存储池

Flex Unified 存储池使用 iSCSI 协议提供块存储，是 SAN 配置所必需的：

- 支持 Flex Unified REGIONAL 池。
- 从 Trident 26.02.1 开始支持 Flex Unified ZONAL 池。
- SAN 工作负载仅支持 **Flex** 服务级别。

配置 Trident SAN 后端

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-san
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
      performance: flex
    network: "<vpc-network>"
    serviceLevel: Flex
```

创建 StorageClass

配置 SAN 后端后，创建一个引用 `google-cloud-netapp-volumes-san`驱动程序的 StorageClass。

文件系统类型在 StorageClass 中定义，而不是在后端。

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

支持的文件系统类型：

- `ext4` (默认)
- `ext3`
- `xfs`

SAN 驱动程序仅支持 Flex 服务级别，不使用特定于 NAS 的后端参数，例如 `exportRule`、`unixPermissions`、 `nasType`、 `snapshotDir`、 `nfsMountOptions`或与分层相关的设置。

配置块卷

ReadWriteOnce（RWO）

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadWriteOncePod（RWOP）

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rwop
spec:
  accessModes:
    - ReadWriteOncePod
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

ReadOnlyMany（ROX）

ROX 的常见模式是克隆现有 ReadWriteOnce 卷并将克隆挂载为只读。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-rox
spec:
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
  dataSource:
    kind: PersistentVolumeClaim
    name: gcnv-san-rwo
```

ReadWriteMany (RWX) — 仅原始块

只有当 `volumeMode: Block`时才支持 ReadWriteMany。

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gcnv-san-raw-rwx
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-san
```

块卷行为

块卷被调配为 iSCSI LUN，并作为块设备呈现给 Kubernetes 节点。

块卷：

- 使用 iSCSI 协议
- 支持文件系统和原始块呈现
- 由 Trident 连接和管理
- 支持多种 Kubernetes 访问模式

访问模式

Trident 配置的块卷支持以下访问模式：

- `ReadWriteOnce`（RWO）
- `ReadOnlyMany`（ROX）
- `ReadWriteOncePod`（RWOP）
- `ReadWriteMany` (RWX)，仅在 `volumeMode: Block`时受支持

volumeMode 行为

`volumeMode` 字段控制块卷的暴露方式：

- `Filesystem` Trident 格式化和装载卷。
- `Block` Trident 附加设备并将其公开为原始块设备。

支持的操作

使用 `google-cloud-netapp-volumes-san` 驱动程序配置的块卷支持：

- 创建
- 删除
- 克隆
- Snapshot
- 调整大小
- 导入

额外的 GiB 过度配置行为

Google Cloud NetApp Volumes 块卷包括内部元数据开销。与调配的容量相比，这种开销减少了内核可见的设备大小。

测试显示：

- 初始创建时约 300 KiB 的开销
- 调整大小后开销高达约 107 MiB

由于 Google Cloud NetApp Volumes 仅接受全 GiB 分配，Trident 确保可用设备大小始终满足或超过 PVC 请求：

- 将请求的大小舍入至下一个整 GiB
- 添加额外的 1 GiB 缓冲区

示例：

- PVC 请求：100 GiB
- Google Cloud NetApp Volumes 中的预配大小：101 GiB
- 应用程序可见的可用空间：至少 100 GiB

Pod 示例

文件系统挂载的块卷 **(RWO)**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-rwo
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeMounts:
    - name: data
      mountPath: /mnt/data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-rwo
```

原始块设备 **(RWX)**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-raw-rwx
spec:
  containers:
  - name: app
    image: ubuntu:22.04
    command: ["sleep", "infinity"]
    volumeDevices:
    - name: data
      devicePath: /dev/xda
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: gcnv-san-raw-rwx
```

连接和挂载行为

对于从 Google Cloud NetApp Volumes 配置的 SAN 卷：

- Trident 在 Flex Unified 存储池中创建逻辑单元号 (LUN)。
- 在发布期间， Trident 将 LUN 映射到每个节点的主机组。
- 在节点暂存期间，Trident：

- 登录到 iSCSI 目标
- 发现 LUN
- 配置多路径

- 如果 `volumeMode: Filesystem`，Trident 会根据需要格式化设备并挂载它。
- 如果 `volumeMode: Block`，Trident 会连接设备并将其直接暴露给 pod，无需格式化或挂载。

> SAN 块卷不提供分布式锁定或写入协调。当一个块卷被多个节点访问时（ReadWriteMany 与 `volumeMode: Block`），应用程序或文件系统必须管理并发。

准备配置 **Google Cloud NetApp Volumes** 后端

在配置 Google Cloud NetApp Volumes 后端之前，需要确保满足以下要求。

NFS 或 **SMB** 卷的先决条件

如果是首次使用 Google Cloud NetApp Volumes 或在新位置使用，则需要一些初始配置才能设置 Google Cloud NetApp Volumes 并创建 NFS 或 SMB 卷。请参见 "开始之前"。

在配置 Google Cloud NetApp Volumes 后端之前，请确保您具有以下内容：

- 使用 Google Cloud NetApp Volumes 服务配置的 Google Cloud 帐户。请参见 "Google Cloud NetApp Volumes"。
- 您的 Google Cloud 帐户的项目编号。请参见 "识别项目"。
- 具有 NetApp Volumes Admin (`roles/netapp.admin`) 角色的 Google Cloud 服务帐户。请参见 "身份和访问管理角色和权限"。
- 您的 GCNV 帐户的 API 密钥文件。请参见 "创建服务帐户密钥"
- 存储池。请参见 "存储池概述"。

有关如何设置 Google Cloud NetApp Volumes 访问权限的详细信息，请参阅 "设置对 Google Cloud NetApp Volumes 的访问权限"。

Google Cloud NetApp Volumes 后端配置选项和示例

了解 Google Cloud NetApp Volumes 的后端配置选项，并查看配置示例。

后端配置选项

每个后端在一个 Google Cloud 区域中配置卷。要在其他区域中创建卷，您可以定义其他后端。

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`storageDriverName` 的值必须指定为"google-cloud-netapp-volumes"。

参数	说明	默认
backendName	(可选) 存储后端的自定义名称	驱动程序名称 + "_" + API 密钥的一部分
storagePools	用于指定卷创建的存储池的可选参数。	
projectNumber	Google Cloud 帐户项目编号。该值位于 Google Cloud 门户主页上。	
location	Trident 创建 GCNV 卷的 Google Cloud 位置。在创建跨区域 Kubernetes 集群时，在 `location`中创建的卷可用于在多个 Google Cloud 区域的节点上计划的工作负载。跨区域流量会产生额外费用。	
apiKey	具有 netapp.admin`角色的 Google Cloud 服务帐户的 API 密钥。它包括 Google Cloud 服务帐户私钥文件的 JSON 格式内容（逐字复制到后端配置文件中）。 `apiKey`必须包括以下键的键值对：`type、 project_id、 client_email、 client_id、 auth_uri、 token_uri、 auth_provider_x509_cert_url`和 `client_x509_cert_url。	
nfsMountOptions	NFS 挂载选项的精细控制。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小高于此值，则设置失败。	" （默认情况下不强制执行）
serviceLevel	存储池及其卷的服务级别。值为 flex、 standard、 premium`或 `extreme。	
labels	要应用于卷的任意 JSON 格式标签集	""
network	用于 GCNV 卷的 Google Cloud 网络。	
debugTraceFlags	故障排除时使用的调试标志。示例， {"api":false, "method":true}。除非正在进行故障排除并需要详细的日志转储，否则不要使用此选项。	空
nasType	配置 NFS 或 SMB 卷创建。选项为 nfs、 `smb`或 null。设置为 null 默认为 NFS 卷。	nfs
supportedTopologies	表示此后端支持的区域和可用区列表。有关详细信息，请参阅 "使用 CSI 拓扑"。例如： supportedTopologies: - topology.kubernetes.io/region: asia-east1 topology.kubernetes.io/zone: asia-east1-a	

卷配置选项

您可以在配置文件的 defaults 部分中控制默认卷配置。

参数	说明	默认
`exportRule`	新卷的导出规则。必须是任何 IPv4 地址组合的逗号分隔列表。	"0.0.0.0/0"
`snapshotDir`	访问 `.snapshot` 目录	`true`, `false` (默认行为可能会有所不同。显式设置) NFSv3 的 "false"
`snapshotReserve`	为快照预留的卷百分比	" (接受默认值 0)
`unixPermissions`	新卷的 unix 权限（4 位八进制数字）。	""

示例配置

以下示例显示了将大多数参数保留为默认值的基本配置。这是定义后端的最简单方法。

最小配置

这是绝对最小后端配置。通过此配置，Trident 发现配置位置中委托给 Google Cloud NetApp Volumes 的所有存储池，并将新卷随机放置在其中一个池中。由于 `nasType`被省略， `nfs`默认值适用，后端将为 NFS 卷进行配置。

当您刚刚开始使用 Google Cloud NetApp Volumes 并尝试使用时，此配置非常理想，但在实践中，您可能需要为配置的卷提供额外的范围。

请将 `<id_value>` 和 `<key_value>` 替换为您的服务帐户凭据。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

SMB 卷的配置

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv1
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123456789"
  location: asia-east1
  serviceLevel: flex
  nasType: smb
  apiKey:
    type: service_account
    project_id: cloud-native-data
    client_email: trident-sample@cloud-native-
data.iam.gserviceaccount.com
    client_id: "123456789737813416734"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/trident-
sample%40cloud-native-data.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

使用 **StoragePools** 过滤器的配置

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  storagePools:
    - premium-pool1-europe-west6
    - premium-pool2-europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

虚拟池配置

此后端配置在单个文件中定义多个虚拟池。虚拟池在 `storage` 部分中定义。当您有多个支持不同服务级别的存储池，并且希望在 Kubernetes 中创建表示这些级别的存储类时，它们非常有用。虚拟池标签用于区分池。例如，在下面的示例中， `performance` 标签和 `serviceLevel` 类型用于区分虚拟池。

您还可以设置一些适用于所有虚拟池的默认值，并覆盖各个虚拟池的默认值。在以下示例中，`snapshotReserve` 和 `exportRule` 用作所有虚拟池的默认值。

有关详细信息，请参阅 "虚拟池"。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "123455380079"
  location: europe-west6
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
```

```
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

适用于 **GKE** 的云标识

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcp-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: '012345678901'
  network: gcnv-network
  location: us-west2
  serviceLevel: Premium
  storagePool: pool-premium1
```

支持的拓扑配置

Trident 便于根据区域和可用区为工作负载调配卷。此后端配置中的 `supportedTopologies`块用于为每个后端提供区域和区域的列表。此处指定的区域和区域值必须与每个 Kubernetes 集群节点上标签的区域和区域值匹配。这些区域和区域表示可以在存储类中提供的允许值列表。对于包含后端提供的区域和区域子集的存储类，Trident 会在上述区域和区域中创建卷。有关详细信息，请参阅 "使用 CSI 拓扑"。

```
---
version: 1
storageDriverName: google-cloud-netapp-volumes
subscriptionID: 9f87c765-4774-fake-ae98-a721add45451
tenantID: 68e4f836-edc1-fake-bff9-b2d865ee56cf
clientID: dd043f63-bf8e-fake-8076-8de91e5713aa
clientSecret: SECRET
location: asia-east1
serviceLevel: flex
supportedTopologies:
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-a
  - topology.kubernetes.io/region: asia-east1
    topology.kubernetes.io/zone: asia-east1-b
```

下一步是什么?

创建后端配置文件后，运行以下命令：

```
kubectl create -f <backend-file>
```

要验证是否已成功创建后端，请运行以下命令：

```
kubectl get tridentbackendconfig

NAME                BACKEND NAME        BACKEND UUID
PHASE    STATUS
backend-tbc-gcnv    backend-tbc-gcnv    b2fd1ff9-b234-477e-88fd-713913294f65
Bound    Success
```

如果后端创建失败，则后端配置有问题。您可以使用 `kubectl get tridentbackendconfig <backend-name>` 命令描述后端或通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在识别并更正配置文件的问题后，您可以删除后端并再次运行 create 命令。

存储类定义

以下是参考上述后端的基本 `StorageClass` 定义。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-nfs-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
```

使用 **`parameter.selector`**字段的定义示例：

使用 `parameter.selector`，您可以为每个 `StorageClass`指定用于托管卷的"虚拟池"。卷将具有所选池中定义的方面。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: extreme-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=extreme
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: premium-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium
  backendType: google-cloud-netapp-volumes

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=standard
  backendType: google-cloud-netapp-volumes
```

有关存储类的更多详细信息，请参阅 "创建存储类"。

SMB 卷的定义示例

使用 `nasType`、 `node-stage-secret-name` 和 `node-stage-secret-namespace`，可以指定 SMB 卷并提供所需的 Active Directory 凭据。任何具有任何权限/无权限的 Active Directory 用户/密码都可以用于节点阶段密码。

默认命名空间的基本配置

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: "default"
```

每个命名空间使用不同的机密

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

每个卷使用不同的密钥

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

`nasType: smb` 支持 SMB 卷的池的筛选器。 `nasType: nfs` 或 `nasType: null` NFS 池的筛选器。

PVC 定义示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: gcnv-nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: gcnv-nfs-sc
```

要验证 PVC 是否已绑定，请运行以下命令：

```
kubectl get pvc gcnv-nfs-pvc

NAME           STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS AGE
gcnv-nfs-pvc   Bound    pvc-b00f2414-e229-40e6-9b16-ee03eb79a213   100Gi
RWX            gcnv-nfs-sc  1m
```

为 **Google Cloud NetApp Volumes** 配置自动分层

自动分层通过 Trident 后端参数和 PersistentVolumeClaim 注释在卷配置期间进行配置。您可以使用 Trident 为 Google Cloud NetApp Volumes 配置自动分层。

概述

自动分层允许 Trident 配置卷，自动将非活动数据从性能层移动到容量层。这降低了存储成本，同时保留了频繁访问数据的性能。

Trident 仅在卷创建时应用自动分层设置。Trident 26.02 不支持预配后更改。

概念

自动分层

自动分层根据访问模式将不经常访问的数据从性能层移动到容量层。数据移动是异步发生的，并不是立即发生的。

分层策略

分层策略确定是否为卷启用自动分层。

支持以下策略：* `auto`：根据访问模式启用自动分层 * `none`：禁用自动分层

冷却天数

冷却天数指定数据块在符合分层条件之前必须保持非活动状态的最短天数。仅当分层策略设置为 `auto`时，才适用冷却天数。

配置模型

配置范围

可以在多个范围内配置自动分层：

- 存储池范围 适用于从池中配置的所有卷。
- 卷范围 通过 PersistentVolumeClaim 注释应用于单个卷。

Trident 根据每个设置的定义位置确定有效配置。

配置优先级

当在多个作用域中定义了相同的设置时，Trident 应用以下优先级顺序：

1. PersistentVolumeClaim 注释
2. Trident 后端配置
3. 存储池默认值

在较高优先级定义的设置将覆盖较低级别的值。

Trident 26.02 中支持的功能

Trident 26.02 支持 Google Cloud NetApp Volumes 的以下自动分层功能：

- 在卷配置期间启用或禁用自动分层
- 在 Trident 后端配置中定义分层策略
- 使用 PVC 注释覆盖分层策略和每个卷的冷却天数
- 为启用自动分层的卷配置冷却天数

Trident 26.02 中不支持的功能

不支持以下操作：

- 创建卷后修改自动分层设置
- 使用 Kubernetes 更新更改现有卷的分层策略
- 在 Trident 管理的配置工作流之外应用自动分层设置

后端配置参数

以下参数在 Trident 后端配置中定义时控制自动分层行为：

参数	必填项	说明
tieringPolicy	否	卷的分层策略（(`auto`或 `none`）
tieringMinimumCoolingDays	否	数据分层前的非活动天数（范围：2–183，默认值：31）

使用 **PersistentVolumeClaim** 注释的卷级覆盖

支持的注释

PersistentVolumeClaim 批注允许按卷覆盖自动分层设置。

标注	说明
`trident.netapp.io/tieringPolicy`	覆盖卷的分层策略
`trident.netapp.io/tieringMinimumCoolingDays`	覆盖该卷的冷却天数值

示例：**PersistentVolumeClaim** 使用自动分层覆盖

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-tiering-pvc
  annotations:
    trident.netapp.io/tieringPolicy: auto
    trident.netapp.io/tieringMinimumCoolingDays: "45"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: google-cloud-netapp-volumes-auto-tiering
  resources:
    requests:
      storage: 500Gi
```

行为和限制

配置行为

- 自动分层设置仅在创建卷时进行评估和应用。
- Trident 在配置后不会协调分层配置。
- 当分层策略设置为 `none`时，冷却天数将被忽略。

平台限制

- 只有 NAS 卷（NFS 和 SMB）才支持自动分层。
- 块卷 (iSCSI) 不支持自动分层。
- Google Cloud NetApp Volumes 存储池必须在 Google Cloud 中启用自动分层。

支持的值

- `tieringMinimumCoolingDays` 的有效范围：2 到 183
- 默认值：31

配置 NetApp HCI 或 SolidFire 后端

了解如何在 Trident 安装中创建和使用 Element 后端。

Element 驱动程序详细信息

Trident 提供 `solidfire-san`存储驱动程序来与集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

> `solidfire-san` 存储驱动程序支持 `_file_` 和 `_block_` 卷模式。对于 `Filesystem` volumeMode，Trident 创建一个卷并创建一个文件系统。文件系统类型由 `StorageClass` 指定。

驱动程序	协议	VolumeMode	支持的访问模式	支持的文件系统
`solidfire-san`	iSCSI	块	RWO、ROX、RWX、RWOP	无文件系统。原始块设备。
`solidfire-san`	iSCSI	Filesystem	RWO、RWOP	`xfs`, `ext3`, `ext4`

开始之前

在创建 Element 后端之前，您需要执行下列操作。

- 运行 Element 软件的受支持存储系统。
- 可管理卷的 NetApp HCI/SolidFire 群集管理员或租户用户的凭据。
- 所有 Kubernetes worker 节点都应安装相应的 iSCSI 工具。请参见 "worker 节点准备信息"。

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
`version`		始终为 1

参数	说明	默认
`storageDriverName`	存储驱动程序的名称	总是 "solidfire-san"
`backendName`	自定义名称或存储后端	"solidfire_" + 存储 (iSCSI) IP 地址
`Endpoint`	具有租户凭据的 SolidFire 集群的 MVIP	
`SVIP`	存储 (iSCSI) IP 地址和端口	
`labels`	要应用于卷的任意 JSON 格式标签集。	""
`TenantName`	要使用的租户名称（未找到时创建）	
`InitiatorIFace`	将 iSCSI 流量限制到特定主机接口	"default"
`UseCHAP`	使用 CHAP 对 iSCSI 进行身份验证。Trident 使用 CHAP。	true
`AccessGroups`	要使用的访问组 ID 列表	查找名为"trident"的访问组的 ID
`Types`	QoS 规范	
`limitVolumeSize`	如果请求的卷大小高于此值，则配置失败	"（默认情况下不强制执行）
`debugTraceFlags`	故障排除时使用的调试标志。例如，{"api":false, "method":true}	空

警告 除非正在进行故障排除并需要详细的日志转储，否则不要使用 `debugTraceFlags`。

示例 **1**：具有三种卷类型的 `solidfire-san` 驱动程序的后端配置

此示例显示了一个使用 CHAP 身份验证并使用特定 QoS 保证对三种卷类型进行建模的后端文件。然后，您很可能会使用 `IOPS` storage class 参数定义要使用其中每个的存储类。

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
labels:
  k8scluster: dev1
  backend: dev1-element-cluster
UseCHAP: true
Types:
  - Type: Bronze
    Qos:
      minIOPS: 1000
      maxIOPS: 2000
      burstIOPS: 4000
  - Type: Silver
    Qos:
      minIOPS: 4000
      maxIOPS: 6000
      burstIOPS: 8000
  - Type: Gold
    Qos:
      minIOPS: 6000
      maxIOPS: 8000
      burstIOPS: 10000
```

示例 **2**：具有虚拟池的 `solidfire-san` 驱动程序的后端和存储类配置

此示例显示了使用虚拟池配置的后端定义文件以及引用它们的 StorageClasses。

Trident 在配置时将存储池上的标签复制到后端存储 LUN。为方便起见，存储管理员可以为每个虚拟池定义标签，并按标签对卷进行分组。

在下面显示的示例后端定义文件中，为所有存储池设置了特定的默认值，将 `type`设置为 Silver。虚拟池在 `storage`部分中定义。在此示例中，一些存储池设置了自己的类型，一些池覆盖了上面设置的默认值。

```
---
version: 1
storageDriverName: solidfire-san
Endpoint: https://<user>:<password>@<mvip>/json-rpc/8.0
SVIP: <svip>:3260
TenantName: <tenant>
UseCHAP: true
Types:
```

```
    - Type: Bronze
      Qos:
        minIOPS: 1000
        maxIOPS: 2000
        burstIOPS: 4000
    - Type: Silver
      Qos:
        minIOPS: 4000
        maxIOPS: 6000
        burstIOPS: 8000
    - Type: Gold
      Qos:
        minIOPS: 6000
        maxIOPS: 8000
        burstIOPS: 10000
  type: Silver
  labels:
    store: solidfire
    k8scluster: dev-1-cluster
  region: us-east-1
  storage:
    - labels:
        performance: gold
        cost: "4"
      zone: us-east-1a
      type: Gold
    - labels:
        performance: silver
        cost: "3"
      zone: us-east-1b
      type: Silver
    - labels:
        performance: bronze
        cost: "2"
      zone: us-east-1c
      type: Bronze
    - labels:
        performance: silver
        cost: "1"
      zone: us-east-1d
```

以下 StorageClass 定义引用上述虚拟池。使用 `parameters.selector` 字段，每个 StorageClass 调用可用于托管卷的虚拟池。卷将具有所选虚拟池中定义的方面。

第一个 StorageClass (`solidfire-gold-four` 将映射到第一个虚拟池。这是唯一提供黄金性能的池，具有 `Volume Type QoS` 为 Gold。最后一个 StorageClass (`solidfire-silver` 调用任何提供银色性能的存储

池。Trident 将决定选择哪个虚拟池，并确保满足存储要求。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold-four
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=gold; cost=4
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-three
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=3
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-bronze-two
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=bronze; cost=2
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver-one
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver; cost=1
  fsType: ext4

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-silver
```

```
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=silver
  fsType: ext4
```

查找更多信息

- "卷访问组"

ONTAP SAN 驱动程序

ONTAP SAN 驱动程序概述

了解如何使用 ONTAP 和 Cloud Volumes ONTAP SAN 驱动程序配置 ONTAP 后端。

ONTAP SAN 驱动程序详细信息

Trident 提供以下 SAN 存储驱动程序以与 ONTAP 集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`ontap-san`	iSCSI SCSI over FC	块	RWO、ROX、RWX、RWOP	无文件系统；原始块设备
`ontap-san`	iSCSI SCSI over FC	Filesystem	RWO、RWOP ROX 和 RWX 在文件系统卷模式下不可用。	`xfs`, `ext3`, `ext4`
`ontap-san`	NVMe/TCP 请参见 NVMe/TCP 的其他注意事项。	块	RWO、ROX、RWX、RWOP	无文件系统；原始块设备
`ontap-san`	NVMe/TCP 请参见 NVMe/TCP 的其他注意事项。	Filesystem	RWO、RWOP ROX 和 RWX 在文件系统卷模式下不可用。	`xfs`, `ext3`, `ext4`
`ontap-san-economy`	iSCSI	块	RWO、ROX、RWX、RWOP	无文件系统；原始块设备

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`ontap-san-economy`	iSCSI	Filesystem	RWO、RWOP ROX 和 RWX 在文件系统卷模式下不可用。	`xfs`, `ext3`, `ext4`

警告

- 仅当预期持久卷使用次数高于"支持的 ONTAP 卷限制"时，才使用 `ontap-san-economy`。
- 仅当预期持久卷使用次数高于"支持的 ONTAP 卷限制"且无法使用 `ontap-san-economy`驱动程序时，才使用 `ontap-nas-economy`。
- 如果您预计需要数据保护、灾难恢复或移动性，请勿使用 `ontap-nas-economy`。
- NetApp 不建议在所有 ONTAP 驱动程序中使用 Flexvol 自动增长，除了 ontap-san。作为一种解决方法，Trident 支持使用快照保留并相应地扩展 Flexvol 卷。

用户权限

Trident 希望以 ONTAP 或 SVM 管理员的身份运行，通常使用 `admin`集群用户或 `vsadmin`SVM 用户，或具有相同角色的不同名称的用户。对于 Amazon FSx for NetApp ONTAP 部署，Trident 希望以 ONTAP 或 SVM 管理员的身份运行，使用集群 `fsxadmin`用户或 `vsadmin`SVM 用户，或具有相同角色的不同名称的用户。`fsxadmin`用户是集群管理员用户的有限替代品。

备注

如果使用 `limitAggregateUsage` 参数，则需要群集管理员权限。将 Amazon FSx for NetApp ONTAP 与 Trident 结合使用时， `limitAggregateUsage` 参数不适用于 `vsadmin` 和 `fsxadmin` 用户帐户。如果指定此参数，配置操作将失败。

虽然可以在 ONTAP 中创建 Trident 驱动程序可以使用的更具限制性的角色，但我们不建议这样做。大多数新版本的 Trident 会调用必须考虑到的其他 API，这使得升级变得困难且容易出错。

NVMe/TCP 的其他注意事项

Trident 支持非易失性存储器 Express (NVMe) 协议，使用 `ontap-san`驱动程序，包括：

- IPv6
- NVMe 卷的快照和克隆
- 调整 NVMe 卷的大小
- 导入在 Trident 之外创建的 NVMe 卷，以便 Trident 可以管理其生命周期
- NVMe 原生多路径
- K8s 节点的优雅或不优雅关闭（24.06）

Trident 不支持：

- NVMe 本机支持的 DH-HMAC-CHAP
- 设备映射器 (DM) 多路径
- LUKS 加密

备注	仅 ONTAP REST API 支持 NVMe，ONTAPI (ZAPI) 不支持 NVMe。

准备使用 **ONTAP SAN** 驱动程序配置后端

了解使用 ONTAP SAN 驱动程序配置 ONTAP 后端的要求和身份验证选项。

要求

对于所有 ONTAP 后端，Trident 要求至少将一个聚合分配给 SVM。

备注	"ASA r2 系统" 不同于其他 ONTAP 系统（ASA、AFF 和 FAS）的存储层实现。在 ASA r2 系统中，使用存储可用区而不是聚合。请参阅 "此" 知识库文章，了解如何在 ASA r2 系统中为 SVM 分配聚合。

请记住，您还可以运行多个驱动程序，并创建指向一个或另一个的存储类。例如，您可以配置一个 `san-dev`类，该类使用 `ontap-san`驱动程序，以及一个 `san-default`类，该类使用 `ontap-san-economy`驱动程序。

所有 Kubernetes 工作节点都必须安装相应的 iSCSI 工具。有关详细信息，请参见 "准备工作节点"。

对 **ONTAP** 后端进行身份验证

Trident 提供两种身份验证 ONTAP 后端的模式。

- 基于凭据：具有所需权限的 ONTAP 用户的用户名和密码。建议使用预定义的安全登录角色，例如 `admin`或 `vsadmin`以确保与 ONTAP 版本的最大兼容性。
- 基于证书：Trident 还可以使用后端安装的证书与 ONTAP 集群进行通信。此处，后端定义必须包含客户端证书、密钥和可信 CA 证书的 Base64 编码值（如果使用）（推荐）。

您可以更新现有后端以在基于凭据和基于证书的方法之间移动。但是，一次仅支持一种身份验证方法。要切换到其他身份验证方法，必须从后端配置中删除现有方法。

警告	如果您尝试提供*凭据和证书*，则后端创建将失败，错误为配置文件中提供了多个身份验证方法。

启用基于凭据的身份验证

Trident 需要向 SVM 范围/集群范围的管理员提供凭据，以便与 ONTAP 后端进行通信。建议使用标准、预定义的角色，如 `admin`或 `vsadmin`。这确保了与未来 ONTAP 版本的向前兼容性，这些版本可能会公开未来 Trident 版本使用的功能 API。可以创建自定义安全登录角色并与 Trident 一起使用，但不建议这样做。

示例后端定义如下所示：

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: password
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password"
}
```

请记住，后端定义是凭据以纯文本形式存储的唯一位置。后端创建后，用户名/密码使用 Base64 进行编码，并存储为 Kubernetes 密码。创建或更新后端是唯一需要了解凭据的步骤。因此，它是一个仅限管理员的操作，由 Kubernetes/存储管理员执行。

启用基于证书的身份验证

新的和现有的后端可以使用证书并与 ONTAP 后端通信。后端定义中需要三个参数。

- clientCertificate: 客户端证书的 Base64 编码值。
- clientPrivateKey: 关联专用密钥的 Base64 编码值。
- trustedCACertificate：受信任的 CA 证书的 Base64 编码值。如果使用受信任的 CA，则必须提供此参数。如果未使用受信任的 CA，则可以忽略此设置。

典型的工作流程包括以下步骤。

步骤

1. 生成客户端证书和密钥。生成时，将公用名 (CN) 设置为要进行身份验证的 ONTAP 用户。

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=admin"
```

2. 将受信任的 CA 证书添加到 ONTAP 集群。这可能已由存储管理员处理。如果未使用受信任的 CA，则忽略。

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. 在 ONTAP 集群上安装客户端证书和密钥（来自步骤 1）。

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

备注	运行此命令后，ONTAP 提示输入证书。粘贴步骤 1 中生成的 `k8senv.pem`文件内容，然后输入 `END`以完成安装。

4. 确认 ONTAP 安全登录角色支持 `cert`身份验证方法。

```
security login create -user-or-group-name admin -application ontapi
-authentication-method cert
security login create -user-or-group-name admin -application http
-authentication-method cert
```

5. 使用生成的证书测试身份验证。将 <ONTAP Management LIF> 和 <vserver name> 替换为管理 LIF IP 和 SVM 名称。

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. 使用 Base64 对证书、密钥和可信 CA 证书进行编码。

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 使用从上一步获得的值创建后端。

```
cat cert-backend.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"trustedCACertificate": "QNFinfO...SiqOyN",
"storagePrefix": "myPrefix_"
}

tridentctl create backend -f cert-backend.json -n trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       0 |
+------------+----------------+--------------------------------------
+--------+---------+
```

更新身份验证方法或轮换凭据

您可以更新现有后端以使用不同的身份验证方法或轮换其凭据。这可以双向工作：可以将使用用户名/密码的后端更新为使用证书；可以将使用证书的后端更新为基于用户名/密码。为此，您必须删除现有的身份验证方法并添加新的身份验证方法。然后使用更新的 backend.json 文件，其中包含执行 `tridentctl backend update` 所需的参数。

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend SanBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| SanBackend | ontap-san      | 586b1cd5-8cf8-428d-a76c-2872713612c1 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

备注	轮换密码时，存储管理员必须首先更新 ONTAP 上用户的密码。然后进行后端更新。轮换证书时，可以向用户添加多个证书。然后更新后端以使用新证书，之后可以从 ONTAP 集群中删除旧证书。

更新后端不会中断对已创建卷的访问，也不会影响之后建立的卷连接。成功的后端更新表明，Trident 可以与 ONTAP 后端通信并处理未来的卷操作。

为 **Trident** 创建自定义 **ONTAP** 角色

您可以使用最低权限创建 ONTAP 集群角色，这样您就不必使用 ONTAP 管理员角色在 Trident 中执行操作。在 Trident 后端配置中包含用户名时，Trident 使用您创建的 ONTAP 集群角色来执行操作。

有关创建 Trident 自定义角色的详细信息，请参见 "Trident 自定义角色生成器"。

使用 **ONTAP CLI**

1. 使用以下命令创建新角色：

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. 为 Trident 用户创建用户名：

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> -vserver
<svm_name\> -comment "user_description"
```

3. 将角色映射到用户：

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

使用 **System Manager**

在 ONTAP System Manager 中执行以下步骤：

1. 创建自定义角色：
   a. 要在集群级别创建自定义角色，请选择 **Cluster > Settings**。

      （或）要在 SVM 级别创建自定义角色，请选择*存储 > Storage VM > `required SVM`> 设置 > 用户和角色*。

   b. 选择 **Users and Roles** 旁边的箭头图标 (→)。
   c. 在 **Roles** 下选择 **+Add**。
   d. 定义角色的规则并单击 **Save**。
2. 将角色映射到 **Trident** 用户：+ 在*用户和角色*页面上执行以下步骤：
   a. 选择 **Users** 下的添加图标 **+**。
   b. 选择所需的用户名，然后在 **Role** 下拉菜单中选择一个角色。
   c. 单击 **Save**。

有关详细信息，请参见以下页面：

- "用于管理 ONTAP 的自定义角色" 或 "定义自定义角色"
- "使用角色和用户"

使用双向 **CHAP** 验证连接

Trident 可以使用 `ontap-san`和 `ontap-san-economy`驱动程序的双向 CHAP 对 iSCSI 会话进行身份验证。这需要在后端定义中启用 `useCHAP`选项。当设置为 `true`时，Trident 将 SVM 的默认启动器安全配置为双向 CHAP，并从后端文件设置用户名和密码。NetApp 建议使用双向 CHAP 来验证连接。请参见以下配置示例：

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
```

警告	useCHAP 参数是一个布尔选项，只能配置一次。默认设置为 false。将其设置为 true 后，无法将其设置为 false。

除了 useCHAP=true`之外，后端定义中还必须包含 `chapInitiatorSecret、chapTargetInitiatorSecret、 `chapTargetUsername`和 `chapUsername`字段。创建后端后，可以通过运行 `tridentctl update`来更改密钥。

工作原理

通过设置 `useCHAP`为 true，存储管理员指示 Trident 在存储后端上配置 CHAP。其中包括以下内容：

- 在 SVM 上设置 CHAP：
  - 如果 SVM 的默认启动器安全类型为 none（默认设置）*和*卷中没有预先存在的 LUN，Trident 会将默认安全类型设置为 `CHAP`并继续配置 CHAP 启动器以及目标用户名和密码。
  - 如果 SVM 包含 LUN，Trident 将不会在 SVM 上启用 CHAP。这可确保对 SVM 上已存在的 LUN 的访问不受限制。
- 配置 CHAP 启动器以及目标用户名和密码；这些选项必须在后端配置中指定（如上所示）。

创建后端后，Trident 创建相应的 tridentbackend CRD，并将 CHAP secrets 和用户名存储为 Kubernetes secrets。由 Trident 在此后端上创建的所有 PV 都将通过 CHAP 进行挂载和连接。

轮换凭据并更新后端

您可以通过更新 backend.json 文件中的 CHAP 参数来更新 CHAP 凭据。这将需要更新 CHAP 密码并使用 tridentctl update 命令来反映这些更改。

警告	更新后端的 CHAP 密码时，必须使用 `tridentctl`来更新后端。请勿使用 ONTAP CLI 或 ONTAP System Manager 更新存储集群上的凭据，因为 Trident 将无法获取这些更改。

```
cat backend-san.json
{
    "version": 1,
    "storageDriverName": "ontap-san",
    "backendName": "ontap_san_chap",
    "managementLIF": "192.168.0.135",
    "svm": "ontap_iscsi_svm",
    "useCHAP": true,
    "username": "vsadmin",
    "password": "password",
    "chapInitiatorSecret": "cl9qxUpDaTeD",
    "chapTargetInitiatorSecret": "rqxigXgkeUpDaTeD",
    "chapTargetUsername": "iJF4heBRT0TCwxyz",
    "chapUsername": "uh2aNCLSd6cNwxyz",
}

./tridentctl update backend ontap_san_chap -f backend-san.json -n trident
+----------------+----------------+--------------------------------------
+--------+---------+
|   NAME         | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+----------------+----------------+--------------------------------------
+--------+---------+
| ontap_san_chap | ontap-san      | aa458f3b-ad2d-4378-8a33-1a472ffbeb5c |
online |       7 |
+----------------+----------------+--------------------------------------
+--------+---------+
```

现有连接将不受影响；如果 Trident 在 SVM 上更新了凭据，它们将继续保持活动状态。新连接使用更新的凭据，现有连接继续保持活动状态。断开和重新连接旧的 PV 将导致它们使用更新的凭据。

ONTAP SAN 配置选项和示例

了解如何在 Trident 安装中创建和使用 ONTAP SAN 驱动程序。本节提供了将后端映射到 StorageClasses 的后端配置示例和详细信息。"ASA r2 系统" 不同于其他 ONTAP 系统（ASA、AFF 和 FAS）的存储层实现。这些变化会影响某些标注参数的使用。"详细了解 ASA r2 系统与其他 ONTAP 系统之间的差异".在 Trident 后端配置中，无需指定您的系统是 ASA r2。当您选择 `ontap-san`作为 `storageDriverName`时，Trident 会自动检测 ASA r2 或其他 ONTAP 系统。某些后端配置参数不适用于 ASA r2 系统，如下表所示。

备注	ASA r2 系统仅支持 `ontap-san` 驱动程序（具有 iSCSI、NVMe/TCP 和 FC 协议）。

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`ontap-san` 或 `ontap-san-economy`
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + dataLIF
`managementLIF`	集群或 SVM 管理 LIF 的 IP 地址。 可以指定完全限定的域名 (FQDN)。 如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。 有关无缝 MetroCluster 切换，请参见 MetroCluster 示例。 备注：如果使用 "vsadmin" 凭据，`managementLIF` 必须是 SVM 的凭据；如果使用 "admin" 凭据，`managementLIF` 必须是集群的凭据。	“10.0.0.1”，“[2001:1234:abcd::fefe]”
`dataLIF`	协议 LIF 的 IP 地址。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。*不为 iSCSI 指定。*Trident 使用"ONTAP 选择性 LUN 映射"来发现建立多路径会话所需的 iSCSI LIF。如果明确定义了 `dataLIF`，则会生成警告。*省略 MetroCluster。*请参阅MetroCluster 示例。	由 SVM 派生
`svm`	要使用的 Storage Virtual Machine *对于 MetroCluster 请省略。*请参阅MetroCluster 示例。	如果指定了 SVM `managementLIF`，则派生
`useCHAP`	使用 CHAP 对 ONTAP SAN 驱动程序的 iSCSI 进行身份验证 [Boolean]。设置为 `true`，Trident 将配置和使用双向 CHAP 作为后端给定的 SVM 的默认身份验证。有关详细信息，请参见 "准备使用 ONTAP SAN 驱动程序配置后端"。不支持 **FCP** 或 **NVMe/TCP**。	`false`
`chapInitiatorSecret`	CHAP 启动器密钥。如果 `useCHAP=true`为必需	""
`labels`	要应用于卷的任意 JSON 格式标签集	""
`chapTargetInitiatorSecret`	CHAP 目标发起者密钥。如果 `useCHAP=true`为必需	""
`chapUsername`	入站用户名。如果 `useCHAP=true`为必需	""

参数	说明	默认
chapTargetUsername	目标用户名。如果 `useCHAP=true`为必需	""
clientCertificate	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端私钥的 Base64 编码值。用于基于证书的身份验证	""
trustedCACertificate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""
username	与 ONTAP 集群通信所需的用户名。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅"使用 Active Directory 凭据向后端 SVM 验证 Trident"。	""
password	与 ONTAP 集群通信所需的密码。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅"使用 Active Directory 凭据向后端 SVM 验证 Trident"。	""
svm	要使用的 Storage Virtual Machine	如果指定了 SVM managementLIF，则派生
storagePrefix	在 SVM 中配置新卷时使用的前缀。以后无法修改。要更新此参数，您需要创建一个新的后端。	trident
aggregate	用于配置的聚合（可选；如果设置，则必须分配给 SVM）。对于 ontap-nas-flexgroup 驱动程序，此选项将被忽略。如果未分配，则可以使用任何可用的聚合来配置 FlexGroup 卷。 备注：当聚合在 SVM 中更新时，它会通过轮询 SVM 在 Trident 中自动更新，而无需重新启动 Trident Controller。当您在 Trident 中配置了特定聚合来配置卷时，如果聚合被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将在 Trident 中移至失败状态。您必须将聚合更改为存在于 SVM 上的聚合，或者将其完全删除，以使后端恢复联机。 请勿为 **ASA r2** 系统指定。	""
limitAggregateUsage	如果使用率超过此百分比，则配置失败。如果您使用的是 Amazon FSx for NetApp ONTAP 后端，请不要指定 limitAggregateUsage。提供的 `fsxadmin`和 `vsadmin`不包含检索聚合使用情况并使用 Trident 限制它所需的权限。请勿为 **ASA r2** 系统指定。	"（默认情况下不强制执行）
limitVolumeSize	如果请求的卷大小高于此值，则设置失败。还限制它为 LUN 管理的卷的最大大小。	"（默认情况下不强制执行）
lunsPerFlexvol	每个 FlexVol 的最大 LUN 数，必须在 [50, 200] 范围内	100

参数	说明	默认
debugTraceFlags	故障排除时使用的调试标志。例如，{"api":false, "method":true} 除非正在进行故障排除并需要详细的日志转储，否则不要使用。	null
useREST	使用 ONTAP REST API 的布尔参数。 `useREST`设置为 `true`时，Trident 使用 ONTAP REST API 与后端通信；设置为 `false`时，Trident 使用 ONTAPI (ZAPI) 调用与后端通信。此功能需要 ONTAP 9.11.1 及更高版本。此外，所使用的 ONTAP 登录角色必须能够访问 `ontapi`应用程序。这通过预定义的 `vsadmin`和 `cluster-admin`角色来满足。从 Trident 24.06 版本和 ONTAP 9.15.1 或更高版本开始，`useREST`默认设置为 `true`；将 `useREST`更改为 `false`以使用 ONTAPI (ZAPI) 调用。 useREST 完全符合 NVMe/TCP。 备注：仅 ONTAP REST API 支持 NVMe，ONTAPI (ZAPI) 不支持 NVMe。 如果指定，则对于 **ASA r2** 系统始终设置为 **true**。	true 适用于 ONTAP 9.15.1 或更高版本，否则 false。
sanType	用于选择 iscsi iSCSI、 nvme NVMe/TCP 或 fcp 光纤通道 (FC) 上的 SCSI。	iscsi 如果为空
formatOptions	使用 formatOptions 为 mkfs 命令指定命令行参数，该参数将在卷格式化时应用。这允许您根据您的偏好格式化卷。请务必指定与 mkfs 命令选项类似的 formatOptions，但不包括设备路径。例如："-E nodiscard" 支持 **ontap-san** 和 **ontap-san-economy** 驱动程序的 **iSCSI** 协议。*此外，使用 iSCSI 和 NVMe/TCP 协议时，支持 ASA r2 系统。*	
limitVolumePoolSize	在 ontap-san-economy 后端中使用 LUN 时可请求的最大 FlexVol 大小。	"（默认情况下不强制执行）
denyNewVolumePools	限制 `ontap-san-economy`后端创建包含其 LUN 的新 FlexVol 卷。仅预先存在的 FlexVol 用于配置新的 PV。	

有关使用 **formatOptions** 的建议

Trident 建议使用以下选项来加快格式化过程：

- **-E nodiscard (ext3, ext4):** 不要尝试在 mkfs 时间丢弃块（丢弃块最初在固态设备和稀疏/精简配置的存储上很有用）。这将替换已弃用的选项 "-K"，并且适用于 ext3、ext4 文件系统。
- **-K (xfs):** 不要尝试在 mkfs 时间丢弃块。此选项适用于 xfs 文件系统。

使用 **Active Directory** 凭据向后端 **SVM** 验证 **Trident**

您可以配置 Trident 使用 Active Directory (AD) 凭据向后端 SVM 进行身份验证。在 AD 帐户可以访问 SVM 之前，必须配置 AD 域控制器对集群或 SVM 的访问。对于使用 AD 帐户的集群管理，必须创建域隧道。有关详细信息，请参见 "在 ONTAP 中配置 Active Directory 域控制器访问"。

步骤

1. 配置后端 SVM 的域名系统 (DNS) 设置：

```
vserver services dns create -vserver <svm_name> -dns-servers
<dns_server_ip1>,<dns_server_ip2>
```

2. 运行以下命令为 Active Directory 中的 SVM 创建计算机帐户：

```
vserver active-directory create -vserver DataSVM -account-name ADSERVER1
-domain demo.netapp.com
```

3. 使用此命令创建 AD 用户或组以管理集群或 SVM

```
security login create -vserver <svm_name> -user-or-group-name
<ad_user_or_group> -application <application> -authentication-method domain
-role vsadmin
```

4. 在 Trident 后端配置文件中，将 `username` 和 `password` 参数分别设置为 AD 用户或组名称和密码。

用于配置卷的后端配置选项

您可以使用配置的 `defaults` 部分中的这些选项来控制默认配置。有关示例，请参阅下面的配置示例。

参数	说明	默认
`spaceAllocation`	LUN 的空间分配	"true" 如果指定，则对于 **ASA r2** 系统设置为 **`true`**。
`spaceReserve`	空间预留模式；"none"（精简）或"volume"（厚）。对于 **ASA r2** 系统，设置为 **`none`**。	"无"
`snapshotPolicy`	要使用的 Snapshot 策略。对于 **ASA r2** 系统设置为 **`none`**。	"无"

参数	说明	默认
`qosPolicy`	要为创建的卷分配的 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。在 Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。您应该使用非共享 QoS 策略组，并确保该策略组单独应用于每个组成部分。共享 QoS 策略组强制执行所有工作负载总吞吐量的上限。	""
`adaptiveQosPolicy`	要为创建的卷分配的自适应 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个	""
`snapshotReserve`	为快照保留的卷的百分比。请勿为 **ASA r2** 系统指定。	"0" 如果 `snapshotPolicy` 为 "none"，否则为 "
`splitOnClone`	创建时从其父级拆分克隆	"false"
`encryption`	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 `false`。必须在群集上许可并启用 NVE 才能使用此选项。如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。有关更多信息，请参阅：“Trident 如何与 NVE 和 NAE 配合使用”。	"false" 如果指定，则对于 **ASA r2** 系统设置为 **`true`**。
`luksEncryption`	启用 LUKS 加密。请参见 “使用 Linux Unified Key Setup (LUKS)”。	" 对于 **ASA r2** 系统，设置为 **`false`**。
`tieringPolicy`	使用 "none" 的分层策略 不要为 **ASA r2** 系统指定。	
`nameTemplate`	用于创建自定义卷名称的模板。	""

卷配置示例

以下是定义了默认值的示例：

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: trident_svm
username: admin
password: <password>
labels:
  k8scluster: dev2
  backend: dev2-sanbackend
storagePrefix: alternate-trident
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: standard
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

备注	对于使用 `ontap-san` 驱动程序创建的所有卷，Trident 会为 FlexVol 增加 10% 的额外容量以容纳 LUN 元数据。LUN 将使用用户在 PVC 中请求的确切大小进行配置。Trident 为 FlexVol 增加 10%（在 ONTAP 中显示为可用大小）。用户现在将获得他们请求的可用容量。此更改还可防止 LUN 变为只读，除非可用空间得到充分利用。这不适用于 ontap-san-economy。

对于定义 `snapshotReserve` 的后端，Trident 计算卷的大小如下：

```
Total volume size = [(PVC requested size) / (1 - (snapshotReserve
percentage) / 100)] * 1.1
```

1.1 是 Trident 添加到 FlexVol 以容纳 LUN 元数据的额外 10%。对于 `snapshotReserve` = 5%，且 PVC 请求 = 5 GiB，总体积大小为 5.79 GiB，可用大小为 5.5 GiB。 `volume show`命令应显示类似于以下示例的结果：

当前，对现有卷使用新计算的唯一方法是调整大小。

最小配置示例

以下示例显示了将大多数参数保留为默认值的基本配置。这是定义后端的最简单方法。

备注	如果您在 Trident 中使用 Amazon FSx for NetApp ONTAP，NetApp 建议为 LIF 指定 DNS 名称而不是 IP 地址。

ONTAP SAN 示例

这是使用 `ontap-san` 驱动程序的基本配置。

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
username: vsadmin
password: <password>
```

MetroCluster 示例

您可以配置后端，以避免在 "SVM 复制和恢复" 期间进行切换和切换后手动更新后端定义。

对于无缝切换和切回，使用 `managementLIF` 指定 SVM 并省略 `svm` 参数。例如：

```
version: 1
storageDriverName: ontap-san
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

ONTAP SAN 经济示例

```
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
username: vsadmin
password: <password>
```

基于证书的身份验证示例

在此基本配置示例中， `clientCertificate`、 `clientPrivateKey` 和 `trustedCACertificate`（可选，如果使用受信任的 CA）填充在 `backend.json` 中，并分别采用客户端证书、私钥和受信任 CA 证书的 base64 编码值。

```
---
version: 1
storageDriverName: ontap-san
backendName: DefaultSANBackend
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
```

双向 **CHAP** 示例

这些示例创建一个后端，其中 `useCHAP` 设置为 `true`。

ONTAP SAN CHAP 示例

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
labels:
  k8scluster: test-cluster-1
  backend: testcluster1-sanbackend
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

ONTAP SAN economy CHAP 示例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
```

NVMe/TCP 示例

必须在 ONTAP 后端上配置具有 NVMe 的 SVM。这是 NVMe/TCP 的基本后端配置。

```
---
version: 1
backendName: NVMeBackend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_nvme
username: vsadmin
password: password
sanType: nvme
useREST: true
```

基于 FC 的 SCSI (FCP) 示例

您必须在 ONTAP 后端上配置带有 FC 的 SVM。这是 FC 的基本后端配置。

```
---
version: 1
backendName: fcp-backend
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_fc
username: vsadmin
password: password
sanType: fcp
useREST: true
```

带有 **nameTemplate** 的后端配置示例

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap-san-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

formatOptions 示例，适用于 **ontap-san-economy** 驱动程序

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: ""
svm: svm1
username: ""
password: "!"
storagePrefix: whelk_
debugTraceFlags:
  method: true
  api: true
defaults:
  formatOptions: -E nodiscard
```

具有虚拟池的后端示例

在这些示例后端定义文件中，为所有存储池设置了特定的默认值，例如 `spaceReserve` 为 none、`spaceAllocation` 为 false 和 `encryption` 为 false。虚拟池在存储部分中定义。

Trident 在"Comments"字段中设置配置标签。注释在 FlexVol 卷上设置，Trident 在配置时将虚拟池中存在的所有标签复制到存储卷。为方便起见，存储管理员可以为每个虚拟池定义标签，并按标签对卷进行分组。

在这些示例中，一些存储池设置了自己的 `spaceReserve`、 `spaceAllocation` 和 `encryption` 值，而一些

池覆盖了默认值。

ONTAP SAN 示例

```
---
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
svm: svm_iscsi
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
  qosPolicy: standard
labels:
  store: san_store
  kubernetes-cluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "40000"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
      adaptiveQosPolicy: adaptive-extreme
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
      qosPolicy: premium
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

ONTAP SAN 经济示例

```
---
version: 1
storageDriverName: ontap-san-economy
managementLIF: 10.0.0.1
svm: svm_iscsi_eco
useCHAP: true
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
username: vsadmin
password: <password>
defaults:
  spaceAllocation: "false"
  encryption: "false"
labels:
  store: san_economy_store
region: us_east_1
storage:
  - labels:
      app: oracledb
      cost: "30"
    zone: us_east_1a
    defaults:
      spaceAllocation: "true"
      encryption: "true"
  - labels:
      app: postgresdb
      cost: "20"
    zone: us_east_1b
    defaults:
      spaceAllocation: "false"
      encryption: "true"
  - labels:
      app: mysqldb
      cost: "10"
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
  - labels:
      department: legal
      creditpoints: "5000"
```

```
    zone: us_east_1c
    defaults:
      spaceAllocation: "true"
      encryption: "false"
```

NVMe/TCP 示例

```
---
version: 1
storageDriverName: ontap-san
sanType: nvme
managementLIF: 10.0.0.1
svm: nvme_svm
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceAllocation: "false"
  encryption: "true"
storage:
  - labels:
      app: testApp
      cost: "20"
    defaults:
      spaceAllocation: "false"
      encryption: "false"
```

将后端映射到 **StorageClasses**

以下 StorageClass 定义请参阅 [具有虚拟池的后端示例]。使用 `parameters.selector` 字段，每个 StorageClass 调用哪些虚拟池可用于托管卷。卷将具有所选虚拟池中定义的方面。

- `protection-gold` StorageClass 将映射到 `ontap-san`后端中的第一个虚拟池。这是唯一提供黄金级保护的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- `protection-not-gold` StorageClass 将映射到 `ontap-san`后端的第二个和第三个虚拟池。这些是唯一提供黄金以外保护级别的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- `app-mysqldb` StorageClass 将映射到 `ontap-san-economy`后端的第三个虚拟池。这是唯一为 mysqldb 类型应用程序提供存储池配置的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- `protection-silver-creditpoints-20k` StorageClass 将映射到 `ontap-san` 后端的第二个虚拟池。这是唯一提供银级保护和 20000 creditpoints 的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- `creditpoints-5k` StorageClass 将映射到 `ontap-san`后端的第三个虚拟池和 `ontap-san-economy`后端的第四个虚拟池。这些是唯一拥有 5000 个信用点的池产品。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

- 该 `my-test-app-sc` StorageClass 将映射到 `testAPP` 驱动程序中的 `ontap-san` 虚拟池，并带有 `sanType: nvme`。这是唯一提供 `testApp` 的池。

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: my-test-app-sc
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=testApp"
  fsType: "ext4"
```

Trident 将决定选择哪个虚拟池，并确保满足存储要求。

ONTAP NAS 驱动程序

ONTAP NAS 驱动程序概述

了解如何使用 ONTAP 和 Cloud Volumes ONTAP NAS 驱动程序配置 ONTAP 后端。

ONTAP NAS 驱动程序详细信息

Trident 提供以下 NAS 存储驱动程序以与 ONTAP 集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`ontap-nas`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	", `nfs`, `smb`
`ontap-nas-economy`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	", `nfs`, `smb`
`ontap-nas-flexgroup`	NFS SMB	Filesystem	RWO、ROX、RWX、RWOP	", `nfs`, `smb`

警告

- 仅当预期持久卷使用次数高于"支持的 ONTAP 卷限制"时，才使用 `ontap-san-economy`。
- 仅当预期持久卷使用次数高于"支持的 ONTAP 卷限制"且无法使用 `ontap-san-economy`驱动程序时，才使用 `ontap-nas-economy`。
- 如果您预计需要数据保护、灾难恢复或移动性，请勿使用 `ontap-nas-economy`。
- NetApp 不建议在所有 ONTAP 驱动程序中使用 Flexvol 自动增长，除了 ontap-san。作为一种解决方法，Trident 支持使用快照保留并相应地扩展 Flexvol 卷。

用户权限

Trident 希望以 ONTAP 或 SVM 管理员的身份运行，通常使用 `admin`集群用户或 `vsadmin`SVM 用户，或具有相同角色的不同名称的用户。

对于 Amazon FSx for NetApp ONTAP 部署，Trident 希望以 ONTAP 或 SVM 管理员的身份运行，使用集群 `fsxadmin`用户或 `vsadmin`SVM 用户，或具有相同角色的不同名称的用户。 `fsxadmin`用户是集群管理员用户的有限替代品。

备注

如果使用 `limitAggregateUsage` 参数，则需要群集管理员权限。将 Amazon FSx for NetApp ONTAP 与 Trident 结合使用时， `limitAggregateUsage` 参数不适用于 `vsadmin` 和 `fsxadmin` 用户帐户。如果指定此参数，配置操作将失败。

虽然可以在 ONTAP 中创建 Trident 驱动程序可以使用的更具限制性的角色，但我们不建议这样做。大多数新版本的 Trident 会调用必须考虑到的其他 API，这使得升级变得困难且容易出错。

准备使用 ONTAP NAS 驱动程序配置后端

了解使用 ONTAP NAS 驱动程序配置 ONTAP 后端的要求、身份验证选项和导出策略。从 25.10 版本开始，NetApp Trident 支持 "NetApp AFX 存储系统"。NetApp AFX 存储系统与其他 ONTAP 系统（ASA、AFF 和 FAS）在存储层的实现方面有所不同。在 Trident 后端配置中，无需指定您的系统是 AFX。当您选择 `ontap-nas`作为 `storageDriverName`时，Trident 会自动检测 AFX 系统。

备注	AFX 系统仅支持 `ontap-nas`驱动程序（使用 NFS 协议）；不支持 SMB 协议。

要求

- 对于所有 ONTAP 后端，Trident 要求至少将一个聚合分配给 SVM。
- 您可以运行多个驱动程序，并创建指向一个或另一个的存储类。例如，您可以配置一个使用 `ontap-nas`驱动程序的 Gold 类和一个使用 `ontap-nas-economy`驱动程序的 Bronze 类。
- 所有 Kubernetes worker 节点都必须安装相应的 NFS 工具。有关更多详细信息，请参见 "此处"。
- Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。有关详细信息，请参见 准备配置 SMB 卷。

对 **ONTAP** 后端进行身份验证

Trident 提供两种身份验证 ONTAP 后端的模式。

- 基于凭据：此模式需要对 ONTAP 后端的足够权限。建议使用与预定义安全登录角色关联的帐户，例如 `admin`或 `vsadmin`，以确保与 ONTAP 版本的最大兼容性。
- 基于证书：此模式需要在后端安装证书，Trident 才能与 ONTAP 集群进行通信。此处，后端定义必须包含客户端证书、密钥和可信 CA 证书的 Base64 编码值（如果使用）（推荐）。

您可以更新现有后端以在基于凭据和基于证书的方法之间移动。但是，一次仅支持一种身份验证方法。要切换到其他身份验证方法，必须从后端配置中删除现有方法。

警告	如果您尝试提供*凭据和证书*，则后端创建将失败，错误为配置文件中提供了多个身份验证方法。

启用基于凭据的身份验证

Trident 需要向 SVM 范围/集群范围的管理员提供凭据，以便与 ONTAP 后端进行通信。建议使用标准、预定义的角色，如 `admin`或 `vsadmin`。这确保了与未来 ONTAP 版本的向前兼容性，这些版本可能会公开未来 Trident 版本使用的功能 API。可以创建自定义安全登录角色并与 Trident 一起使用，但不建议这样做。

示例后端定义如下所示：

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
credentials:
  name: secret-backend-creds
```

JSON

```json
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "credentials": {
        "name": "secret-backend-creds"
    }
}
```

请记住，后端定义是凭据以纯文本形式存储的唯一位置。后端创建后，用户名/密码使用 Base64 进行编码，并存储为 Kubernetes 密码。创建/更新后端是唯一需要了解凭据的步骤。因此，它是一个仅限管理员的操作，由 Kubernetes/存储管理员执行。

启用基于证书的身份验证

新的和现有的后端可以使用证书并与 ONTAP 后端通信。后端定义中需要三个参数。

- clientCertificate: 客户端证书的 Base64 编码值。
- clientPrivateKey: 关联专用密钥的 Base64 编码值。
- trustedCACertificate：受信任的 CA 证书的 Base64 编码值。如果使用受信任的 CA，则必须提供此参数。如果未使用受信任的 CA，则可以忽略此设置。

典型的工作流程包括以下步骤。

步骤

1. 生成客户端证书和密钥。生成时，将公用名 (CN) 设置为要进行身份验证的 ONTAP 用户。

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=vsadmin"
```

2. 将受信任的 CA 证书添加到 ONTAP 集群。这可能已由存储管理员处理。如果未使用受信任的 CA，则忽略。

```
security certificate install -type server -cert-name <trusted-ca-cert-
name> -vserver <vserver-name>
ssl modify -vserver <vserver-name> -server-enabled true -client-enabled
true -common-name <common-name> -serial <SN-from-trusted-CA-cert> -ca
<cert-authority>
```

3. 在 ONTAP 集群上安装客户端证书和密钥（来自步骤 1）。

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

4. 确认 ONTAP 安全登录角色支持 `cert`身份验证方法。

```
security login create -user-or-group-name vsadmin -application ontapi
-authentication-method cert -vserver <vserver-name>
security login create -user-or-group-name vsadmin -application http
-authentication-method cert -vserver <vserver-name>
```

5. 使用生成的证书测试身份验证。将 <ONTAP Management LIF> 和 <vserver name> 替换为管理 LIF IP 和 SVM 名称。您必须确保 LIF 的服务策略设置为 `default-data-management`。

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns="http://www.netapp.com/filer/admin" version="1.21"
vfiler="<vserver-name>"><vserver-get></vserver-get></netapp>'
```

6. 使用 Base64 对证书、密钥和可信 CA 证书进行编码。

```
base64 -w 0 k8senv.pem >> cert_base64
base64 -w 0 k8senv.key >> key_base64
base64 -w 0 trustedca.pem >> trustedca_base64
```

7. 使用从上一步获得的值创建后端。

```
cat cert-backend-updated.json
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"clientCertificate": "Faaaakkkkeeee...Vaaalllluuuueeee",
"clientPrivateKey": "LS0tFaKE...0VaLuES0tLS0K",
"storagePrefix": "myPrefix_"
}

#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

更新身份验证方法或轮换凭据

您可以更新现有后端以使用不同的身份验证方法或轮换其凭据。这可以双向工作：可以将使用用户名/密码的后端更新为使用证书；可以将使用证书的后端更新为基于用户名/密码。为此，您必须删除现有的身份验证方法并添加新的身份验证方法。然后使用更新的 backend.json 文件，其中包含执行 `tridentctl update backend` 所需的参数。

```
cat cert-backend-updated.json
```

```
{
"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "NasBackend",
"managementLIF": "1.2.3.4",
"dataLIF": "1.2.3.8",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}
```

```
#Update backend with tridentctl
tridentctl update backend NasBackend -f cert-backend-updated.json -n
trident
+------------+----------------+--------------------------------------
+--------+---------+
|    NAME    | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+------------+----------------+--------------------------------------
+--------+---------+
| NasBackend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       9 |
+------------+----------------+--------------------------------------
+--------+---------+
```

备注	轮换密码时，存储管理员必须首先更新 ONTAP 上用户的密码。然后进行后端更新。轮换证书时，可以向用户添加多个证书。然后更新后端以使用新证书，之后可以从 ONTAP 集群中删除旧证书。

更新后端不会中断对已创建卷的访问，也不会影响之后建立的卷连接。成功的后端更新表明，Trident 可以与 ONTAP 后端通信并处理未来的卷操作。

为 **Trident** 创建自定义 **ONTAP** 角色

您可以使用最低权限创建 ONTAP 集群角色，这样您就不必使用 ONTAP 管理员角色在 Trident 中执行操作。在 Trident 后端配置中包含用户名时，Trident 使用您创建的 ONTAP 集群角色来执行操作。

有关创建 Trident 自定义角色的详细信息，请参见 "Trident 自定义角色生成器"。

使用 **ONTAP CLI**

1. 使用以下命令创建新角色：

```
security login role create <role_name\> -cmddirname "command" -access all
-vserver <svm_name\>
```

2. 为 Trident 用户创建用户名：

```
security login create -username <user_name\> -application ontapi
-authmethod <password\> -role <name_of_role_in_step_1\> -vserver
<svm_name\> -comment "user_description"
```

3. 将角色映射到用户：

```
security login modify username <user_name\> -vserver <svm_name\> -role
<role_name\> -application ontapi -application console -authmethod
<password\>
```

使用 **System Manager**

在 ONTAP System Manager 中执行以下步骤：

1. 创建自定义角色：
   a. 要在集群级别创建自定义角色，请选择 **Cluster > Settings**。

      （或）要在 SVM 级别创建自定义角色，请选择*存储 > Storage VM > `required SVM`> 设置 > 用户和角色*。

   b. 选择 **Users and Roles** 旁边的箭头图标 (→)。
   c. 在 **Roles** 下选择 **+Add**。
   d. 定义角色的规则并单击 **Save**。
2. 将角色映射到 **Trident** 用户：+ 在*用户和角色*页面上执行以下步骤：
   a. 选择 **Users** 下的添加图标 **+**。
   b. 选择所需的用户名，然后在 **Role** 下拉菜单中选择一个角色。
   c. 单击 **Save**。

有关详细信息，请参见以下页面：

- "用于管理 ONTAP 的自定义角色" 或 "定义自定义角色"
- "使用角色和用户"

管理 NFS 导出策略

Trident 使用 NFS 导出策略来控制对其提供的卷的访问。

使用出口策略时，Trident 提供两种选择：

- Trident 可以动态管理导出策略本身；在这种操作模式下，存储管理员指定表示可允许 IP 地址的 CIDR 块列表。Trident 会在发布时自动将落在这些范围内的适用节点 IP 添加到导出策略中。或者，当未指定 CIDR 时，在要发布的卷的节点上找到的所有全局范围的单播 IP 都将添加到导出策略中。
- 存储管理员可以创建导出策略并手动添加规则。除非在配置中指定不同的导出策略名称，否则 Trident 使用默认导出策略。

动态管理导出策略

Trident 能够动态管理 ONTAP 后端的导出策略。这使存储管理员能够为工作节点 IP 指定允许的地址空间，而不是手动定义显式规则。它大大简化了导出策略管理；对导出策略的修改不再需要对存储集群进行手动干预。此外，这有助于将对存储集群的访问限制为仅挂载卷且 IP 位于指定范围内的工作节点，从而支持精细化和自动化管理。

备注	使用动态导出策略时不要使用网络地址转换 (NAT)。对于 NAT，存储控制器看到的是前端 NAT 地址，而不是实际的 IP 主机地址，因此在导出规则中未找到匹配项时将拒绝访问。

示例

必须使用两个配置选项。以下是后端定义示例：

```
---
version: 1
storageDriverName: ontap-nas-economy
backendName: ontap_nas_auto_export
managementLIF: 192.168.0.135
svm: svm1
username: vsadmin
password: password
autoExportCIDRs:
  - 192.168.0.0/24
autoExportPolicy: true
```

备注	使用此功能时，必须确保 SVM 中的根接合点具有先前创建的导出策略以及允许节点 CIDR 块的导出规则（例如默认导出策略）。始终遵循 NetApp 推荐的最佳实践，为 Trident 专用一个 SVM。

以下是使用上述示例说明此功能工作原理的解释：

- `autoExportPolicy` 设置为 `true`。这表示 Trident 为使用此后端为 `svm1` SVM 配置的每个卷创建导出策略，并使用 `autoexportCIDRs`地址块处理规则的添加和删除。在卷连接到节点之前，该卷使用没有规则的空导出策略来防止对该卷的不必要访问。当卷发布到节点时，Trident 会创建一个与底层 qtree 同名的导出策略，该 qtree 包含指定 CIDR 块中的节点 IP。这些 IP 也将被添加到父 FlexVol 卷使用的导出策略中
  - 例如：
    - 后端 UUID 403b5326-8482-40db-96d0-d83fb3f4daec
    - `autoExportPolicy` 设置为 `true`
    - 存储前缀 `trident`

- PVC UUID a79bcf5f-7b6d-4a40-9876-e2551f159c1c
- 名为 trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c 的 qtree 为名为 `trident-403b5326-8482-40db96d0-d83fb3f4daec`的 FlexVol 创建导出策略，为名为 `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`的 qtree 创建导出策略，并在 SVM 上创建名为 `trident_empty`的空导出策略。FlexVol 导出策略的规则将是 qtree 导出策略中包含的任何规则的超集。空导出策略将由未附加的任何卷重用。

- `autoExportCIDRs` 包含地址块列表。此字段是可选的，默认为 ["0.0.0.0/0", "::/0"]。如果未定义，Trident 会添加在工作节点上找到的所有全局范围单播地址及其发布。

在此示例中，提供了 `192.168.0.0/24` 地址空间。这表示此地址范围内包含发布的 Kubernetes 节点 IP 将被添加到 Trident 创建的导出策略中。当 Trident 注册其运行的节点时，它会检索节点的 IP 地址，并根据 `autoExportCIDRs` 中提供的地址块进行检查。发布时，过滤 IP 后，Trident 为其发布到的节点的客户端 IP 创建导出策略规则。

您可以在创建后端后对其 `autoExportPolicy`和 `autoExportCIDRs`进行更新。您可以为自动管理的后端追加新的 CIDR 或删除现有的 CIDR。删除 CIDR 时要小心，以确保现有连接不会丢失。您还可以选择对后端禁用 `autoExportPolicy`并回退到手动创建的导出策略。这将需要在后端配置中设置 `exportPolicy`参数。

在 Trident 创建或更新后端后，您可以使用 `tridentctl`或相应的 `tridentbackend` CRD 检查后端：

```
./tridentctl get backends ontap_nas_auto_export -n trident -o yaml
items:
- backendUUID: 403b5326-8482-40db-96d0-d83fb3f4daec
  config:
    aggregate: ""
    autoExportCIDRs:
    - 192.168.0.0/24
    autoExportPolicy: true
    backendName: ontap_nas_auto_export
    chapInitiatorSecret: ""
    chapTargetInitiatorSecret: ""
    chapTargetUsername: ""
    chapUsername: ""
    dataLIF: 192.168.0.135
    debug: false
    debugTraceFlags: null
    defaults:
      encryption: "false"
      exportPolicy: <automatic>
      fileSystemType: ext4
```

删除节点时，Trident 检查所有导出策略，以删除与该节点对应的访问规则。通过从托管后端的导出策略中删除此节点 IP，Trident 可以防止流氓挂载，除非此 IP 被集群中的新节点重用。

对于以前存在的后端，使用 `tridentctl update backend`更新后端可确保 Trident 自动管理导出策略。这会在需要时创建两个以后端 UUID 和 qtree 名称命名的新导出策略。后端上存在的卷在卸载并再次装载后将使用新创建的导出策略。

备注	删除具有自动管理导出策略的后端将删除动态创建的导出策略。如果重新创建后端，它将被视为新的后端，并将导致创建新的导出策略。

如果实时节点的 IP 地址已更新，则必须在节点上重新启动 Trident pod。然后，Trident 将更新其管理的后端的导出策略，以反映此 IP 更改。

准备配置 SMB 卷

通过一些额外的准备，您可以使用 `ontap-nas` 驱动程序配置 SMB 卷。

警告	必须在 SVM 上配置 NFS 和 SMB/CIFS 协议，才能为 ONTAP 本地群集创建 `ontap-nas-economy` SMB 卷。无法配置这些协议中的任何一个都将导致 SMB 卷创建失败。
备注	`autoExportPolicy` 不支持 SMB 卷。

开始之前

在设置 SMB 卷之前，必须具有下列内容。

- 具有 Linux 控制器节点和至少一个运行 Windows Server 2022 的 Windows worker 节点的 Kubernetes 集群。Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。
- 至少有一个包含您的 Active Directory 凭据的 Trident 密码。要生成密钥 `smbcreds`：

```
kubectl create secret generic smbcreds --from-literal username=user
--from-literal password='password'
```

- 配置为 Windows 服务的 CSI 代理。要配置 `csi-proxy`，请参阅"GitHub：CSI Proxy"或"GitHub：适用于 Windows 的 CSI 代理"了解在 Windows 上运行的 Kubernetes 节点。

步骤

1. 对于本地 ONTAP，您可以选择创建 SMB 共享，或者 Trident 可以为您创建一个共享。

备注	Amazon FSx for ONTAP 需要 SMB 共享。

   您可以使用 "Microsoft 管理控制台" 共享文件夹管理单元或使用 ONTAP CLI 以两种方式之一创建 SMB 管理共享。要使用 ONTAP CLI 创建 SMB 共享：

   a. 如有必要，请为共享创建目录路径结构。

      此 `vserver cifs share create` 命令在共享创建期间检查 -path 选项中指定的路径。如果指定的路径不存在，则命令失败。

   b. 创建与指定 SVM 关联的 SMB 共享：

```
vserver cifs share create -vserver vserver_name -share-name
share_name -path path [-share-properties share_properties,...]
[other_attributes] [-comment text]
```

c. 验证是否已创建此共享：

```
vserver cifs share show -share-name share_name
```

备注 有关详细信息，请参见 "创建 SMB 共享"。

2. 创建后端时，必须配置以下内容以指定 SMB 卷。对于所有 FSx for ONTAP 后端配置选项，请参阅 "FSx for ONTAP 配置选项和示例"。

参数	说明	示例
smbShare	可以指定以下选项之一：使用 Microsoft Management Console 或 ONTAP CLI 创建的 SMB 共享的名称；允许 Trident 创建 SMB 共享的名称；或者可以将参数留空以阻止对卷的公共共享访问。此参数对于本地 ONTAP 是可选的。此参数是 Amazon FSx for ONTAP 后端所必需的，不能为空。	smb-share
nasType	*必须设置为 smb。*如果为 null，则默认为 nfs。	smb
securityStyle	新卷的安全样式。对于 **SMB** 卷，必须设置为 **ntfs**` 或 `**mixed**。	ntfs 或 mixed 用于 SMB 卷
unixPermissions	新卷的模式。对于 **SMB** 卷，必须留空。	""

启用安全 **SMB**

从 25.06 版本开始，NetApp Trident 支持使用 `ontap-nas`和 `ontap-nas-economy`后端创建的 SMB 卷的安全配置。启用安全 SMB 后，可以使用访问控制列表 (ACL) 为 Active Directory (AD) 用户和用户组提供对 SMB 共享的受控访问。

需要记住的要点

- 不支持导入 `ontap-nas-economy`卷。
- 仅支持 ontap-nas-economy 卷的只读克隆。
- 如果启用了安全 SMB，Trident 将忽略后端中提到的 SMB 共享。
- 更新 PVC 注释、存储类注释和后端字段不会更新 SMB 共享 ACL。
- 在克隆 PVC 的注释中指定的 SMB 共享 ACL 将优先于源 PVC 中的 ACL。
- 确保在启用安全 SMB 的同时提供有效的 AD 用户。无效用户将不会添加到 ACL。
- 如果在后端、存储类和 PVC 中为相同的 AD 用户提供不同的权限，则权限优先级为：PVC、存储类，然后是后端。
- 安全 SMB 受 `ontap-nas`托管卷导入支持，不适用于非托管卷导入。

步骤

1. 如下面的示例所示，在 TridentBackendConfig 中指定 adAdminUser：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.193.176.x
  svm: svm0
  useREST: true
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

2. 在存储类中添加批注。

   将 `trident.netapp.io/smbShareAdUser` 注释添加到存储类，以始终启用安全的 SMB。为注释 `trident.netapp.io/smbShareAdUser`指定的用户值应与 `smbcreds`密钥中指定的用户名相同。您可以从以下选项中选择一项 `smbShareAdUserPermission`： `full_control`、 `change`或 `read`。默认权限为 `full_control`。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1. 创建 PVC。

以下示例创建了 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/snapshotDirectory: "true"
    trident.netapp.io/smbShareAccessControl: |
      read:
        - tridentADtest
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

ONTAP NAS 配置选项和示例

了解如何在 Trident 安装中创建和使用 ONTAP NAS 驱动程序。本节提供了将后端映射到 StorageClasses 的后端配置示例和详细信息。从 25.10 版本开始，NetApp Trident 支持 "NetApp AFX 存储系统"。NetApp AFX 存储系统与其他基于 ONTAP 的系统（ASA、AFF 和 FAS）在存储层的实现方面有所不同。

备注	仅支持 `ontap-nas`驱动程序（使用 NFS 协议）用于 NetApp AFX 系统；不支持 SMB 协议。

后端配置选项

在 Trident 后端配置中，无需指定您的系统是 NetApp AFX 存储系统。当您选择 `ontap-nas`作为 `storageDriverName`时，Trident 会自动检测 AFX 存储系统。某些后端配置参数不适用于 AFX 存储系统。

下表显示了后端配置选项：

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称 备注：对于 NetApp AFX 系统，仅支持 `ontap-nas`。	`ontap-nas`， `ontap-nas-economy`，或 `ontap-nas-flexgroup`
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + dataLIF

参数	说明	默认
managementLIF	集群或 SVM 管理 LIF 的 IP 地址，也可以指定完全限定域名（FQDN）。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。有关无缝 MetroCluster 切换，请参见 MetroCluster 示例。	“10.0.0.1”，“[2001:1234:abcd::fefe]”
dataLIF	协议 LIF 的 IP 地址。NetApp 建议指定 dataLIF。如果未提供，Trident 将从 SVM 获取 dataLIF。可以指定要用于 NFS 挂载操作的完全限定域名 (FQDN)，允许您创建轮询 DNS 以跨多个 dataLIF 进行负载平衡。可以在初始设置后更改。请参见 。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。*省略 MetroCluster。*请参阅MetroCluster 示例。	指定地址或源自 SVM，如果未指定（不推荐）
svm	要使用的 Storage Virtual Machine *对于 MetroCluster 请省略。*请参阅MetroCluster 示例。	如果指定了 SVM managementLIF，则派生
autoExportPolicy	启用自动导出策略创建和更新 [Boolean]。使用 `autoExportPolicy`和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	false
autoExportCIDRs	启用 `autoExportPolicy`时用于过滤 Kubernetes 节点 IP 的 CIDR 列表。使用 `autoExportPolicy`和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	["0.0.0.0/0", "::/0"]`
labels	要应用于卷的任意 JSON 格式标签集	""
clientCertificate	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端私钥的 Base64 编码值。用于基于证书的身份验证	""
trustedCACertificate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证	""
username	连接到集群/SVM 的用户名。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证 Trident"。	
password	连接到集群/SVM 的密码。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证 Trident"。	
storagePrefix	在 SVM 中配置新卷时使用的前缀。设置后无法更新 备注：当使用 ontap-nas-economy 和 24 个或更多字符的 storagePrefix 时，qtree 将不会嵌入存储前缀，尽管它将位于卷名称中。	“trident”

参数	说明	默认
aggregate	用于配置的聚合（可选；如果设置，则必须分配给 SVM）。对于 ontap-nas-flexgroup 驱动程序，此选项将被忽略。如果未分配，则可以使用任何可用的聚合来配置 FlexGroup 卷。 备注：当聚合在 SVM 中更新时，它会通过轮询 SVM 在 Trident 中自动更新，而无需重新启动 Trident Controller。当您在 Trident 中配置了特定聚合来配置卷时，如果聚合被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将在 Trident 中移至失败状态。您必须将聚合更改为存在于 SVM 上的聚合，或者将其完全删除，以使后端恢复联机。 不要为 **AFF** 存储系统指定。	""
limitAggregateUsage	如果使用率超过此百分比，则配置失败。不适用于 **Amazon FSx for ONTAP**。不要为 **AFF** 存储系统指定。	"（默认情况下不强制执行）
flexgroupAggregateList	用于配置的聚合列表（可选；如果设置，则必须分配给 SVM）。分配给 SVM 的所有聚合都用于配置 FlexGroup 卷。支持 **ontap-nas-flexgroup** 存储驱动程序。 备注：在 SVM 中更新聚合列表时，通过轮询 SVM 自动在 Trident 中更新列表，而无需重新启动 Trident Controller。当您在 Trident 中配置了特定的聚合列表来配置卷时，如果聚合列表被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将移至 Trident 中的失败状态。您必须将聚合列表更改为 SVM 上存在的列表，或者将其完全删除，以使后端恢复联机。	""
limitVolumeSize	如果请求的卷大小高于此值，则设置失败。	"（默认情况下不强制执行）
debugTraceFlags	故障排除时使用的调试标志。例如，{"api":false, "method":true} 除非正在进行故障排除并需要详细的日志转储，否则不要使用 debugTraceFlags。	空
nasType	配置 NFS 或 SMB 卷创建。选项为 nfs、 smb`或 null。设置为 null 默认为 NFS 卷。如果指定，对于 **AFF** 存储系统始终设置为 `**nfs**。	nfs

参数	说明	默认
nfsMountOptions	NFS 挂载选项的逗号分隔列表。Kubernetes 持久卷的挂载选项通常在存储类中指定，但如果存储类中未指定挂载选项，则 Trident 将回退到使用存储后端配置文件中指定的挂载选项。如果存储类或配置文件中未指定挂载选项，Trident 将不会在关联的持久卷上设置任何挂载选项。	""
qtreesPerFlexvol	每个 FlexVol 的最大 Qtrees，必须在 [50, 300] 范围内	"200"
smbShare	可以指定以下选项之一：使用 Microsoft Management Console 或 ONTAP CLI 创建的 SMB 共享的名称；允许 Trident 创建 SMB 共享的名称；或者可以将参数留空以阻止对卷的公共共享访问。此参数对于本地 ONTAP 是可选的。此参数是 Amazon FSx for ONTAP 后端所必需的，不能为空。	smb-share
useREST	使用 ONTAP REST API 的布尔参数。 useREST`设置为 `true`时，Trident 使用 ONTAP REST API 与后端通信；设置为 `false`时，Trident 使用 ONTAPI (ZAPI) 调用与后端通信。此功能需要 ONTAP 9.11.1 及更高版本。此外，所使用的 ONTAP 登录角色必须能够访问 `ontapi`应用程序。这通过预定义的 `vsadmin`和 `cluster-admin`角色来满足。从 Trident 24.06 版本和 ONTAP 9.15.1 或更高版本开始， `useREST`默认设置为 `true；将 useREST`更改为 `false`以使用 ONTAPI (ZAPI) 调用。如果指定，对于 **AFF** 存储系统始终设置为 `**true**。	true 适用于 ONTAP 9.15.1 或更高版本，否则 false。
limitVolumePoolSize	在 ontap-nas-economy 后端使用 Qtrees 时的最大可请求 FlexVol 大小。	" （默认情况下不强制执行）
denyNewVolumePools	限制 `ontap-nas-economy`后端创建新 FlexVol 卷以包含其 Qtree。仅预先存在的 FlexVol 用于配置新的 PV。	
adAdminUser	具有 SMB 共享完全访问权限的 Active Directory 管理员用户或用户组。使用此参数为具有完全控制权限的 SMB 共享提供管理员权限。	

用于配置卷的后端配置选项

您可以使用配置的 defaults 部分中的这些选项来控制默认配置。有关示例，请参阅下面的配置示例。

参数	说明	默认
spaceAllocation	Qtree 的空间分配	"true"
spaceReserve	空间预留模式；"none"（精简）或 "volume"（厚）	"无"
snapshotPolicy	要使用的 Snapshot 策略	"无"

参数	说明	默认
qosPolicy	要为创建的卷分配的 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。ontap-nas-economy 不支持此功能。	""
snapshotReserve	为快照预留的卷百分比	"0" 如果 snapshotPolicy 为 "none"，否则为 "
splitOnClone	创建时从其父级拆分克隆	"false"
encryption	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 false。必须在群集上许可并启用 NVE 才能使用此选项。如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。有关更多信息，请参阅："Trident 如何与 NVE 和 NAE 配合使用"。	"false"
tieringPolicy	要使用"none"的分层策略	
unixPermissions	新卷的模式	NFS 卷为 "777"；SMB 卷为空（不适用）
snapshotDir	控制对 .snapshot 目录的访问	true, false (显式设置)。
exportPolicy	要使用的导出策略	"default"
securityStyle	新卷的安全样式。NFS 支持 `mixed`和 `unix`安全样式。SMB 支持 `mixed`和 `ntfs`安全样式。	NFS 默认值为 unix。SMB 默认值为 ntfs。
nameTemplate	用于创建自定义卷名称的模板。	""

备注	在 Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。您应该使用非共享 QoS 策略组，并确保该策略组单独应用于每个组成部分。共享 QoS 策略组强制执行所有工作负载总吞吐量的上限。

卷配置示例

以下是定义了默认值的示例：

```
---
version: 1
storageDriverName: ontap-nas
backendName: customBackendName
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
labels:
  k8scluster: dev1
  backend: dev1-nasbackend
svm: trident_svm
username: cluster-admin
password: <password>
limitAggregateUsage: 80%
limitVolumeSize: 50Gi
nfsMountOptions: nfsvers=4
debugTraceFlags:
  api: false
  method: true
defaults:
  spaceReserve: volume
  qosPolicy: premium
  exportPolicy: myk8scluster
  snapshotPolicy: default
  snapshotReserve: "10"
```

对于 `ontap-nas`和 `ontap-nas-flexgroups`，Trident 现在使用新的计算来确保 FlexVol 使用 snapshotReserve 百分比和 PVC 正确调整大小。当用户请求 PVC 时，Trident 使用新的计算创建具有更多空间的原始 FlexVol。此计算可确保用户收到他们在 PVC 中请求的可写空间，而不是少于他们请求的空间。在 v21.07 之前，当用户请求 PVC（例如 5 GiB）时，如果 snapshotReserve 为 50%，则仅获得 2.5 GiB 的可写空间。这是因为用户请求的是整个卷，而 `snapshotReserve`是该卷的百分比。对于 `Trident 21.07`，用户要求的是可写空间，`Trident` 将 `snapshotReserve`数字定义为整个卷的百分比。此情况不适用于 `ontap-nas-economy`。请参见以下示例以了解其工作原理：

计算结果如下所示：

```
Total volume size = <PVC requested size> / (1 - (<snapshotReserve
percentage> / 100))
```

对于 snapshotReserve = 50% 和 PVC 请求 = 5 GiB，总体积大小为 5/.5 = 10 GiB，可用大小为 5 GiB，这是用户在 PVC 请求中请求的内容。 `volume show`命令应显示类似于以下示例的结果：

升级 Trident 时，先前安装的现有后端将按上述说明配置卷。对于升级前创建的卷，应调整其卷的大小，以便观察更改。例如，较早的 2 GiB PVC 与 `snapshotReserve=50`导致提供 1 GiB 可写空间的卷。例如，将卷的大小调整为 3 GiB，可在 6 GiB 卷上为应用程序提供 3 GiB 的可写空间。

最小配置示例

以下示例显示了将大多数参数保留为默认值的基本配置。这是定义后端的最简单方法。

备注	如果要在 Trident 上使用 Amazon FSx for NetApp ONTAP，建议为 LIF 指定 DNS 名称而不是 IP 地址。

ONTAP NAS 经济示例

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

ONTAP NAS FlexGroup 示例

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

MetroCluster 示例

您可以配置后端，以避免在 "SVM 复制和恢复" 期间进行切换和切换后手动更新后端定义。

对于无缝切换和切回，使用 `managementLIF` 指定 SVM 并省略 `dataLIF` 和 `svm` 参数。例如：

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 192.168.1.66
username: vsadmin
password: password
```

SMB 卷示例

```
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
nasType: smb
securityStyle: ntfs
unixPermissions: ""
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
```

基于证书的身份验证示例

这是一个最小后端配置示例。 `clientCertificate`、 `clientPrivateKey` 和 `trustedCACertificate`（可选，如果使用受信任的 CA）分别填充在 `backend.json` 中并获取客户端证书、私钥和可信 CA 证书的 base64 编码值。

```
---
version: 1
backendName: DefaultNASBackend
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.15
svm: nfs_svm
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

自动导出策略示例

此示例演示如何指导 Trident 使用动态导出策略自动创建和管理导出策略。这对 `ontap-nas-economy` 和 `ontap-nas-flexgroup` 驱动程序也同样适用。

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-nasbackend
autoExportPolicy: true
autoExportCIDRs:
- 10.0.0.0/24
username: admin
password: password
nfsMountOptions: nfsvers=4
```

IPv6 地址示例

此示例显示了 `managementLIF`使用 IPv6 地址。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas_ipv6_backend
managementLIF: "[5c5d:5edf:8f:7657:bef8:109b:1b41:d491]"
labels:
  k8scluster: test-cluster-east-1a
  backend: test1-ontap-ipv6
svm: nas_ipv6_svm
username: vsadmin
password: password
```

使用 **SMB** 卷的 **Amazon FSx for ONTAP** 示例

对于使用 SMB 卷的 FSx for ONTAP， `smbShare` 参数是必需的。

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
dataLIF: 10.0.0.15
svm: nfs_svm
smbShare: smb-share
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

带有 **nameTemplate** 的后端配置示例

```
---
version: 1
storageDriverName: ontap-nas
backendName: ontap-nas-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

具有虚拟池的后端示例

在下面显示的示例后端定义文件中，为所有存储池设置了特定的默认值，例如 `spaceReserve`为 none、`spaceAllocation`为 false 和 `encryption`为 false。虚拟池在存储部分中定义。

Trident 在"Comments"字段中设置配置标签。Comments 设置在 FlexVol 上用于 `ontap-nas`或 `FlexGroup` 用于 `ontap-nas-flexgroup`。Trident 在配置时将虚拟池上存在的所有标签复制到存储卷。为方便起见，存储管理员可以为每个虚拟池定义标签，并按标签对卷进行分组。

在这些示例中，一些存储池设置了自己的 `spaceReserve`、 `spaceAllocation` 和 `encryption` 值，而一些池覆盖了默认值。

ONTAP NAS 示例

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
svm: svm_nfs
username: admin
password: <password>
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: "false"
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      app: msoffice
      cost: "100"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
      adaptiveQosPolicy: adaptive-premium
  - labels:
      app: slack
      cost: "75"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
```

```
      app: wordpress
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS FlexGroup 示例

```
---
version: 1
storageDriverName: ontap-nas-flexgroup
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: flexgroup_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      protection: gold
      creditpoints: "50000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: gold
      creditpoints: "30000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: silver
      creditpoints: "20000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
```

```
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

ONTAP NAS 经济示例

```
---
version: 1
storageDriverName: ontap-nas-economy
managementLIF: 10.0.0.1
svm: svm_nfs
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  encryption: "false"
labels:
  store: nas_economy_store
region: us_east_1
storage:
  - labels:
      department: finance
      creditpoints: "6000"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      protection: bronze
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: engineering
      creditpoints: "3000"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      department: humanresource
      creditpoints: "2000"
    zone: us_east_1d
    defaults:
```

```
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

将后端映射到 StorageClasses

以下 StorageClass 定义参考了 [具有虚拟池的后端示例]。使用 `parameters.selector` 字段，每个 StorageClass 调用哪些虚拟池可用于托管卷。卷将具有所选虚拟池中定义的方面。

- `protection-gold` StorageClass 将映射到 `ontap-nas-flexgroup`后端中的第一个和第二个虚拟池。这些是唯一提供黄金级保护的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=gold"
  fsType: "ext4"
```

- `protection-not-gold` StorageClass 将映射到 `ontap-nas-flexgroup`后端的第三个和第四个虚拟池。这些是唯一提供黄金以外防护等级的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-not-gold
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection!=gold"
  fsType: "ext4"
```

- `app-mysqldb` StorageClass 将映射到 `ontap-nas`后端的第四个虚拟池。这是唯一为 mysqldb 类型应用程序提供存储池配置的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: app-mysqldb
provisioner: csi.trident.netapp.io
parameters:
  selector: "app=mysqldb"
  fsType: "ext4"
```

- `protection-silver-creditpoints-20k` StorageClass 将映射到 `ontap-nas-flexgroup`后端中的第三个虚拟池。这是唯一提供银级保护和 20000 creditpoints 的池。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: protection-silver-creditpoints-20k
provisioner: csi.trident.netapp.io
parameters:
  selector: "protection=silver; creditpoints=20000"
  fsType: "ext4"
```

- `creditpoints-5k` StorageClass 将映射到 `ontap-nas`后端中的第三个虚拟池和 `ontap-nas-economy`后端中的第二个虚拟池。这些是唯一拥有 5000 个信用点的池产品。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: creditpoints-5k
provisioner: csi.trident.netapp.io
parameters:
  selector: "creditpoints=5000"
  fsType: "ext4"
```

Trident 将决定选择哪个虚拟池，并确保满足存储要求。

初始配置后更新 `dataLIF`

您可以在初始配置后通过运行以下命令更改 dataLIF，以提供具有更新的 dataLIF 的新后端 JSON 文件。

```
tridentctl update backend <backend-name> -f <path-to-backend-json-file-
with-updated-dataLIF>
```

备注	如果 PVC 连接到一个或多个 pod，则必须关闭所有相应的 pod，然后将其重新启动，以使新的 dataLIF 生效。

安全 **SMB** 示例

带有 **ontap-nas** 驱动程序的后端配置

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

使用 **ontap-nas-economy** 驱动程序的后端配置

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas-economy
  managementLIF: 10.0.0.1
  svm: svm2
  nasType: smb
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

使用存储池的后端配置

```yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.0.0.1
  svm: svm0
  useREST: false
  storage:
  - labels:
      app: msoffice
    defaults:
      adAdminUser: tridentADuser
  nasType: smb
  credentials:
    name: backend-tbc-ontap-invest-secret
```

带有 **ontap-nas** 驱动程序的存储类示例

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADtest
parameters:
  backendType: ontap-nas
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

备注	请确保添加 `annotations`以启用安全的 SMB。如果没有注释，无论后端或 PVC 中设置的配置如何，Secure SMB 都无法正常工作。

具有 **ontap-nas-economy** 驱动程序的存储类示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-smb-sc
  annotations:
    trident.netapp.io/smbShareAdUserPermission: change
    trident.netapp.io/smbShareAdUser: tridentADuser3
parameters:
  backendType: ontap-nas-economy
  csi.storage.k8s.io/node-stage-secret-name: smbcreds
  csi.storage.k8s.io/node-stage-secret-namespace: trident
  trident.netapp.io/nasType: smb
provisioner: csi.trident.netapp.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

单个 **AD** 用户的 **PVC** 示例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      change:
        - tridentADtest
      read:
        - tridentADuser
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

具有多个 **AD** 用户的 **PVC** 示例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-test-pvc
  annotations:
    trident.netapp.io/smbShareAccessControl: |
      full_control:
        - tridentTestuser
        - tridentuser
        - tridentTestuser1
        - tridentuser1
      change:
        - tridentADuser
        - tridentADuser1
        - tridentADuser4
        - tridentTestuser2
      read:
        - tridentTestuser2
        - tridentTestuser3
        - tridentADuser2
        - tridentADuser3
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Amazon FSx for NetApp ONTAP

将 **Trident** 与 **Amazon FSx for NetApp ONTAP** 结合使用

"Amazon FSx for NetApp ONTAP" 是一项完全托管的 AWS 服务，运行由 NetApp ONTAP 存储操作系统支持的文件系统。它提供 ONTAP 功能、性能和管理，具有 AWS 的可扩展性和操作简便性。文件系统是 Amazon FSx 中的主要资源，类似于本地 ONTAP 集群。每个文件系统包含一个或多个存储虚拟机 (SVM)，每个 SVM 包含一个或多个存储文件和目录的卷。这种集成使在 Amazon Elastic Kubernetes Service (EKS) 中运行的 Kubernetes 集群能够为块和文件工作负载配置 ONTAP 支持的持久卷。

要求

除了"Trident 要求"，要将 FSx for ONTAP 与 Trident 集成，您还需要：

- 已安装 `kubectl`的现有 Amazon EKS 集群或自我管理的 Kubernetes 集群。
- 可从群集的工作节点访问的现有 Amazon FSx for NetApp ONTAP 文件系统和 Storage Virtual Machine

(SVM)。

- 已为 "NFS 或 iSCSI" 准备好的工作节点。

> 备注 请确保遵循 Amazon Linux 和 Ubuntu "Amazon Machine Images" (AMI) 所需的节点准备步骤，具体取决于您的 EKS AMI 类型。

注意事项

- SMB 卷：
  - 仅使用 `ontap-nas` 驱动程序支持 SMB 卷。
  - Trident EKS 加载项不支持 SMB 卷。
  - Trident 仅支持将 SMB 卷挂载到在 Windows 节点上运行的 Pod。有关详细信息，请参见 "准备配置 SMB 卷"。
- 在 Trident 24.02 之前，在启用了自动备份的 Amazon FSx 文件系统上创建的卷无法被 Trident 删除。要防止 Trident 24.02 或更高版本中的此问题，请在 AWS FSx for ONTAP 的后端配置文件中指定 `fsxFilesystemID`、AWS `apiRegion`、AWS `apikey` 和 AWS `secretKey`。

> 备注 如果要为 Trident 指定 IAM 角色，则可以省略为 Trident 明确指定 `apiRegion`、 `apiKey`和 `secretKey`字段。有关详细信息，请参阅 "FSx for ONTAP 配置选项和示例"。

同时使用 **Trident SAN/iSCSI** 和 **EBS-CSI** 驱动程序

如果您计划将 ontap-san 驱动程序（例如 iSCSI）与 AWS（EKS、ROSA、EC2 或任何其他实例）一起使用，则节点上所需的多路径配置可能会与 Amazon Elastic Block Store (EBS) CSI 驱动程序冲突。为了确保多路径功能不会干扰同一节点上的 EBS 磁盘，您需要在多路径设置中排除 EBS。此示例显示了一个 `multipath.conf`文件，其中包含所需的 Trident 设置，同时将 EBS 磁盘排除在多路径之外：

```
defaults {
    find_multipaths no
}
blacklist {
    device {
        vendor "NVME"
        product "Amazon Elastic Block Store"
    }
}
```

身份验证

Trident 提供了两种身份验证模式。

- 基于凭据（推荐）：在 AWS Secrets Manager 中安全地存储凭据。您可以使用文件系统的 `fsxadmin` 用户或为 SVM 配置的 `vsadmin` 用户。

警告	Trident 希望以 `vsadmin` SVM 用户身份或以具有相同角色的不同名称的用户身份运行。Amazon FSx for NetApp ONTAP 有一个 `fsxadmin` 用户，它是 ONTAP `admin` 集群用户的有限替代品。我们强烈建议将 `vsadmin` 与 Trident 一起使用。

- 基于证书：Trident 将使用 SVM 上安装的证书与 FSx 文件系统上的 SVM 进行通信。

有关启用身份验证的详细信息，请参阅您的驱动程序类型的身份验证：

- "ONTAP NAS 身份验证"
- "ONTAP SAN 身份验证"

已测试的 **Amazon Machine Images (AMIs)**

EKS 集群支持各种操作系统，但 AWS 已针对容器和 EKS 优化了某些 Amazon Machine Images (AMI)。以下 AMI 已通过 NetApp Trident 25.02 测试。

AMI	NAS	NAS-经济型	iSCSI	iSCSI-经济性
AL2023_x86_64_STANDARD	是	是	是	是
AL2_x86_64	是	是	是*	是*
BOTTLEROCKET_x86_64	是**	是	不适用	不适用
AL2023_ARM_64_STANDARD	是	是	是	是
AL2_ARM_64	是	是	是*	是*
BOTTLEROCKET_ARM_64	是**	是	不适用	不适用

- * 如果不重新启动节点，则无法删除 PV
- ** 不适用于 Trident 版本 25.02 的 NFSv3。

备注	如果您所需的 AMI 未在此处列出，这并不意味着它不受支持；它只是意味着它尚未经过测试。此列表可作为已知有效的 AMI 的指南。

执行测试使用：

- EKS 版本：1.32
- 安装方法：Helm 25.06 和作为 AWS 插件 25.06
- 对于 NAS，已测试 NFSv3 和 NFSv4.1。
- 对于 SAN，仅测试了 iSCSI，而未测试 NVMe-oF。

已执行的测试：

- 创建：Storage Class、pvc、pod
- 删除：pod、pvc（常规、qtree/lun – economy、带 AWS 备份的 NAS）

查找更多信息

- "Amazon FSx for NetApp ONTAP 文档"
- "关于 Amazon FSx for NetApp ONTAP 的博客文章"

创建 **IAM** 角色和 **AWS Secret**

您可以通过作为 AWS IAM 角色进行身份验证，而不是提供明确的 AWS 凭据，来配置 Kubernetes Pod 以访问 AWS 资源。

备注	要使用 AWS IAM 角色进行身份验证，必须使用 EKS 部署 Kubernetes 集群。

创建 **AWS Secrets Manager** 密钥

由于 Trident 将针对 FSx vserver 发布 API 来为您管理存储，因此需要凭据才能执行此操作。传递这些凭据的安全方法是通过 AWS Secrets Manager 密钥。因此，如果您还没有 AWS Secrets Manager 密钥，则需要创建包含 vsadmin 帐户凭据的 AWS Secrets Manager 密钥。

此示例创建 AWS Secrets Manager 密码以存储 Trident CSI 凭据：

```
aws secretsmanager create-secret --name trident-secret --description
"Trident CSI credentials"\
  --secret-string
"{\"username\":\"vsadmin\",\"password\":\"<svmpassword>\"}"
```

创建 **IAM** 策略

Trident 还需要 AWS 权限才能正常运行。因此，您需要创建一个策略，为 Trident 提供所需的权限。

以下示例使用 AWS CLI 创建 IAM 策略：

```
aws iam create-policy --policy-name AmazonFSxNCSIDriverPolicy --policy
-document file://policy.json
  --description "This policy grants access to Trident CSI to FSxN and
Secrets manager"
```

策略 **JSON** 示例：

```
{
  "Statement": [
    {
      "Action": [
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes",
        "fsx:CreateVolume",
        "fsx:RestoreVolumeFromSnapshot",
        "fsx:DescribeStorageVirtualMachines",
        "fsx:UntagResource",
        "fsx:UpdateVolume",
        "fsx:TagResource",
        "fsx:DeleteVolume"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Resource": "arn:aws:secretsmanager:<aws-region>:<aws-account-
id>:secret:<aws-secret-manager-name>*"
    }
  ],
  "Version": "2012-10-17"
}
```

为服务帐户关联 (IRSA) 创建 Pod Identity 或 IAM 角色

您可以将 Kubernetes 服务帐户配置为具有 EKS Pod Identity 或 IAM role for Service account association (IRSA) 的 AWS Identity and Access Management (IAM) 角色。然后，配置为使用服务帐户的任何 Pod 都可以访问角色有权访问的任何 AWS 服务。

Pod 身份

Amazon EKS Pod Identity 关联提供了管理应用程序凭据的功能，类似于 Amazon EC2 实例配置文件向 Amazon EC2 实例提供凭据的方式。

在 **EKS** 集群上安装 **Pod Identity**：

您可以通过 AWS 控制台或使用以下 AWS CLI 命令创建 Pod 标识：

```
aws eks create-addon --cluster-name <EKS_CLUSTER_NAME> --addon-name
eks-pod-identity-agent
```

有关详细信息，请参阅 "设置 Amazon EKS Pod Identity Agent"。

创建 **trust-relationship.json**：

创建 trust-relationship.json 以启用 EKS Service Principal 承担 Pod Identity 的此角色。然后使用此信任策略创建角色：

```
aws iam create-role \
  --role-name fsxn-csi-role --assume-role-policy-document file://trust-
relationship.json \
  --description "fsxn csi pod identity role"
```

trust-relationship.json 文件：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

将角色策略附加到 **IAM** 角色：

将上一步中的角色策略附加到已创建的 IAM 角色：

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:111122223333:policy/fsxn-csi-policy \
  --role-name fsxn-csi-role
```

创建 **pod** 标识关联：

在 IAM 角色和 Trident 服务帐户（trident-controller）之间创建 pod 身份关联

```
aws eks create-pod-identity-association \
  --cluster-name <EKS_CLUSTER_NAME> \
  --role-arn arn:aws:iam::111122223333:role/fsxn-csi-role \
  --namespace trident --service-account trident-controller
```

服务帐户关联 **(IRSA)** 的 **IAM** 角色

使用 **AWS CLI**：

```
aws iam create-role --role-name AmazonEKS_FSxN_CSI_DriverRole \
  --assume-role-policy-document file://trust-relationship.json
```

trust-relationship.json 文件：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<account_id>:oidc-
provider/<oidc_provider>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<oidc_provider>:aud": "sts.amazonaws.com",
          "<oidc_provider>:sub":
"system:serviceaccount:trident:trident-controller"
        }
      }
    }
  ]
}
```

更新 `trust-relationship.json` 文件中的以下值：

- **<account_id>** - 您的 AWS 账户 ID
- **<oidc_provider>** - EKS 集群的 OIDC。您可以通过运行以下命令获取 oidc_provider：

```
aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer"\
  --output text | sed -e "s/^https:\/\///"
```

将 **IAM** 角色与 **IAM** 策略绑定：

创建角色后，使用此命令将策略（在上述步骤中创建的）附加到角色：

```
aws iam attach-role-policy --role-name my-role --policy-arn <IAM policy
ARN>
```

验证 **OICD** 提供者是否关联：

验证您的 OIDC 提供程序是否与您的集群关联。您可以使用以下命令验证它：

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

如果输出为空，请使用以下命令将 IAM OIDC 关联到您的群集：

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name
--approve
```

如果使用 **eksctl**，请使用以下示例在 EKS 中为服务帐户创建 IAM 角色：

```
eksctl create iamserviceaccount --name trident-controller --namespace
trident \
  --cluster <my-cluster> --role-name AmazonEKS_FSxN_CSI_DriverRole
--role-only \
  --attach-policy-arn <IAM-Policy ARN> --approve
```

安装 **Trident**

Trident 简化了 Kubernetes 中 Amazon FSx for NetApp ONTAP 存储管理，使您的开发人员和管理员能够专注于应用程序部署。您可以使用以下方法之一安装 Trident：

- Helm

- EKS 附加项

如果要使用快照功能，请安装 CSI 快照控制器加载项。有关详细信息，请参见 "为 CSI 卷启用快照功能"。

通过 **helm** 安装 **Trident**

Pod 身份

1. 添加 Trident Helm 存储库：

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

2. 请使用以下示例安装 Trident：

```
helm install trident-operator netapp-trident/trident-operator --version 100.2502.1 --namespace trident --create-namespace
```

您可以使用 `helm list` 命令查看安装详细信息，例如名称、命名空间、图表、状态、应用版本和修订版号。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2502.0    25.02.0
```

服务账户关联 (IRSA)

1. 添加 Trident Helm 存储库：

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

2. 设置 **cloud provider** 和 **cloud identity** 的值：

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1  \
--set cloudProvider="AWS" \
--set cloudIdentity="'eks.amazonaws.com/role-arn:
arn:aws:iam::<accountID>:role/<AmazonEKS_FSxN_CSI_DriverRole>'" \
--namespace trident \
--create-namespace
```

您可以使用 `helm list` 命令查看安装详细信息，例如名称、命名空间、图表、状态、应用版本和修订版号。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

trident-operator    trident     1           2024-10-14
14:31:22.463122 +0300 IDT    deployed     trident-operator-
100.2510.0    25.10.0
```

备注

如果您计划使用 iSCSI，请确保在您的客户端计算机上启用了 iSCSI。如果您使用的是 AL2023 Worker 节点操作系统，则可以通过在 helm 安装中添加 node prep 参数来自动安装 iSCSI 客户端：

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2502.1 --namespace trident --create-namespace --
set nodePrep={iscsi}
```

通过 **EKS** 附加组件安装 **Trident**

Trident EKS 加载项包含最新的安全补丁、错误修复，并经过 AWS 验证可与 Amazon EKS 配合使用。EKS 加载项使您能够始终如一地确保 Amazon EKS 集群的安全和稳定，并减少安装、配置和更新加载项所需的工作量。

前提条件

在为 AWS EKS 配置 Trident 加载项之前，请确保具备以下条件：

- 具有附加订阅的 Amazon EKS 集群帐户
- AWS 对 AWS marketplace 的权限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI 类型：Amazon Linux 2 (AL2_x86_64) 或 Amazon Linux 2 Arm(AL2_ARM_64)
- 节点类型：AMD 或 ARM
- 现有 Amazon FSx for NetApp ONTAP 文件系统

为 **AWS** 启用 **Trident** 加载项

管理控制台

1. 在 https://console.aws.amazon.com/eks/home#/clusters 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择 **Clusters**。
3. 选择要为其配置 NetApp Trident CSI 加载项的群集的名称。
4. 选择 **Add-ons**，然后选择 **Get more add-ons**。
5. 按照以下步骤选择附加组件：
   a. 向下滚动到 **AWS Marketplace add-ons** 部分，然后在搜索框中键入 **"Trident"**。
   b. 选中 Trident by NetApp 框右上角的复选框。
   c. 选择 **Next**。
6. 在 **Configure selected add-ons** 设置页面上，执行以下操作：

   备注 | 如果您使用的是 **Pod Identity** 关联，请跳过这些步骤。

   a. 选择您想要使用的 **Version**。
   b. 如果使用 IRSA 身份验证，请确保在可选配置设置中设置可用的配置值：
      - 选择您想要使用的 **Version**。
      - 按照*附加组件配置模式*，将*配置值*部分的*configurationValues*参数设置为您在上一步骤中创建的角色 arn（值应采用以下格式）：

```
{

  "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
  "cloudProvider": "AWS"

}
```

+
如果为冲突解决方法选择覆盖，则可以使用 Amazon EKS 加载项设置覆盖现有加载项的一个或多个设置。如果未启用此选项，并且与现有设置存在冲突，则操作将失败。您可以使用生成的错误消息来排除冲突。在选择此选项之前，请确保 Amazon EKS 加载项不会管理您需要自我管理的设置。

7. 选择 **Next**。
8. 在 **Review and add** 页面上，选择 **Create**。

   加载项安装完成后，您将看到已安装的加载项。

AWS CLI

1.创建 **`add-on.json`**文件：

对于 **Pod Identity**，请使用以下格式：

备注 使用

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
}
```

对于 **IRSA** 身份验证，请使用以下格式：

```
{
  "clusterName": "<eks-cluster>",
  "addonName": "netapp_trident-operator",
  "addonVersion": "v25.6.0-eksbuild.1",
  "serviceAccountRoleArn": "<role ARN>",
  "configurationValues": {
    "cloudIdentity": "'eks.amazonaws.com/role-arn: <role ARN>'",
    "cloudProvider": "AWS"
  }
}
```

备注 将 `<role ARN>` 替换为上一步中创建的角色的 ARN。

2.安装 **Trident EKS** 附加组件。

```
aws eks create-addon --cli-input-json file://add-on.json
```

eksctl

以下示例命令安装 Trident EKS 加载项：

```
eksctl create addon --name netapp_trident-operator --cluster
<cluster_name> --force
```

更新 **Trident EKS** 附加组件

管理控制台

1. 打开 Amazon EKS 控制台 https://console.aws.amazon.com/eks/home#/clusters。
2. 在左侧导航窗格中，选择 **Clusters**。
3. 选择要为其更新 NetApp Trident CSI 加载项的群集的名称。
4. 选择 **Add-ons** 选项卡。
5. 选择 **Trident by NetApp** ，然后选择 **Edit** 。
6. 在 **Configure Trident by NetApp** 页面上，执行以下操作：
   a. 选择您想要使用的 **Version**。
   b. 展开 **Optional configuration settings** 并根据需要进行修改。
   c. 选择 **Save changes**。

AWS CLI

以下示例更新 EKS 加载项：

```
aws eks update-addon --cluster-name <eks_cluster_name> --addon-name
netapp_trident-operator --addon-version v25.6.0-eksbuild.1 \
  --service-account-role-arn <role-ARN> --resolve-conflict preserve \
  --configuration-values "{\"cloudIdentity\":
\"'eks.amazonaws.com/role-arn: <role ARN>'\"}"
```

eksctl

- 检查 FSxN Trident CSI 附加组件的当前版本。将 `my-cluster` 替换为您的集群名称。

```
eksctl get addon --name netapp_trident-operator --cluster my-cluster
```

输出示例：

```
NAME                          VERSION              STATUS    ISSUES
IAMROLE    UPDATE AVAILABLE    CONFIGURATION VALUES
netapp_trident-operator    v25.6.0-eksbuild.1    ACTIVE    0
{"cloudIdentity":"'eks.amazonaws.com/role-arn:
arn:aws:iam::139763910815:role/AmazonEKS_FSXN_CSI_DriverRole'"}
```

- 将加载项更新到上一步输出中的 UPDATE AVAILABLE 下返回的版本。

```
eksctl update addon --name netapp_trident-operator --version
v25.6.0-eksbuild.1 --cluster my-cluster --force
```

如果您删除该 `--force` 选项，并且任何 Amazon EKS 加载项设置与现有设置冲突，则更新 Amazon EKS 加载项失败；您会收到一条错误消息，以帮助您解决冲突。在指定此选项之前，请确保 Amazon EKS 加载项不管理您需要管理的设置，因为这些设置会被此选项覆盖。有关此设置的其他选项的详细信息，请参见 "插件"。有关 Amazon EKS Kubernetes 字段管理的详细信息，请参见 "Kubernetes 字段管理"。

卸载/删除 **Trident EKS** 插件

您有两种删除 Amazon EKS 加载项的选项：

- 保留集群上的附加软件 – 此选项可删除任何设置的 Amazon EKS 管理。它还删除了 Amazon EKS 通知您更新并在您启动更新后自动更新 Amazon EKS 加载项的功能。但是，它会保留集群上的加载项软件。此选项使加载项成为自我管理的安装，而不是 Amazon EKS 加载项。使用此选项，附加组件不会停机。保留命令中的 `--preserve` 选项以保留加载项。
- 从集群中完全移除附加软件 – NetApp 建议仅当您的集群上没有依赖该附加组件的资源时，才从集群中移除 Amazon EKS 附加组件。从 `--preserve`命令中移除 `delete`选项以移除该附加组件。

备注	如果附加组件具有关联的 IAM 帐户，则不会删除该 IAM 帐户。

管理控制台

1. 在 https://console.aws.amazon.com/eks/home#/clusters 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择 **Clusters**。
3. 选择要删除其 NetApp Trident CSI 加载项的群集的名称。
4. 选择 **Add-ons** 选项卡，然后选择 **Trident by NetApp**。
5. 选择 **Remove**。
6. 在 删除 **netapp_trident-operator** 确认 对话框中，执行以下操作：
   a. 如果希望 Amazon EKS 停止管理加载项的设置，请选择 **Preserve on cluster**。如果要将加载项软件保留在集群上，以便可以自行管理加载项的所有设置，请执行此操作。
   b. 输入 **netapp_trident-operator**。
   c. 选择 **Remove**。

AWS CLI

将 `my-cluster` 替换为群集的名称，然后运行以下命令。

```
aws eks delete-addon --cluster-name my-cluster --addon-name
netapp_trident-operator --preserve
```

eksctl

以下命令卸载 Trident EKS 加载项：

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass 对象"^] 标识置备程序并指示置备程序如何置备卷。本节将向您展示如何配置将 Trident 指定为置备程序的 Kubernetes StorageClass 对象。`

创建 **StorageClass** 对象

当您为 FSx for ONTAP 创建 StorageClass 时，Trident 将自动创建后端配置。

备注	如果要手动配置存储后端，请参阅 [create-a-kubernetes-storageclass-without-automatic-backend-configuration] 部分了解如何分别创建 Trident 后端和存储类。

指定所需 **StorageClass** 参数

创建 StorageClass 时需要定义以下三个参数：

参数	必填项	类型	说明
`fsxFilesystemID`	是	string	FSx for NetApp ONTAP 文件系统 ID
`storageDriverName`	是	string	Trident 存储驱动程序（例如， `ontap-nas` 或 `ontap-san`）
`credentialsName`	是	string	包含 FSx for ONTAP 凭据的 Kubernetes Secret 的名称

指定可选参数

您可以通过 StorageClass 传递可选的后端参数。在 StorageClass `parameters` 部分中将所有可选值定义为字符串。有关后端参数的完整列表，请参阅："FSx for NetApp ONTAP 后端配置"。

示例 **StorageClass** 配置文件。

以下示例显示了触发自动后端配置的 StorageClass。

YAML

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-fsx-demo
  annotations:
    description: "Demo StorageClass for FSx for NetApp ONTAP"
provisioner: csi.trident.netapp.io
parameters:
  fsxFilesystemID: "fs-0abc123"
  storageDriverName: "ontap-nas"
  credentialsName: trident-fsx-credentials
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

创建 **StorageClass**

创建配置文件后，请运行以下命令来创建存储类。

```
kubectl create -f storage-class-ontapnas.yaml
```

现在，您应该在 Kubernetes 和 Trident 中都看到 **basic-csi** 存储类，并且 Trident 应该已经发现了后端的池。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

应用 StorageClass 后，Trident 会自动创建后端。然后，您可以创建引用此 StorageClass 的 PersistentVolumeClaims。

验证后端配置状态

Trident 在 StorageClass 注释中记录后端创建的结果。

标注	说明
`trident.netapp.io/configuratorStatus`	配置结果 (`Success` 或 `Failure`)
`trident.netapp.io/configuratorMessage`	详细状态或错误消息
`trident.netapp.io/configuratorName`	内部配置器资源的名称
`trident.netapp.io/managed`	表示 StorageClass 由 Trident 管理
`trident.netapp.io/additionalStoragePools`	为此后端创建的存储池

要验证状态，请运行：

```
kubectl get storageclass ontap-fsx-demo -o yaml
```

确认 `trident.netapp.io/configuratorStatus`已设置为 `Success。如果值为 `Failure`，请检查 `trident.netapp.io/configuratorMessage`以了解错误。

添加其他 **FSxN** 文件系统

如果在继续使用相同的 StorageClass 的同时需要额外的存储容量，请添加其他 FSxN 文件系统 ID。

编辑 StorageClass 并添加以下注释：

```
metadata:
  annotations:
    trident.netapp.io/additionalFsxNFileSystemID: '["fs-
xxxxxxxxxxxxxxxxx"]'
```

应用更改后，Trident 更新后端配置并更新 StorageClass 注释。

操作注意事项和限制

- 删除具有自动后端配置的 StorageClass 通常会删除关联的 Trident 后端。这可能会中断存储连接并中断正在运行的工作负载。在删除托管的 StorageClass 之前，请验证影响。
- 只有适用于 NetApp ONTAP 的 AWS FSx 才支持自动后端配置。

创建没有自动后端配置的 **Kubernetes StorageClass**

如果要单独创建 Trident 后端和 StorageClass，请按照以下步骤操作。

了解自动后端配置的工作原理

Trident 从 StorageClass 定义中导出后端配置。当您应用 StorageClass 时，Trident 验证所需的参数，创建后端，并使用状态注释 StorageClass。

Trident 只创建一次 VolumeSnapshotClass。Trident 会为后续的 StorageClasses 重复使用同一个 VolumeSnapshotClass。

创建 **Trident** 后端

要创建 Trident 后端，需要创建 JSON 或 YAML 格式的配置文件。文件需要指定所需的存储类型（NAS 或 SAN ）、文件系统和获取该文件的 SVM 以及使用该文件进行身份验证的方式。以下示例演示如何定义基于 NAS 的存储并使用 AWS 密钥将凭据存储到要使用的 SVM：

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
  credentials:
    name: "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name"
    type: awsarn
```

JSON

```
{
  "apiVersion": "trident.netapp.io/v1",
  "kind": "TridentBackendConfig",
  "metadata": {
    "name": "backend-tbc-ontap-nas"
    "namespace": "trident"
  },
  "spec": {
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx"
    },
    "managementLIF": null,
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name",
      "type": "awsarn"
    }
  }
}
```

FSx for ONTAP 驱动程序详细信息

您可以使用以下驱动程序将 Trident 与 Amazon FSx for NetApp ONTAP 集成：

驱动程序名称	说明
`ontap-san`	每个配置的 PV 都是其自己的 Amazon FSx for NetApp ONTAP 卷中的一个 LUN。推荐用于块存储。
`ontap-nas`	配置的每个 PV 都是完整的 Amazon FSx for NetApp ONTAP 卷。推荐用于 NFS 和 SMB。
`ontap-san-economy`	每个配置的 PV 都是一个 LUN，每个 Amazon FSx for NetApp ONTAP 卷具有可配置数量的 LUN。
`ontap-nas-economy`	配置的每个 PV 都是一个 qtree，每个 Amazon FSx for NetApp ONTAP 卷具有可配置数量的 qtree。
`ontap-nas-flexgroup`	每个已配置的 PV 都是一个完整的 Amazon FSx for NetApp ONTAP FlexGroup 卷。

有关驱动程序的详细信息，请参阅 "NAS 驱动程序" 和 "SAN 驱动程序"。

创建后端

创建配置文件后，运行以下命令以创建和验证 Trident 后端配置 (TBC)：

- 从 yaml 文件创建 Trident 后端配置 (TBC) 并运行以下命令：

```
kubectl create -f backendconfig.yaml -n trident
```

```
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-nas created
```

- 验证已成功创建 Trident 后端配置 (TBC)：

```
Kubectl get tbc -n trident
```

```
NAME                            BACKEND NAME          BACKEND UUID
PHASE   STATUS

backend-tbc-ontap-nas           tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9   Bound   Success
```

有关其他配置选项的详细信息，请参见以下 [Backend-advanced-configuration-and-examples] 部分。

配置存储类*无*自动后端配置

以下是与 Trident 和 FSx for ONTAP 配合使用的存储类配置示例。

NFS 存储类

您可以使用此示例为使用 NFS 的卷设置 StorageClass（有关属性的完整列表，请参阅下面的 Trident 属性部分）：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
```

适用于 **iSCSI** 的 **Storage Class**

使用此示例为使用 iSCSI 的卷设置 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  provisioningType: "thin"
  snapshots: "true"
```

使用 **NFSv3** 和 **AWS Bottlerocket** 的存储类

要在 AWS Bottlerocket 上配置 NFSv3 卷，请将所需的 `mountOptions`添加到存储类：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
mountOptions:
  - nfsvers=3
  - nolock
```

Trident StorageClass 属性

这些参数确定应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	类型	值	提供	请求	支持方
媒体[1]	string	hdd、hybrid、ssd	池包含此类型的介质；混合意味着两者兼有	指定媒体类型	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，solidfire-san
provisioningType	string	薄、厚	池支持此配置方法	已指定配置方法	thick：所有 ONTAP；thin：所有 ONTAP 和 solidfire-san
backendType	string	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，solidfire-san，azure-netapp-files，ontap-san-economy	池属于此类型的后端	指定后端	所有驱动程序
snapshots	布尔值	true，false	池支持具有快照的卷	已启用快照的卷	ontap-nas、ontap-san、solidfire-san
个克隆	布尔值	true，false	池支持克隆卷	已启用克隆的卷	ontap-nas、ontap-san、solidfire-san

属性	类型	值	提供	请求	支持方
加密	布尔值	true，false	池支持加密卷	已启用加密的卷	ontap-nas，ontap-nas-economy，ontap-nas-flexgroups，ontap-san
IOPS	int	正整数	池能够保证此范围内的 IOPS	卷保证这些 IOPS	solidfire-san

[1]：ONTAP Select 或 FSx for ONTAP 系统不支持

请参阅"Kubernetes 和 Trident 对象"，了解存储类如何与 `PersistentVolumeClaim`交互，以及用于控制 Trident 配置卷的参数详情。

创建存储类

配置 StorageClass 后，您可以在 Kubernetes 中创建它。

步骤

1. 这是一个 Kubernetes 对象，因此请使用 `kubectl` 在 Kubernetes 中创建它。

```
kubectl create -f storage-class-ontapnas.yaml
```

2. 现在，您应该在 Kubernetes 和 Trident 中都看到 **basic-csi** 存储类，并且 Trident 应该已经发现了后端的池。

```
kubectl get sc basic-csi
```

```
NAME          PROVISIONER             AGE
basic-csi     csi.trident.netapp.io   15h
```

配置 SMB 卷

您可以使用 `ontap-nas` 驱动程序配置 SMB 卷。但是，要执行此操作，您必须完成以下步骤："准备配置 SMB 卷"。

后端高级配置和示例

有关后端配置选项，请参见下表：

参数	说明	示例
`version`		始终为 1

参数	说明	示例
`storageDriverName`	存储驱动程序的名称	`ontap-nas`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `ontap-san`, `ontap-san-economy`
`backendName`	自定义名称或存储后端	驱动程序名称 + "_" + dataLIF
`managementLIF`	集群或 SVM 管理 LIF 的 IP 地址，也可以指定完全限定域名（FQDN）。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须以方括号定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。如果你在 `fsxFilesystemID`下的 `aws` 字段中提供了 `managementLIF`，则无需再提供 `managementLIF`，因为 Trident 会从 AWS 检索 SVM 信息。因此，你必须为 SVM 下的用户（例如：vsadmin）提供凭据，并且该用户必须具有 `vsadmin`角色。	“10.0.0.1”，“[2001:1234:abcd::fefe]”
`dataLIF`	协议 LIF 的 IP 地址。**ONTAP NAS** 驱动程序：NetApp 建议指定 dataLIF。如果未提供，Trident 将从 SVM 获取 dataLIF。可以指定要用于 NFS 挂载操作的完全限定域名 (FQDN)，允许您创建轮询 DNS 以跨多个 dataLIF 进行负载平衡。可以在初始设置后更改。**ONTAP SAN** 驱动程序：不要为 iSCSI 指定。Trident 使用 ONTAP Selective LUN Map 来发现建立多路径会话所需的 iSCSI LIF。如果明确定义了 dataLIF，则会生成警告。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须以方括号定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。	
`autoExportPolicy`	启用自动导出策略创建和更新 [Boolean]。使用 `autoExportPolicy` 和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	`false`
`autoExportCIDRs`	启用 `autoExportPolicy`时用于过滤 Kubernetes 节点 IP 的 CIDR 列表。使用 `autoExportPolicy`和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	"["0.0.0.0/0", "::/0"]"
`labels`	要应用于卷的任意 JSON 格式标签集	""

参数	说明	示例
`clientCertificate`	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
`clientPrivateKey`	客户端私钥的 Base64 编码值。用于基于证书的身份验证	""
`trustedCACertificate`	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""
`username`	连接到集群或 SVM 的用户名。用于基于凭据的身份验证。例如，vsadmin。	
`password`	连接到集群或 SVM 的密码。用于基于凭据的身份验证。	
`svm`	要使用的 Storage Virtual Machine	如果指定了 SVM managementLIF，则派生。
`storagePrefix`	在 SVM 中配置新卷时使用的前缀。创建后无法修改。要更新此参数，您需要创建一个新的后端。	`trident`
`limitAggregateUsage`	*请勿为 Amazon FSx for NetApp ONTAP 指定。*提供的 `fsxadmin` 和 `vsadmin`不包含检索聚合使用情况并使用 Trident 限制它所需的权限。	请勿使用。
`limitVolumeSize`	如果请求的卷大小高于此值，则设置失败。还限制其为 qtree 和 LUN 管理的卷的最大大小，并且该 `qtreesPerFlexvol`选项允许自定义每个 FlexVol 卷的最大 qtree 数	" （默认情况下不强制执行）
`lunsPerFlexvol`	每个 FlexVol volume 的最大 LUN 数必须在 [50, 200] 范围内。仅限 SAN。	“100”
`debugTraceFlags`	故障排除时使用的调试标志。例如，{"api":false, "method":true} 除非正在进行故障排除并需要详细的日志转储，否则不要使用 `debugTraceFlags`。	空
`nfsMountOptions`	NFS 挂载选项的逗号分隔列表。Kubernetes 持久卷的挂载选项通常在存储类中指定，但如果存储类中未指定挂载选项，则 Trident 将回退到使用存储后端配置文件中指定的挂载选项。如果存储类或配置文件中未指定挂载选项，Trident 将不会在关联的持久卷上设置任何挂载选项。	""

参数	说明	示例
nasType	配置 NFS 或 SMB 卷创建。选项为 nfs、 smb`或 null。*对于 SMB 卷，必须设置为 `smb。*设置为 null 默认为 NFS 卷。	nfs
qtreesPerFlexvol	每个 FlexVol 卷的最大 Qtree 数，必须在 [50, 300] 范围内	"200"
smbShare	您可以指定以下选项之一：使用 Microsoft Management Console 或 ONTAP CLI 创建的 SMB 共享的名称，或允许 Trident 创建 SMB 共享的名称。Amazon FSx for ONTAP 后端需要此参数。	smb-share
useREST	使用 ONTAP REST API 的布尔参数。当设置为 `true`时，Trident 将使用 ONTAP REST API 与后端进行通信。此功能需要 ONTAP 9.11.1 及更高版本。此外，所使用的 ONTAP 登录角色必须能够访问 `ontap`应用程序。这通过预定义的 `vsadmin`和 `cluster-admin`角色来满足。	false
aws	您可以在 AWS FSx for ONTAP 的配置文件中指定以下内容：- fsxFilesystemID：指定 AWS FSx 文件系统的 ID。- apiRegion：AWS API 区域名称。- apikey：AWS API 密钥。- secretKey：AWS 密钥。	"" "" ""
credentials	指定要存储在 AWS Secrets Manager 中的 FSx SVM 凭据。- name：机密的 Amazon 资源名称 (ARN)，其中包含 SVM 的凭据。- type：设置为 awsarn。有关详细信息，请参见 "创建 AWS Secrets Manager 密钥"。	

用于配置卷的后端配置选项

您可以使用配置的 defaults 部分中的这些选项来控制默认配置。有关示例，请参阅下面的配置示例。

参数	说明	默认
spaceAllocation	LUN 的空间分配	true
spaceReserve	空间预留模式；"none"（精简）或 "volume"（厚）	none
snapshotPolicy	要使用的 Snapshot 策略	none

参数	说明	默认
qosPolicy	要为创建的卷分配的 QoS 策略组。为每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。在 Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。您应该使用非共享 QoS 策略组，并确保该策略组单独应用于每个组成部分。共享 QoS 策略组强制执行所有工作负载总吞吐量的上限。	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。为每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。ontap-nas-economy 不支持此功能。	""
snapshotReserve	为快照"0"保留的卷的百分比	如果 snapshotPolicy 是 none，else ""
splitOnClone	创建时从其父级拆分克隆	false
encryption	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 false。必须在群集上许可并启用 NVE 才能使用此选项。如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。有关更多信息，请参阅："Trident 如何与 NVE 和 NAE 配合使用"。	false
luksEncryption	启用 LUKS 加密。请参见 "使用 Linux Unified Key Setup (LUKS)"。仅限 SAN。	""
tieringPolicy	要使用的分层策略 none	
unixPermissions	新卷的模式。对于 **SMB** 卷，请留空。	""
securityStyle	新卷的安全样式。NFS 支持 `mixed` 和 `unix`安全样式。SMB 支持 `mixed`和 `ntfs`安全样式。	NFS 默认值为 unix。SMB 默认值为 ntfs。

配置 PVC

本节包含有关如何创建使用已配置 Kubernetes StorageClass 来请求 PV 的 PersistentVolumeClaim（PVC）的说明。如果成功，您随后可以将该 PV 挂载到 pod。

创建 PVC

https://kubernetes.io/docs/concepts/storage/persistent-volumes["_PersistentVolumeClaim_"^]（PVC）是访问集群上 PersistentVolume 的请求。PVC 可以配置为请求特定大小或访问模式的存储。使用关联的 StorageClass，集群管理员可以控制超出 PersistentVolume 大小和访问模式的更多内容—例如性能或服务级别。

创建 Trident 后端和 StorageClass 后，您可以创建 PVC。创建 PVC 后，您可以将卷挂载到 Pod 中。

示例清单

以下示例显示了基本的 PVC 配置选项。

带 **RWX** 访问的 **PVC**

此示例显示了一个与名为 `basic-csi`的 StorageClass 相关联的具有 RWX 访问权限的基本 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-gold
```

PVC 使用 **iSCSI** 示例

此示例显示了一个与名为 `protection-gold`的 StorageClass 相关联的具有 RWO 访问权限的 iSCSI 基本 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-san
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: protection-gold
```

创建 **PVC**

步骤

1. 创建 PVC。

```
kubectl create -f pvc.yaml
```

2. 验证 PVC 状态。

```
kubectl get pvc
```

```
NAME         STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                          5m
```

请参阅"Kubernetes 和 Trident 对象"，了解存储类如何与 `PersistentVolumeClaim`交互，以及用于控制 Trident 配置卷的参数详情。

部署应用程序

创建存储类和 PVC 后，您可以将 PV 安装到 Pod 上。本节列出了将 PV 附加到 Pod 的示例命令和配置。

部署示例应用程序

步骤

1. 在 Pod 中挂载卷。

```
kubectl create -f pv-pod.yaml
```

以下示例显示了将 PVC 连接到 pod 的基本配置：基本配置：

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: pv-storage
```

备注 您可以使用 `kubectl get pod --watch` 监控进度。

2. 验证卷是否已装入 `/my/mount/path`。

```
kubectl exec -it pv-pod -- df -h /my/mount/path
```

```
Filesystem                                                      Size
Used  Avail Use%  Mounted on
192.168.188.78:/trident_pvc_ae45ed05_3ace_4e7c_9080_d2a83ae03d06  1.1G
320K  1.0G   1%   /my/mount/path
```

现在您可以删除 Pod 了。Pod 应用程序将不再存在，但卷将保持不变。

```
kubectl delete pod pv-pod
```

在 EKS 集群上配置 Trident EKS 附加组件

NetApp Trident 简化了 Kubernetes 中 Amazon FSx for NetApp ONTAP 存储管理，使您的开发人员和管理员能够专注于应用程序部署。NetApp Trident EKS 加载项包含最新的安全补丁、错误修复，并经过 AWS 验证可与 Amazon EKS 配合使用。EKS 加载项使您能够始终如一地确保 Amazon EKS 集群的安全和稳定，并减少安装、配置和更新加载项所需的工作量。

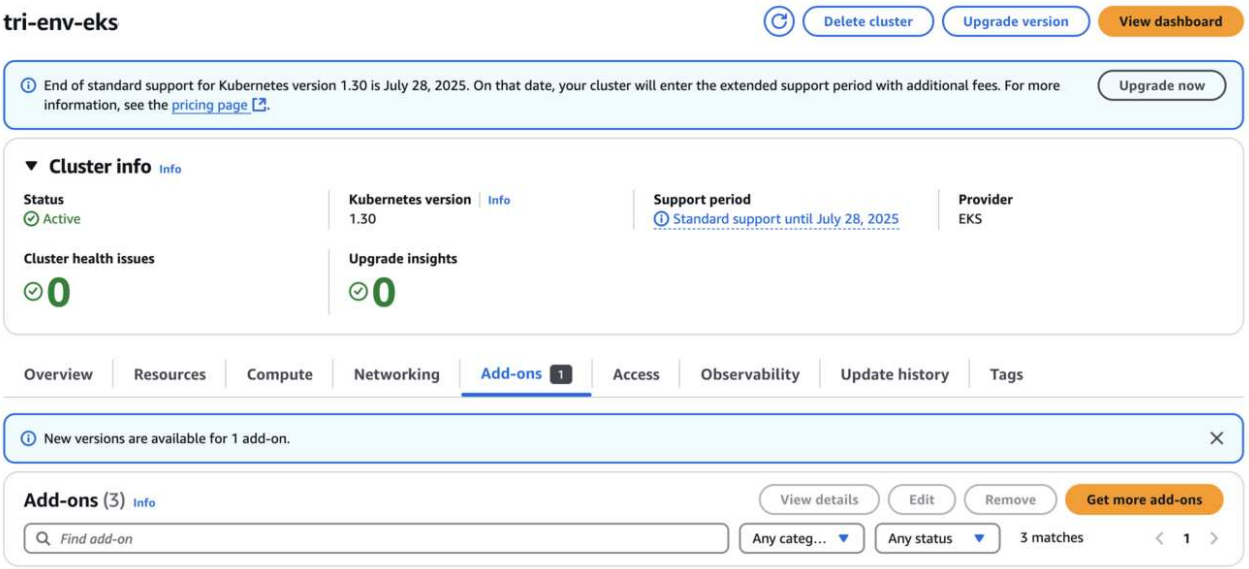
前提条件

在为 AWS EKS 配置 Trident 加载项之前，请确保具备以下条件：

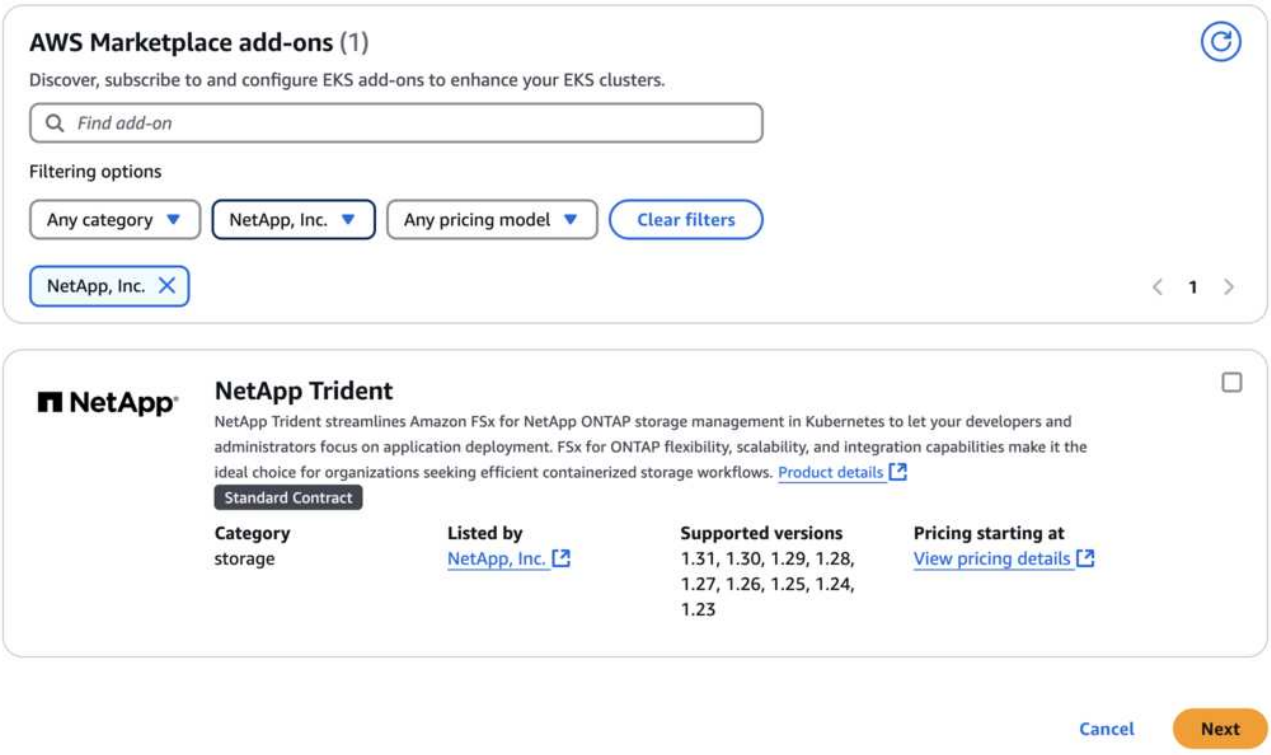
- 具有使用加载项权限的 Amazon EKS 群集帐户。请参见 "Amazon EKS 附加组件"。
- AWS 对 AWS marketplace 的权限：
  ```
  "aws-marketplace:ViewSubscriptions",
  "aws-marketplace:Subscribe",
  "aws-marketplace:Unsubscribe
  ```
- AMI 类型：Amazon Linux 2 (AL2_x86_64) 或 Amazon Linux 2 Arm(AL2_ARM_64)
- 节点类型：AMD 或 ARM
- 现有 Amazon FSx for NetApp ONTAP 文件系统

步骤

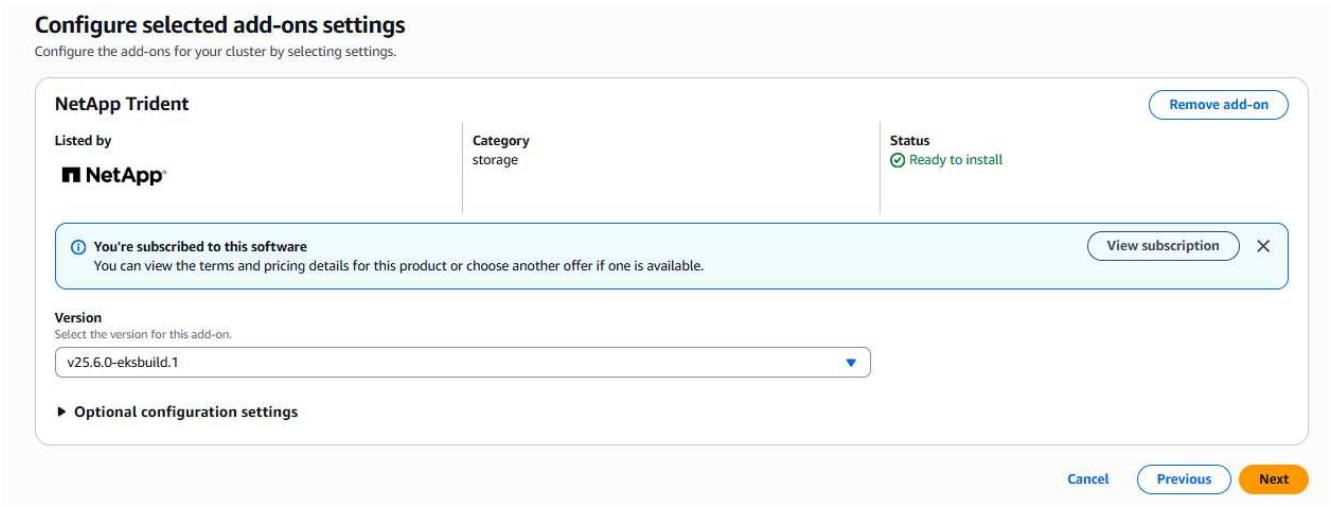
1. 请确保创建 IAM 角色和 AWS 密钥，以使 EKS Pod 能够访问 AWS 资源。有关说明，请参阅 "创建 IAM 角色和 AWS Secret"。
2. 在 EKS Kubernetes 集群上，导航到 **Add-ons** 选项卡。



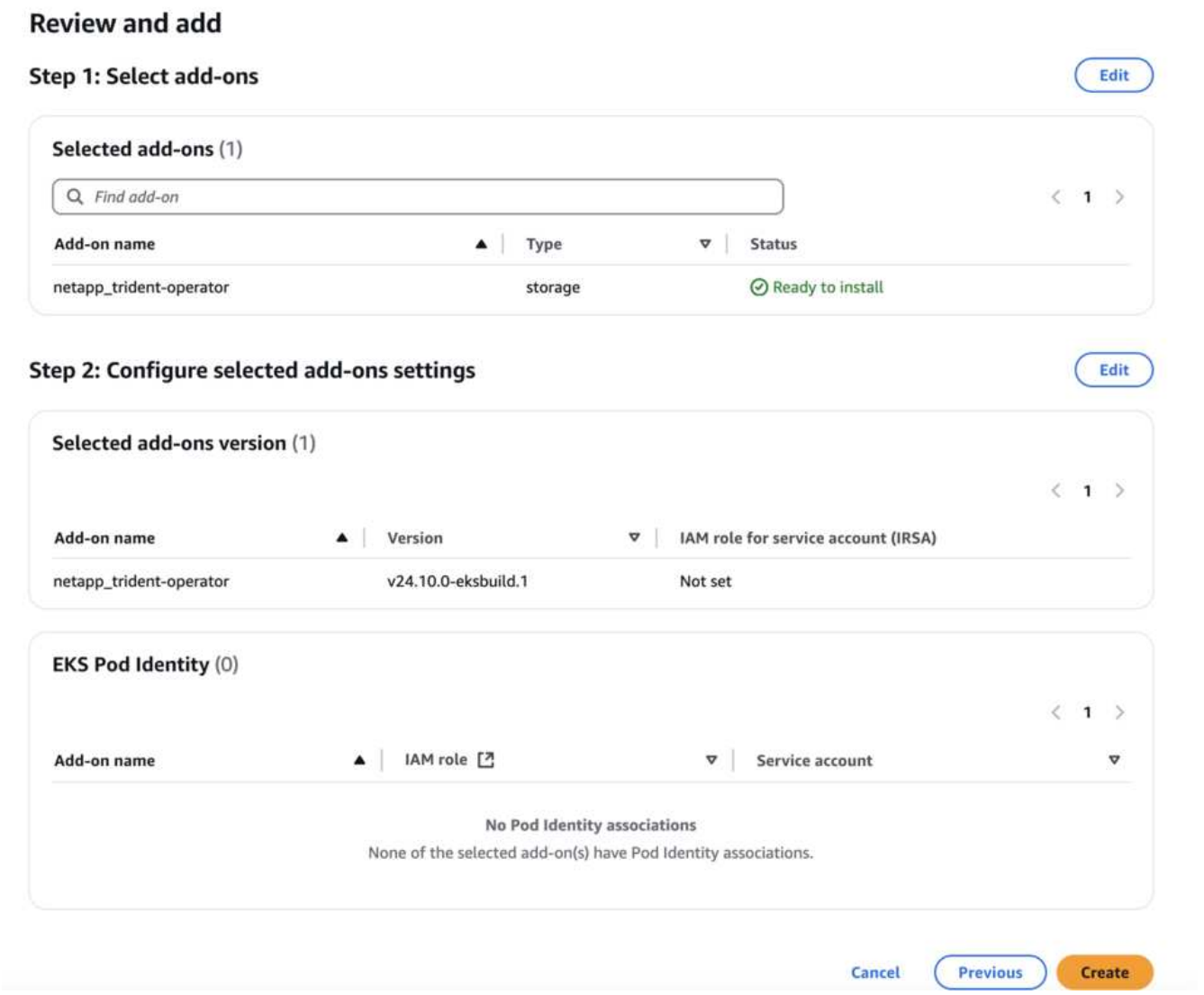
3. 转到 *AWS Marketplace 加载项*并选择 *storage* 类别。



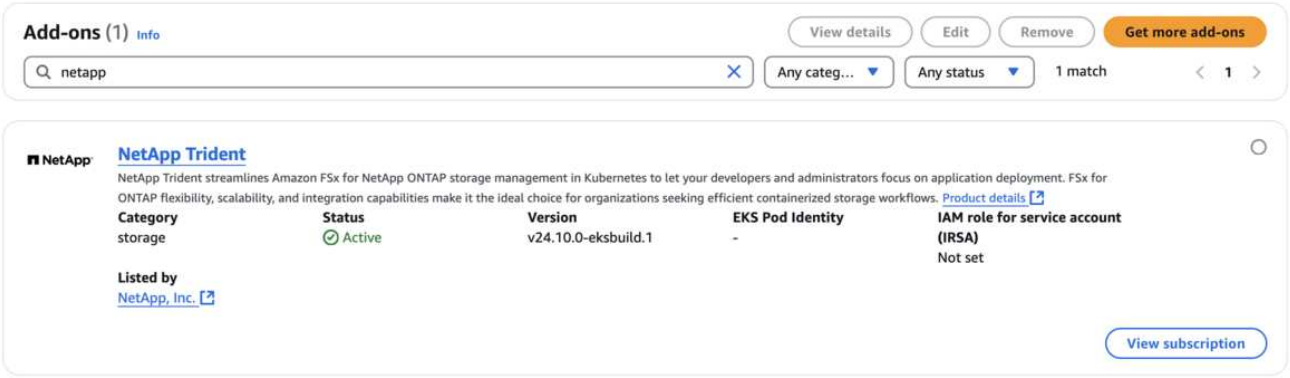
4. 找到 **NetApp Trident** 并选中 Trident 附加组件的复选框，然后单击 **Next**。
5. 选择所需的加载项版本。



6. 配置所需的附加组件设置。



7. 如果使用 IRSA（服务帐户的 IAM 角色），请参阅其他配置步骤"此处"。
8. 选择 **Create**。
9. 验证加载项的状态是否为 *Active*。



10. 运行以下命令来验证集群上是否正确安装了 Trident：

```
kubectl get pods -n trident
```

11. 继续设置并配置存储后端。有关信息，请参见"配置存储后端"。

使用 CLI 安装/卸载 Trident EKS 附加组件

使用 CLI 安装 NetApp Trident EKS 附加组件：

以下示例命令安装 Trident EKS 加载项：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.0-eksbuild.1`（带有专用版本）

以下示例命令安装 Trident EKS 插件版本 25.6.1：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.1-eksbuild.1`（带有专用版本）

以下示例命令安装 Trident EKS 插件版本 25.6.2：
`eksctl create addon --cluster clusterName --name netapp_trident-operator --version v25.6.2-eksbuild.1`（带有专用版本）

使用 CLI 卸载 NetApp Trident EKS 附加组件：

以下命令卸载 Trident EKS 加载项：

```
eksctl delete addon --cluster K8s-arm --name netapp_trident-operator
```

使用 kubectl 创建后端

后端定义 Trident 与存储系统之间的关系。它告诉 Trident 如何与该存储系统进行通信，以及 Trident 应如何从中配置卷。安装 Trident 后，下一步是创建后端。`TridentBackendConfig` 自定义资源定义 (CRD) 使您能够直接通过 Kubernetes 界面创建和管理 Trident 后端。您可以通过使用 `kubectl` 或为您的 Kubernetes 发行版使用等效的 CLI 工具来完成此操作。

`TridentBackendConfig`

`TridentBackendConfig` (`tbc`, `tbconfig`, `tbackendconfig`) 是一个前端、命名空间的 CRD，使您能够使用 `kubectl` 管理 Trident 后端。Kubernetes 和存储管理员现在可以直接通过 Kubernetes CLI 创建和管理后端，而无需专用的命令行实用程序 (`tridentctl`)。

创建 `TridentBackendConfig` 对象后，会发生以下情况：

- Trident 将根据您提供的配置自动创建后端。这在内部表示为 `TridentBackend` (`tbe`, `tridentbackend`) CR。
- `TridentBackendConfig`唯一绑定到由 `Trident` 创建的 `TridentBackend`。

每个 `TridentBackendConfig`都与 `TridentBackend`保持一对一映射。前者是提供给用户用于设计和配置后端的接口；后者是 Trident 表示实际后端对象的方式。

警告

TridentBackend CR 由 Trident 自动创建。您*不应该*修改它们。如果要对后端进行更新，请通过修改 `TridentBackendConfig`对象来执行此操作。

有关 TridentBackendConfig CR 的格式，请参见以下示例：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

您还可以查看 "trident-installer" 目录中的示例，了解所需存储平台/服务的示例配置。

`spec` 需要特定于后端的配置参数。在此示例中，后端使用 `ontap-san` 存储驱动程序，并使用此处列出的配置参数。有关所需存储驱动程序的配置选项列表，请参阅 xref:{relative_path}backends.html["您的存储驱动程序的后端配置信息"^]。

`spec` 部分还包括 `credentials` 和 `deletionPolicy` 字段，这些字段是在 `TridentBackendConfig` CR 中新引入的：

- credentials：此参数为必填字段，包含用于对存储系统/服务进行身份验证的凭据。这将设置为用户创建的 Kubernetes Secret。凭据无法以纯文本形式传递，将导致错误。
- deletionPolicy：此字段定义删除 TridentBackendConfig 时应执行的操作。它可以采用以下两种可能的值之一：
  - delete：这将导致 TridentBackendConfig CR 和相关后端都被删除。这是默认值。
  - retain：删除 TridentBackendConfig CR 时，后端定义仍将存在，可以使用 tridentctl 进行管理。将删除策略设置为 retain 允许用户降级到较早版本（21.04 之前）并保留创建的后端。此字段的值可以在创建 TridentBackendConfig 后更新。

备注

后端名称使用 spec.backendName 设置。如果未指定，则后端的名称将设置为 TridentBackendConfig 对象的名称（metadata.name）。建议使用 spec.backendName 显式设置后端名称。

提示	使用 tridentctl`创建的后端没有关联的 `TridentBackendConfig`对象。你可以通过创建 `TridentBackendConfig CR，选择用 kubectl`来管理这些后端。必须注意指定相同的配置参数（如 `spec.backendName、 spec.storagePrefix、 `spec.storageDriverName`等）。Trident 会自动将新创建的 `TridentBackendConfig`与已有的后端绑定。

步骤概述

要使用 kubectl 创建新的后端，应执行以下操作：

1. 创建 "Kubernetes Secret"。密钥包含 Trident 与存储集群/服务通信所需的凭据。
2. 创建 TridentBackendConfig 对象。其中包含有关存储群集/服务的详细信息，并引用了上一步中创建的密码。

创建后端后，您可以使用 `kubectl get tbc <tbc-name> -n <trident-namespace>`观察其状态并收集其他详细信息。

步骤 1：创建 **Kubernetes Secret**

创建一个包含后端访问凭据的 Secret。这对于每个存储服务/平台都是独一无二的。下面是一个示例：

```
kubectl -n trident create -f backend-tbc-ontap-san-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-ontap-san-secret
type: Opaque
stringData:
  username: cluster-admin
  password: password
```

下表总结了每个存储平台的 Secret 中必须包含的字段：

存储平台 **Secret** 字段描述	密钥	字段说明
Azure NetApp Files	clientID	来自应用注册的客户端 ID
Element（NetApp HCI/SolidFire）	端点	具有租户凭据的 SolidFire 集群的 MVIP
ONTAP	用户名	连接到集群/SVM 的用户名。用于基于凭据的身份验证
ONTAP	password	连接到集群/SVM 的密码。用于基于凭据的身份验证

存储平台 **Secret** 字段描述	密钥	字段说明
ONTAP	clientPrivateKey	客户端私钥的 Base64 编码值。用于基于证书的身份验证
ONTAP	chapUsername	入站用户名。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP 启动器密钥。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP 目标发起者密钥。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`

在此步骤中创建的 Secret 将在下一步创建的 `TridentBackendConfig` 对象的 `spec.credentials` 字段中被引用。

步骤 **2**：创建 `TridentBackendConfig` **CR**

您现在可以创建 `TridentBackendConfig` CR 了。在此示例中，使用 `ontap-san`驱动程序的后端是通过使用以下所示的 `TridentBackendConfig`对象创建的：

```
kubectl -n trident create -f backend-tbc-ontap-san.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
spec:
  version: 1
  backendName: ontap-san-backend
  storageDriverName: ontap-san
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  svm: trident_svm
  credentials:
    name: backend-tbc-ontap-san-secret
```

第 3 步：验证 `TridentBackendConfig` CR 的状态

现在已创建 `TridentBackendConfig` CR，您可以验证状态。请参见以下示例：

```
kubectl -n trident get tbc backend-tbc-ontap-san
NAME                        BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-san   ontap-san-backend     8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success
```

已成功创建后端并绑定到 `TridentBackendConfig` CR。

Phase 可以采用以下其中一个值：

- `Bound`： `TridentBackendConfig` CR 与后端相关联，该后端包含 `configRef`设置为 `TridentBackendConfig` CR 的 uid。
- `Unbound`：使用 "" 表示。 `TridentBackendConfig` 对象未绑定到后端。默认情况下，所有新创建的 `TridentBackendConfig` CR 都处于此阶段。阶段变更后，无法再次还原为未绑定状态。
- `Deleting`： `TridentBackendConfig` CR `deletionPolicy` 已设置为删除。删除 `TridentBackendConfig` CR 后，它将转换到"删除"状态。
  - 如果后端上不存在持久卷声明（PVC），则删除 `TridentBackendConfig`将导致 `Trident` 删除后端和 `TridentBackendConfig` CR。
  - 如果后端存在一个或多个 PVC，则会进入删除状态。 `TridentBackendConfig` CR 随后也进入删除阶段。只有在删除所有 PVC 后，才会删除后端和 `TridentBackendConfig`。
- `Lost`：与 `TridentBackendConfig` CR 关联的后端被意外或故意删除，而 `TridentBackendConfig` CR 仍然引用已删除的后端。无论 `TridentBackendConfig`值如何， `deletionPolicy` CR 仍然可以被删除。
- `Unknown`：Trident 无法确定与 `TridentBackendConfig` CR 关联的后端的状态或存在。例如，如果 API 服务器没有响应或 `tridentbackends.trident.netapp.io` CRD 丢失。这可能需要干预。

在此阶段，已成功创建后端！有几个操作可以额外处理，例如"backend 更新和 backend 删除"。

(可选) 步骤 4：获取更多详细信息

您可以运行以下命令来获取有关后端的详细信息：

```
kubectl -n trident get tbc backend-tbc-ontap-san -o wide
```

```
NAME                      BACKEND NAME         BACKEND UUID
PHASE    STATUS     STORAGE DRIVER    DELETION POLICY
backend-tbc-ontap-san    ontap-san-backend    8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8    Bound    Success    ontap-san         delete
```

此外，您还可以获取 `TridentBackendConfig` 的 YAML/JSON 转储。

```
kubectl -n trident get tbc backend-tbc-ontap-san -o yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo` 包含了 `backendName` 和 `backendUUID`，这些是响应 `TridentBackendConfig` CR 创建的后端信息。 `lastOperationStatus` 字段表示 `TridentBackendConfig` CR 最后一次操作的状态，该操作可以由用户触发（例如，用户在 `spec` 中进行了更改）或由 Trident 触发（例如，在 Trident 重启期间）。它可以是 Success 或 Failed。 `phase` 表示 `TridentBackendConfig` CR 与后端之间关系的状态。在上面的示例中， `phase` 的值为 Bound，这意味着 `TridentBackendConfig` CR 已与后端关联。

您可以运行 `kubectl -n trident describe tbc <tbc-cr-name>` 命令以获取事件日志的详细信息。

警告	您无法使用 `tridentctl`更新或删除包含关联 `TridentBackendConfig`对象的后端。要了解在 `tridentctl`和 `TridentBackendConfig`之间切换所涉及的步骤，"请参见此处"。

管理后端

使用 **kubectl** 执行后端管理

了解如何使用 `kubectl`执行后端管理操作。

删除后端

通过删除 `TridentBackendConfig`，您指示 Trident 删除/保留后端（基于 `deletionPolicy`）。要删除后端，请确保将 `deletionPolicy`设置为 `delete`。要仅删除 `TridentBackendConfig`，请确保将 `deletionPolicy`设置为 retain。这可以确保后端仍然存在，并且可以通过使用 `tridentctl`进行管理。

运行以下命令：

```
kubectl delete tbc <tbc-name> -n trident
```

Trident 不会删除正在使用的 Kubernetes Secrets `TridentBackendConfig`。Kubernetes 用户负责清理机密。删除机密时必须谨慎行事。仅当机密未被后端使用时，才应将其删除。

查看现有后端

运行以下命令：

```
kubectl get tbc -n trident
```

您还可以运行 `tridentctl get backend -n trident`或 `tridentctl get backend -o yaml -n trident`以获取所有现有后端的列表。此列表还将包括使用 `tridentctl`创建的后端。

更新后端

更新后端可能有多种原因：

- 存储系统的凭据已更改。要更新凭据，必须更新 `TridentBackendConfig` 对象中使用的 Kubernetes Secret。Trident 将使用提供的最新凭据自动更新后端。运行以下命令以更新 Kubernetes Secret：

  ```
  kubectl apply -f <updated-secret-file.yaml> -n trident
  ```

- 需要更新参数（例如正在使用的 ONTAP SVM 的名称）。
  - 您可以使用以下命令直接通过 Kubernetes 更新 `TridentBackendConfig`对象：

    ```
    kubectl apply -f <updated-backend-file.yaml>
    ```

  - 或者，您可以使用以下命令更改现有 `TridentBackendConfig` CR：

    ```
    kubectl edit tbc <tbc-name> -n trident
    ```

备注	• 如果后端更新失败，则后端继续保持其最后已知的配置。您可以通过运行 `kubectl get tbc <tbc-name> -o yaml -n trident`或 `kubectl describe tbc <tbc-name> -n trident`来查看日志以确定原因。 • 确定并更正配置文件的问题后，您可以重新运行更新命令。

使用 **tridentctl** 执行后端管理

了解如何使用 `tridentctl`执行后端管理操作。

创建后端

创建 "后端配置文件" 后，运行以下命令：

```
tridentctl create backend -f <backend-file> -n trident
```

如果后端创建失败，则表示后端配置有问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在识别并更正配置文件的问题后，只需再次运行此 `create` 命令即可。

删除后端

要从 Trident 删除后端，请执行以下操作：

1. 检索后端名称：

   ```
   tridentctl get backend -n trident
   ```

2. 删除后端：

   ```
   tridentctl delete backend <backend-name> -n trident
   ```

备注	如果 Trident 已从此后端配置仍然存在的卷和快照，则删除后端会阻止其配置新卷。后端将继续处于"Deleting"状态。

查看现有后端

要查看 Trident 知道的后端，请执行以下操作：

- 要获取摘要，请运行以下命令：

```
tridentctl get backend -n trident
```

- 要获取所有详细信息，请运行以下命令：

```
tridentctl get backend -o json -n trident
```

更新后端

创建新的后端配置文件后，运行以下命令：

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

如果后端更新失败，则说明后端配置有问题或您尝试的更新无效。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在识别并更正配置文件的问题后，只需再次运行此 `update` 命令即可。

标识使用后端的存储类

这是一个示例，说明您可以使用 `tridentctl` 为后端对象输出的 JSON 来回答这类问题。这使用了 `jq` 实用程序，您需要安装它。

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

这也适用于使用 `TridentBackendConfig` 创建的后端。

在后端管理选项之间移动

了解在 Trident 中管理后端的不同方法。

管理后端的选项

随着 `TridentBackendConfig` 的推出，管理员现在有两种独特的后端管理方式。这就提出了以下问题：

- 使用 `tridentctl`创建的后端可以通过 `TridentBackendConfig`进行管理吗?
- 使用 `TridentBackendConfig`创建的后端可以通过 `tridentctl`进行管理吗?

使用 tridentctl`管理后端 `TridentBackendConfig

本节介绍了管理通过 tridentctl 直接通过 Kubernetes 界面创建 TridentBackendConfig 对象而创建的后端所需的步骤。

这适用于以下情形：

- 已存在的后端，没有 TridentBackendConfig，因为它们是使用 tridentctl 创建的。
- 已使用 `tridentctl`创建新后端，但存在其他 `TridentBackendConfig`对象。

在这两种情况下，后端都将继续存在，Trident 调度卷并对其进行操作。管理员在此处有两种选择：

- 继续使用 tridentctl 来管理使用它创建的后端。
- 将使用 `tridentctl`创建的后端绑定到新的 `TridentBackendConfig`对象。这样做意味着后端将使用 `kubectl`而不是 `tridentctl`进行管理。

要使用 `kubectl`管理预先存在的后端，您需要创建一个 `TridentBackendConfig`绑定到现有后端。以下是其工作原理概述：

1. 创建 Kubernetes Secret。密钥包含 Trident 与存储集群/服务通信所需的凭据。
2. 创建 TridentBackendConfig 对象。其中包含有关存储群集/服务的详细信息，并引用了上一步中创建的密码。必须注意指定相同的配置参数（如 spec.backendName、 spec.storagePrefix、 spec.storageDriverName 等）。 spec.backendName 必须设置为现有后端的名称。

第 0 步：识别后端

要创建绑定到现有后端的 TridentBackendConfig，您需要获取后端配置。在此示例中，假设后端是使用以下 JSON 定义创建的：

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+---------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE   | VOLUMES |
+---------------------+----------------
+--------------------------------------+---------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+---------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

步骤 1：创建 **Kubernetes Secret**

创建一个包含后端凭据的 Secret，如以下示例所示：

```
cat tbc-ontap-nas-backend-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-nas-backend-secret
type: Opaque
stringData:
  username: cluster-admin
  password: admin-password
```

```
kubectl create -f tbc-ontap-nas-backend-secret.yaml -n trident
secret/backend-tbc-ontap-san-secret created
```

步骤 2：创建 `TridentBackendConfig` **CR**

下一步是创建一个 `TridentBackendConfig` CR，该 CR 将自动绑定到预先存在的 `ontap-nas-backend`（如本示例所示）。确保满足以下要求：

- 在 `spec.backendName`中定义了相同的后端名称。
- 配置参数与原始后端相同。
- 虚拟池（如果存在）必须保持与原始后端相同的顺序。
- 凭据通过 Kubernetes Secret 提供，而不是以纯文本形式提供。

在此情况下， `TridentBackendConfig` 将如下所示：

```
cat backend-tbc-ontap-nas.yaml
```

```yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

第 3 步：验证 `TridentBackendConfig` CR 的状态

创建 `TridentBackendConfig` 后，其阶段必须为 `Bound`。它还应反映与现有后端相同的后端名称和 UUID。

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+---------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+---------------
+--------------------------------------+--------+---------+
```

现在，将使用 tbc-ontap-nas-backend TridentBackendConfig 对象来完全管理后端。

使用 TridentBackendConfig`管理后端 `tridentctl

`tridentctl` 可用于列出使用 `TridentBackendConfig` 创建的后端。此外，管理员还可以选择通过 `tridentctl` 完全管理此类后端，方法是删除 `TridentBackendConfig` 并确保将 `spec.deletionPolicy` 设置为 `retain`。

第 **0** 步：识别后端

例如，让我们假设以下后端是使用 TridentBackendConfig 创建的：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

从输出中可以看出， `TridentBackendConfig` 已成功创建并绑定到后端 [观察后端的 UUID]。

步骤 **1**：确认 `deletionPolicy` 设置为 `retain`

让我们来看看 `deletionPolicy` 的价值。这需要设置为 `retain`。这可以确保在删除 `TridentBackendConfig` CR 时，后端定义仍然存在，并且可以使用 `tridentctl` 进行管理。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

备注	除非 `deletionPolicy`设置为 `retain`，否则请勿继续下一步。

步骤 2：删除 `TridentBackendConfig` CR

最后一步是删除 `TridentBackendConfig` CR。确认 `deletionPolicy`设置为 `retain`后，您可以继续删除：

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

删除 `TridentBackendConfig` 对象后，Trident 只需将其删除，而不会实际删除后端本身。

创建和管理存储类

创建存储类

配置 Kubernetes StorageClass 对象并创建存储类，以指导 Trident 如何配置卷。

配置 Kubernetes StorageClass 对象

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass 对象"^]` 将 `Trident` 识别为用于该类的预配程序，并指示 `Trident` 如何预配卷。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: csi.trident.netapp.io
mountOptions:
  - nfsvers=3
  - nolock
parameters:
  backendType: "ontap-nas"
  media: "ssd"
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

请参阅"Kubernetes 和 Trident 对象"，了解存储类如何与 `PersistentVolumeClaim`交互，以及用于控制 Trident 配置卷的参数详情。

创建存储类

创建 StorageClass 对象后，可以创建存储类。[存储类示例] 提供了一些可以使用或修改的基本示例。

步骤

1. 这是一个 Kubernetes 对象，因此请使用 `kubectl` 在 Kubernetes 中创建它。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

2. 现在，您应该在 Kubernetes 和 Trident 中都看到 **basic-csi** 存储类，并且 Trident 应该已经发现了后端的池。

```
kubectl get sc basic-csi
```

```
NAME         PROVISIONER             AGE
basic-csi    csi.trident.netapp.io   15h
```

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

存储类示例

Trident 可提供 "用于特定后端的简单存储类定义"。

或者，您可以编辑安装程序附带的 `sample-input/storage-class-csi.yaml.templ` 文件，并将 *`BACKEND_TYPE`* 替换为存储驱动程序名称。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

管理存储类

您可以查看现有存储类、设置默认存储类、标识存储类后端以及删除存储类。

查看现有存储类

- 要查看现有的 Kubernetes 存储类，请运行以下命令：

```
kubectl get storageclass
```

- 要查看 Kubernetes 存储类详细信息，请运行以下命令：

```
kubectl get storageclass <storage-class> -o json
```

- 要查看 Trident 的同步存储类，请运行以下命令：

```
tridentctl get storageclass
```

- 要查看 Trident 的同步存储类详细信息，请运行以下命令：

```
tridentctl get storageclass <storage-class> -o json
```

设置默认存储类

Kubernetes 1.6 增加了设置默认存储类的功能。如果用户未在 Persistent Volume Claim (PVC) 中指定存储类，则此存储类将用于配置 Persistent Volume。

- 通过在存储类定义中将注释 `storageclass.kubernetes.io/is-default-class`设置为 true 来定义默认存储类。根据规范，注释的任何其他值或缺失将被解释为 false。
- 您可以使用以下命令将现有存储类配置为默认存储类：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 类似地，您可以使用以下命令删除默认存储类注释：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

Trident 安装程序包中还有包含此注释的示例。

备注	群集中一次只能有一个默认存储类。从技术上讲，Kubernetes 不会阻止您拥有多个存储类，但它会表现得好像根本没有默认存储类一样。

标识存储类的后端

这是一个示例，说明您可以使用 `tridentctl` 为 Trident 后端对象输出的 JSON 来回答的问题类型。这使用了 `jq` 实用程序，您可能需要先安装它。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

删除存储类

要从 Kubernetes 中删除存储类，请运行以下命令：

```
kubectl delete storageclass <storage-class>
```

`<storage-class>` 应替换为您的存储类。

通过此存储类创建的任何持久卷将保持不变，Trident 将继续管理它们。

备注	Trident 为其创建的卷强制使用空白 `fsType`。对于 iSCSI 后端，建议在 StorageClass 中强制执行 `parameters.fsType`。您应该删除现有的 StorageClasses 并使用指定的 `parameters.fsType`重新创建它们。

配置和管理卷

预配卷

创建一个 PersistentVolumeClaim（PVC），该 PVC 使用配置的 Kubernetes StorageClass 来请求对 PV 的访问权限。然后，您可以将 PV 挂载到 pod 上。

概述

`https://kubernetes.io/docs/concepts/storage/persistent-volumes["_PersistentVolumeClaim_"^]`（PVC）是访问集群上 `PersistentVolume` 的请求。

PVC 可以配置为请求特定大小或访问模式的存储。使用关联的 StorageClass，集群管理员可以控制超出 PersistentVolume 大小和访问模式的更多内容——例如性能或服务级别。

创建 PVC 后，您可以将卷挂载到 pod 中。

创建 **PVC**

步骤

1. 创建 PVC。

```
kubectl create -f pvc.yaml
```

2. 验证 PVC 状态。

```
kubectl get pvc
```

```
NAME         STATUS  VOLUME     CAPACITY   ACCESS MODES  STORAGECLASS AGE
pvc-storage Bound   pv-name    1Gi        RWO                   5m
```

1. 在 Pod 中挂载卷。

```
kubectl create -f pv-pod.yaml
```

> 备注　您可以使用 `kubectl get pod --watch` 监控进度。

2. 验证卷是否已装入 `/my/mount/path`。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. 现在您可以删除 Pod 了。Pod 应用程序将不再存在，但卷将保持不变。

```
kubectl delete pod pv-pod
```

示例清单

PersistentVolumeClaim 示例清单

这些示例显示了基本的 PVC 配置选项。

带 **RWO** 访问的 **PVC**

此示例显示了一个与名为 `basic-csi`的 StorageClass 相关联的具有 RWO 访问权限的基本 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

带 **NVMe/TCP** 的 **PVC**

此示例显示了一个与名为 `protection-gold`的 StorageClass 相关联的具有 RWO 访问权限的 NVMe/TCP 基本 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Pod 清单示例

这些示例显示了将 PVC 连接到 pod 的基本配置。

基本配置

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

基本 NVMe/TCP 配置

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

请参阅"Kubernetes 和 Trident 对象"，了解存储类如何与 `PersistentVolumeClaim`交互，以及用于控制 Trident 配置卷的参数详情。

扩展卷

Trident 为 Kubernetes 用户提供了在创建卷后扩展卷的功能。查找有关扩展 iSCSI、NFS、SMB、NVMe/TCP 和 FC 卷所需配置的信息。

扩展 iSCSI 卷

您可以使用 CSI 置备程序扩展 iSCSI 永久卷 (PV)。

备注	`ontap-san`、 `ontap-san-economy`、 `solidfire-san` 驱动程序支持 iSCSI 卷扩展，并且需要 Kubernetes 1.16 及更高版本。

步骤 1：配置 StorageClass 以支持卷扩展

编辑 StorageClass 定义以将 `allowVolumeExpansion` 字段设置为 `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的 StorageClass，请对其进行编辑以包含 `allowVolumeExpansion` 参数。

步骤 2：使用创建的 StorageClass 创建 PVC

编辑 PVC 定义并更新 `spec.resources.requests.storage`以反映新期望的尺寸，该尺寸必须大于原始尺寸。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (PVC) 相关联。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

步骤 **3**：定义一个连接 **PVC** 的 **pod**

将 PV 连接到 pod 以调整其大小。调整 iSCSI PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 在扩展操作成功完成后更新 PVC 大小。

在此示例中，创建了一个使用 `san-pvc` 的 Pod。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

步骤 **4**：扩展 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage`更新为 2Gi。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

步骤 **5**：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证扩展是否正常工作：

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

扩展 FC 卷

您可以使用 CSI 配置程序扩展 FC 持久卷 (PV)。

> 备注 `ontap-san` 驱动程序支持 FC 卷扩展，需要 Kubernetes 1.16 及更高版本。

步骤 1：配置 StorageClass 以支持卷扩展

编辑 StorageClass 定义以将 `allowVolumeExpansion` 字段设置为 `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的 StorageClass，请对其进行编辑以包含 `allowVolumeExpansion` 参数。

步骤 2：使用创建的 **StorageClass** 创建 **PVC**

编辑 PVC 定义并更新 `spec.resources.requests.storage`以反映新期望的尺寸，该尺寸必须大于原始尺寸。

```
cat pvc-ontapsan.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (PVC) 相关联。

```
kubectl get pvc
NAME       STATUS    VOLUME                                     CAPACITY
ACCESS MODES    STORAGECLASS    AGE
san-pvc    Bound     pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO             ontap-san       8s

kubectl get pv
NAME                                        CAPACITY    ACCESS MODES
RECLAIM POLICY    STATUS    CLAIM              STORAGECLASS    REASON    AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671    1Gi         RWO
Delete            Bound     default/san-pvc    ontap-san                 10s
```

步骤 3：定义一个连接 **PVC** 的 **pod**

将 PV 连接到 pod 以调整其大小。调整 FC PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 在扩展操作成功完成后更新 PVC 大小。

在此示例中，创建了一个使用 `san-pvc` 的 Pod。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

步骤 **4**：扩展 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage`更新为 2Gi。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

步骤 **5**：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证扩展是否正常工作：

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

扩展 **NFS** 卷

Trident 支持在 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`和 `azure-netapp-files`后端上配置的 NFS PV 的卷扩展。

步骤 **1**：配置 **StorageClass** 以支持卷扩展

要调整 NFS PV 的大小，管理员首先需要通过将 `allowVolumeExpansion`字段设置为 `true`来配置存储类以允许卷扩展：

```
cat storageclass-ontapnas.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

如果您已经创建了一个没有此选项的存储类，则可以通过使用 `kubectl edit storageclass` 来允许卷扩展

，从而编辑现有存储类。

步骤 2：使用创建的 **StorageClass** 创建 **PVC**

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident 应为此 PVC 创建 20 MiB NFS PV：

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES    STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

步骤 3：扩展 **PV**

要将新创建的 20 MiB PV 调整为 1 GiB，请编辑 PVC 并设置 `spec.resources.requests.storage`为 1 GiB：

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

步骤 **4**：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证调整大小是否正确：

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

了解 **RWX NVMe** 子系统限制

使用 NVMe 协议的 ReadWriteMany (RWX) 卷的可扩展性限制为每卷 64 个节点。以下包括限制，解释了所涉及的 NVMe 子系统体系结构，并概述了所需的解决步骤。

了解 **64** 节点限制

如果您计划将 ReadWriteMany (RWX) 卷与 NVMe 协议一起使用，则单个 RWX NVMe 卷不能由 Kubernetes 群集中超过 64 个节点装载。

请勿计划在 64 个以上节点上挂载相同 RWX NVMe PersistentVolumeClaim 的工作负载。

此限制仅适用于使用 NVMe 协议的 RWX 卷。

了解 **NVMe** 子系统模型

每卷子系统模型（**Trident** 版本早于 **26.02**）

在 26.02 之前的 Trident 版本中，RWX NVMe 卷使用每个卷的子系统模型进行调配。每个 RWX NVMe 卷都映射到 ONTAP 上其自己的专用 NVMe 子系统。

此模型很简单，但具有较低的可扩展性限制。在大型 Kubernetes 集群中，由于每个 RWX 卷消耗一个专用子系统，因此子系统控制器限制很快就会达到。

超级子系统模型（在 **Trident 26.02** 中引入）

从 Trident 26.02 开始，RWX NVMe 卷使用共享超级子系统模型。多个 RWX NVMe 卷共享同一个 NVMe 子系统。

每个超级子系统最多支持 1024 个命名空间（卷）。该模型显著提高了 RWX 工作负载的可扩展性，并降低了达到 ONTAP 子系统限制的可能性。

每个 RWX NVMe 卷最多支持 64 个节点。

识别错误症状

如果大规模创建或附加 RWX NVMe 卷，您可能会发现类似于以下错误：

```
Maximum number of controllers reached. No more controllers can be created.
```

此错误表示已达到 ONTAP NVMe 子系统控制器限制。

解决子系统限制错误

要超越每卷子系统的限制并利用超级子系统模型，请升级到 Trident 26.02 或更高版本。

升级 **Trident** 以应用超级子系统模型

要为 RWX NVMe 卷应用超级子系统模型：

1. 将 Trident 升级到 26.02 或更高版本。
2. 将使用 RWX NVMe 卷的所有 Pod 缩小到零副本。
3. 确认没有工作负载正在使用 RWX NVMe 卷。
4. 将 pod 重新扩容。

此重新启动顺序可确保使用超级子系统模型连接 RWX NVMe 卷。

- 此限制仅适用于使用 NVMe 协议的 RWX 卷。
- 64 节点限制适用于每个 RWX NVMe 卷。
- 其他访问模式和其他协议不受影响。

控制器可扩展性

Trident 通过改进多个存储驱动程序的并发性来引入控制器可扩展性。客户可以确定哪些 Trident 驱动程序在一般可用时支持控制器可扩展性，以及哪些驱动程序在 Trident 26.02 中作为技术预览可用。这为可扩展的 Kubernetes 环境提供了明智的部署决策和适当的风险管理。

关键概念和定义

控制器可扩展性

控制器可扩展性是指 Trident 控制器并行处理多个存储操作的能力，而不是在单个锁定后对其进行序列化。这些操作包括卷创建、删除、调整大小、快照创建和删除、卷发布和取消发布以及后端管理。

启用控制器可扩展性后，不同卷和后端上的操作将同时进行。这增加了吞吐量，并减少了具有大量并发 PersistentVolumeClaim 和 VolumeSnapshot 操作的环境中的端到端操作时间。

控制器可扩展性支持

Trident 支持不同成熟度级别的控制器可扩展性，具体取决于存储驱动程序。

正式发布

以下驱动程序在 Trident 26.02 正式上市时支持控制器可扩展性：

- `ontap-san`
- `ontap-nas`
- `google-cloud-netapp-volumes`

备注

`google-cloud-netapp-volumes` 和 `google-cloud-netapp-volumes-san` 驱动程序不同。仅支持 `google-cloud-netapp-volumes`。请勿在后端配置或示例中使用 `google-cloud-netapp-volumes-san`。

启用控制器可扩展性

控制器可扩展性由 `enableConcurrency` 配置选项控制。必须在 Trident 安装期间或通过更新现有部署显式启用此选项。

Trident 操作员部署

要通过 Trident operator 实现控制器可扩展性，请在 `TridentOrchestrator` 自定义资源中将 `enableConcurrency` 设置为 `true`。

新安装

创建或编辑 `TridentOrchestrator` CR，将 `enableConcurrency` 设置为 `true`：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

应用 CR：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

现有安装

修补现有 `TridentOrchestrator` CR 以启用控制器可扩展性：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

验证设置是否已应用：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm 部署

要使用 Helm 启用控制器可扩展性，请将 `enableConcurrency`值设置为 `true`。

新安装

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

现有安装

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

或者，在自定义 `values.yaml` 文件中将 `enableConcurrency` 设置为 `true`：

```
# values.yaml
enableConcurrency: true
```

然后使用 values 文件进行安装或升级：

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl 部署

要使用 `tridentctl` 启用控制器可扩展性，请在安装过程中传递 `--enable-concurrency` 标志。

新安装

```
tridentctl install -n trident --enable-concurrency
```

现有安装

要在现有基于 `tridentctl` 的部署上启用控制器可扩展性，请卸载并使用标志重新安装：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

验证是否已启用控制器可扩展性

启用控制器可扩展性后，通过检查控制器 Pod 日志来验证 Trident 控制器正在启用并发运行：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

您应该会看到一个日志条目，指示已启用并发。

技术预览

以下驱动程序支持控制器可扩展性作为 Trident 26.02 的技术预览：

- `nas-eco`
- `san-eco`

对于这些驱动程序：

- 控制器并发可用于评估和测试
- 行为在未来版本中可能会发生变化

- 不建议在生产环境中使用

并发行为

启用控制器可扩展性时：

- Trident 将单个全局锁定替换为每个资源的细粒度锁定
- 修改相同资源的操作被序列化，以保持数据的一致性
- 只能从资源读取的操作可以与该资源上的其他读取操作同时进行
- Trident 将并发 ONTAP API 请求限制为每个管理 LIF 20 个，以防止后端存储系统过载
- 如果多个后端共享相同的管理 LIF，则它们共享此 20 个请求的限制

已知限制和注意事项

以下考虑因素适用于控制器可扩展性：

- 并发由 Trident 控制器内部管理
- 此版本中没有用户可配置的并发限制
- 总吞吐量取决于：
  - 正在使用的存储驱动程序
  - 后端响应能力
  - Kubernetes API 服务器性能
- 高并发可能会增加后端存储系统的负载

注意事项和限制

Trident 26.02 适用以下限制：

- 控制器可扩展性行为在所有驱动程序中并不相同
- 技术预览驱动程序可能表现出：
  - 高负载下性能不一致
  - 发布之间的行为变化
- 由于并行执行，调试并发操作可能会更复杂
- 指标和日志可能显示交错操作输出

建议

- 将通用可用性 (GA) 驱动程序用于需要高可扩展性的生产环境
- 在非生产环境中评估技术预览驱动程序
- 大规模运行时监控后端和控制器性能
- 避免在自动化脚本中假设操作顺序

自动卷扩展

自动卷扩展使 Trident 配置的持久卷能够在使用的容量达到定义的阈值时自动增长。此功能可降低运营开销，并有助于防止因容量耗尽而导致的应用程序中断。使用 Autogrow Policies 实现自动卷扩展。Autogrow Policy 定义：

- 触发扩展的利用率阈值
- 卷增长的量
- 卷可以达到的最大大小

备注	当超过定义的利用率阈值时，卷的大小会自动增加。卷永远不会自动缩减。

要求

在配置自动卷扩展之前，请确保满足以下要求：

- Trident 26.02 或更高版本
- 用于创建 `TridentAutogrowPolicy`自定义资源的基于角色的访问控制权限
- StorageClasses 配置为 `allowVolumeExpansion: true`
- 支持的 ONTAP 协议：
  - 网络文件系统 (NFS)
  - Internet 小型计算机系统接口 (iSCSI)
  - 非易失性内存快速通道 (NVMe)
  - 光纤通道协议 (FCP)

限制

- 早于 ONTAP 9.16.1 的 ONTAP Non-Volatile Memory Express 原始块卷不支持自动扩展。
- 对于存储区域网络卷，如果 `growthAmount`小于或等于 `50 mebibytes`，则 `Trident` 会在调整大小之前自动将值增加到 `51 mebibytes`，前提是生成的大小不超过 `maxSize`。
- 在棕地环境中，由于卷发布迁移行为，某些现有卷可能无法自动扩展。
- 当卷达到 `maxSize`时，不会发生进一步的扩展。
- 支持自动卷扩展的协议：
  - 网络文件系统 (NFS)
  - Internet 小型计算机系统接口 (iSCSI)
  - 非易失性内存快速通道 (NVMe)
  - 光纤通道协议 (FCP)

使用自动增长策略配置卷

可以在两个级别配置自动增长策略：

- 存储类级别：为所有卷设置默认值（使用注释）
- PVC 级别：覆盖存储类默认值（使用注释）

创建 **Autogrow** 策略

当卷达到定义的容量阈值时，Autogrow Policies 启用自动卷扩展。

请确保您拥有：

- 已安装 Trident 26.02 或更高版本
- 基于角色的访问控制权限以创建 `TridentAutogrowPolicy` 资源
- 了解工作负载增长需求

Autogrow Policy 定义了卷在达到定义的容量阈值时如何自动扩展。

您可以在工作流程中随时创建 Autogrow Policies：

- 在创建 StorageClasses 和卷之前
- 在 StorageClasses 存在后
- 配置卷后

这种灵活性使您能够引入自动扩展，而无需重新创建现有资源。

Autogrow 策略规范

Autogrow Policies 是 Kubernetes 自定义资源，定义如下：

字段	说明	格式	必填项	示例	默认
name	唯一策略标识符	字符串	是	production-db-policy	无
usedThreshold	触发扩展的容量百分比	百分比字符串	是	"80%"	无
growthAmount	达到阈值时要增长的金额	百分比或大小	否	"10%" 或 "5Gi"	"10%"
maxSize	最大卷大小限制	Kubernetes 数量	否	"500Gi"	无限制

创建 **Autogrow** 策略

步骤

1. 创建定义自动增长策略的 YAML 文件：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. 将策略应用于您的集群：

```
kubectl apply -f autogrow-policy.yaml
```

3. 验证策略是否已创建：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

预期输出

```
NAME                USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow   80%              10%             Success
```

策略状态

创建策略后，Trident 验证规范并分配以下状态之一：

州/省	说明	需采取行动
成功	策略已验证并可供使用。	无。
失败	检测到验证错误。	查看并修复规范。
正在删除	正在执行删除。	等待完成。

将策略与 **StorageClass** 关联

您可以使用 `trident.netapp.io/autogrowPolicy`批注将自动增长策略与StorageClass关联起来。从该StorageClass配置的所有卷都会继承该策略。

备注	StorageClass 必须具有 `allowVolumeExpansion: true`。

步骤

1. 使用 Autogrow Policy 注释创建或修改 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. 应用 StorageClass：

```
kubectl apply -f storageclass.yaml
```

3. 验证标注：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
production-db-policy
```

策略优先级

当在 StorageClass 和 PVC 上设置 Autogrow Policy 注释时，Trident 应用以下优先级规则：

1. *PVC 注释优先。*如果 PVC 设置 `trident.netapp.io/autogrowPolicy`，则始终使用该值。
2. **StorageClass** 注释仅适用于 **PVC** 没有注释的情况。
3. 如果两者都没有注释， 则不会应用 Autogrow Policy。

StorageClass 标注	**PVC** 标注	有效行为
`trident.netapp.io/autogrowPolicy: standard-agp`	未设置	用途 `standard-agp`。
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	使用 `logs-policy`（PVC 覆盖 StorageClass）。

StorageClass 标注	PVC 标注	有效行为
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	无自动增长策略（PVC 禁用自动增长）。
未设置	`trident.netapp.io/autogrowPolicy: dev-policy`	用途 `dev-policy`。
未设置	未设置	无自动增长策略。

配置示例

以下示例显示了不同用例的常见 Autogrow Policy 配置。

生产数据库的保守策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

具有固定增长增量的日志存储

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

具有默认值的最小策略

当您忽略 `growthAmount`和 `maxSize`时，Trident 使用默认值(`10%`增长，无限大小）：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

具有自定义 **maxSize** 和默认 **growthAmount** 的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

无限制 **maxSize** 的激进增长

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

具有分数百分比的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

备注 支持分数百分比。如果指定的小数位数超过三位，Trident 会将值舍入到小数点后三位。

NAS StorageClass 使用 Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass 使用 Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

管理 Autogrow 策略

创建 Autogrow 策略后，可以根据需要查看、更新和删除它们。您还可以监控哪些卷正在使用给定的策略。

查看 Autogrow 策略

列出所有策略

使用 `kubectl` 列出集群中的所有 Autogrow Policies：

```
kubectl get tridentautogrowpolicy
```

或者，也可以使用 `tridentctl`：

```
tridentctl get autogrowpolicy
```

查看策略详细信息

要查看策略的完整规范和状态：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

要以 YAML 格式查看策略及其关联的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

更新 **Autogrow** 策略

您可以修改现有策略以更改其阈值、增长量或最大大小。更改将立即对使用该策略的所有卷生效。

重要说明	更改会影响当前使用此策略的所有卷。尽可能先在非生产环境中测试更改。

步骤

1. 编辑策略：

   ```
   kubectl edit tridentautogrowpolicy production-db-policy
   ```

2. 根据需要修改 `spec`字段：

   ```
   spec:
     usedThreshold: "75%"     # Changed from 80%
     growthAmount: "20%"      # Changed from 10%
     maxSize: "1Ti"           # Changed from 500Gi
   ```

3. 保存并退出。更改立即生效。

更新注意事项

- 即时效应： 所有使用该策略的卷在下一次增长评估时都采用新参数。
- *无需重新启动卷：*更改适用于下一次增长操作。
- 先测试： 尽可能在非生产环境中验证更改。
- *沟通更改：*修改共享策略时通知团队。

删除 **Autogrow** 策略

Autogrow 策略使用终结器保护，以防止在卷正在使用时意外删除。

步骤

1. 删除策略：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. 如果卷仍在使用策略，则删除进入 `Deleting` 状态。检查受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 从每个受影响的卷中删除策略。从下列选项中选择一项：

   - 选项 **A**：通过将注释设置为 **`"none"`**来显式禁用自动增长：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

   - 选项 **B**：完全删除注释：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

删除行为

场景	行为
没有卷使用该策略	策略将立即删除。
卷正在使用此策略	策略进入 `Deleting`状态。终结器阻止完成，直到删除所有卷。
所有卷均已从策略中删除	终结器将被删除，策略也将被删除。

监控 **Autogrow** 策略使用情况

使用策略检查卷

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

查找卷使用的策略

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

监控策略事件

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

支持的协议

Autogrow 支持以下存储协议：

- NFS
- iSCSI
- FCP
- NVMe

备注	对于 SAN 卷，如果配置的 `growthAmount`为 `50 MiB` 或更少，`Trident` 会自动将调整大小操作的增长量增加到 `51 MB`，前提是结果大小不超过 `maxSize`。

已知限制

- **ONTAP NVMe** 原始块卷： 使用早于 9.16.1 的 ONTAP 版本创建的卷不支持自动增长。
- *现有卷（brownfield 部署）：*即使应用了有效的 Autogrow Policy，Autogrow 可能也不适用于现有卷。这是由于卷发布的持续迁移。要确认迁移已完成，请检查 Trident 控制器日志中的 `"Migration completed"`消息。

常见问题解答

Trident 何时评估阈值?

Trident 持续监控卷使用情况。当使用的容量超过 `usedThreshold`时，Trident 会创建一个内部调整大小请求，并按配置的 `growthAmount`扩展卷。

例如，此策略将以 80% 的容量触发扩展，并每次将卷增长 10%，最多可达 500 GiB：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

是否可以在已配置卷后应用策略?

是您可以随时创建 Autogrow Policy，并通过添加或更新 `trident.netapp.io/autogrowPolicy`注释将其应用于现有 PVC。您无需重新创建 PVC 或 StorageClass。

将策略应用于现有 PVC：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

要将策略应用于现有 StorageClass：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

如果我同时在 **StorageClass** 和 **PVC** 上设置自动增长策略，会发生什么?

PVC 注释始终优先。如果 PVC 具有 `trident.netapp.io/autogrowPolicy`注释，则无论StorageClass指定什么，Trident 都会使用该值。有关详细信息，请参见 "策略优先级"。

例如，假设此 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

此 PVC 覆盖 StorageClass 策略：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Trident 使用 `logs-policy`用于 `database-pvc`，而不是 `standard-agp`。

如何禁用特定卷的自动增长？

将 PVC 注释设置为 `"none"`。这将覆盖该卷的任何 StorageClass 级别策略：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

您可以验证是否已禁用自动增长：

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
none
```

当卷达到 **maxSize** 时会发生什么？

Trident 停止扩展卷。即使使用量继续增加超过 `usedThreshold`，也不会为该卷创建进一步的调整大小请求。

例如，通过此策略，Trident 在卷达到 100 GiB 后停止增长：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

要允许无限增长，请忽略 `maxSize` 或将其设置为 `0`：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

我可以在不重新启动卷的情况下更改策略吗?

是更新策略时，使用该策略的所有卷将在下一次增长评估时采用新参数。不需要重新启动卷。

要就地更新策略：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

根据需要修改字段：

```
spec:
  usedThreshold: "75%"     # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

保存并退出。验证更新后的策略：

```
kubectl get tridentautogrowpolicy production-db-policy
```

预期输出

```
NAME                    USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy    75%              20%             Success
```

为什么我的策略处于失败状态?

`Failed` 状态表示策略规范包含验证错误。运行以下命令来查看错误详细信息：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

常见原因包括无效的 `usedThreshold`（必须为 1–99%）、 `growthAmount` 超过 `maxSize`，或无效的 Kubernetes 数量格式。请更正规范并重新应用：

```
kubectl apply -f autogrow-policy.yaml
```

为什么我无法删除策略?

策略使用终结器保护。如果卷仍在使用该策略，则删除将进入 `Deleting` 状态，并等待从策略中删除所有卷。

识别受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

然后从每个 PVC 中删除注释：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

删除所有卷后，将释放终结器并删除策略。

自动增长是否适用于所有 **ONTAP** 后端?

Autogrow 支持 NFS、iSCSI、FCP 和 NVMe 协议。但是，NVMe 原始块卷需要 ONTAP 9.16.1 或更高版本。

棕色字段部署中的现有卷可能需要在自动增长生效之前完成卷发布迁移。通过检查 Trident 控制器日志来验证迁

移状态：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
completed"
```

以下 StorageClass 示例显示为 NAS 和 SAN 后端配置的自动增长：

NAS 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 卷的最低增长量是多少?

对于 SAN 卷，有效最低增长量为 51 MB。如果将 `growthAmount`配置为 50 MiB 或更少，Trident 会自动将调整大小操作的增长增加到 51 MB。

例如，此策略设置 `growthAmount`的 `"40Mi"`，但 Trident 对使用它的任何 SAN 卷应用 51 MB 增长：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

要避免这种自动调整，请将 `growthAmount`设置为大于 50 MiB 的值：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的持久卷声明 (PVC) 将现有存储卷导入为 Kubernetes PV。

概述和注意事项

您可以将卷导入 Trident 以：

- 容器化应用程序并重用其现有数据集
- 将数据集的克隆用于临时应用程序
- 重建失败的 Kubernetes 集群
- 在灾难恢复期间迁移应用程序数据

注意事项

在导入卷之前，请查看以下注意事项。

- Trident 只能导入 RW（读写）类型的 ONTAP 卷。DP（数据保护）类型卷是 SnapMirror 目标卷。在将卷导入 Trident 之前，您应该中断镜像关系。
- 我们建议导入没有活动连接的卷。要导入正在使用的卷，请克隆该卷，然后执行导入。

警告	这对于块卷尤其重要，因为 Kubernetes 不会意识到以前的连接，并且可以轻松地将活动卷附加到 pod。这可能会导致数据损坏。

- 虽然 `StorageClass`必须在 PVC 上指定，但 Trident 在导入过程中不使用此参数。存储类用于在卷创建期间根据存储特性从可用池中进行选择。由于卷已存在，因此在导入过程中不需要选择池。因此，即使卷存在于与 PVC 中指定的存储类不匹配的后端或池中，导入也不会失败。
- 现有卷大小在 PVC 中确定和设置。卷由存储驱动程序导入后，PV 将使用 ClaimRef 创建到 PVC。
  - 回收策略最初在 PV 中设置为 `retain`。Kubernetes 成功绑定 PVC 和 PV 后，回收策略将更新为与 Storage Class 的回收策略匹配。
  - 如果存储类的回收策略为 `delete`，则在删除 PV 时将删除存储卷。
- 默认情况下，Trident 管理 PVC 并重命名后端上的 FlexVol 卷和 LUN。您可以传递 `--no-manage` 标记以导入非托管卷，并传递 `--no-rename` 标记以保留卷名称。
  - `--no-manage`* - 如果您使用 `--no-manage` 标志，Trident 不会在对象生命周期中对 PVC 或 PV 执行任何附加操作。删除 PV 时不会删除存储卷，也会忽略其他操作，如卷克隆和卷大小调整。
  - `--no-rename`* - 如果使用 `--no-rename` 标志，Trident 保留现有卷名，同时导入卷并管理卷的生命周期。只有 `ontap-nas`、 `ontap-san`（包括 ASA r2 系统）和 `ontap-san-economy` 驱动程序才支持此选项。

提示	如果要将 Kubernetes 用于容器化工作负载，但要管理 Kubernetes 之外的存储卷的生命周期，则这些选项非常有用。

- 在 PVC 和 PV 上添加了一个注释，该注释具有双重目的，用于指示已导入该卷以及是否已管理 PVC 和 PV。不应修改或删除此注释。

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的 PVC 来导入卷。

备注	如果使用 PVC 注释，则无需下载或使用 `tridentctl`来导入卷。

使用 **tridentctl**

步骤

1. 创建一个 PVC 文件（例如， `pvc.yaml`），该文件将用于创建 PVC。PVC 文件应包括 `name`、`namespace`、 `accessModes`和 `storageClassName`。或者，您可以在 PVC 定义中指定 `unixPermissions`。

   以下是最小规格的示例：

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: my_claim
     namespace: my_namespace
   spec:
     accessModes:
       - ReadWriteOnce
     storageClassName: my_storage_class
   ```

备注	仅包括所需的参数。PV 名称或卷大小等附加参数可能导致导入命令失败。

2. 使用 `tridentctl import`命令可指定包含卷的 Trident 后端的名称以及唯一标识存储上卷的名称（例如：ONTAP FlexVol、Element Volume）。指定 PVC 文件的路径需要 `-f`参数。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

使用 **PVC** 注释

步骤

1. 使用所需的 Trident 导入注释创建 PVC YAML 文件（例如， `pvc.yaml`）。PVC 文件应包括：

   - `name` 和 `namespace` 在元数据中
   - `accessModes`、 `resources.requests.storage` 和 `storageClassName` 在规格中
   - 标注：
     - `trident.netapp.io/importOriginalName`: 后端上的卷名称
     - `trident.netapp.io/importBackendUUID`：卷存在的后端 UUID
     - `trident.netapp.io/notManaged` (*Optional*)：为非托管卷设置为 `"true"`。默认值为 `"false"`。

     下面是导入托管卷的示例规范：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. 将 PVC YAML 文件应用到 Kubernetes 集群：

```
kubectl apply -f <pvc-file>.yaml
```

Trident 将自动导入卷并将其绑定到 PVC。

示例

查看以下受支持驱动程序的卷导入示例。

ONTAP NAS 和 ONTAP NAS FlexGroup

Trident 支持使用 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序进行卷导入。

备注	• Trident 不支持使用 `ontap-nas-economy` 驱动程序进行卷导入。 • `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名。

使用 `ontap-nas` 驱动程序创建的每个卷都是 ONTAP 集群上的一个 FlexVol 卷。使用 `ontap-nas` 驱动程序导入 FlexVol 卷的工作原理相同。ONTAP 集群上已存在的 FlexVol 卷可以作为 `ontap-nas` PVC 导入。同样，FlexGroup 卷可以作为 `ontap-nas-flexgroup` PVC 导入。

使用 **tridentctl** 的 **ONTAP NAS** 示例

以下示例演示如何使用 `tridentctl` 导入托管卷和非托管卷。

托管卷

以下示例导入名为 `managed_volume` 的卷到名为 `ontap_nas` 的后端：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

非受管卷

使用 `--no-manage`参数时， Trident 不会重命名卷。

以下示例在 `ontap_nas` 后端导入 `unmanaged_volume`：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

使用 **PVC** 注释的 **ONTAP NAS** 示例

以下示例演示如何使用 PVC 注释导入托管和非托管卷。

托管卷

以下示例从后端 `81abcb27-ea63-49bb-b606-0a5315ac5f21`导入名为 `ontap_volume1`的 1GiB `ontap-nas` 卷，使用 PVC 注释设置了 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

非受管卷

以下示例使用 PVC 注释从后端 `34abcb27-ea63-49bb-b606-0a5315ac5f34`导入名为 `ontap-volume2`的 1Gi `ontap-nas` 卷，并设置 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Trident 支持使用 `ontap-san`（iSCSI、NVMe/TCP 和 FC）和 `ontap-san-economy`驱动程序进行卷导入。

Trident 可以导入包含单个 LUN 的 ONTAP SAN FlexVol 卷。这与 `ontap-san`驱动程序一致，该驱动程序为每个 PVC 创建一个 FlexVol 卷，并在 FlexVol 卷中创建一个 LUN。Trident 导入 FlexVol 卷并将其与 PVC 定义相关联。Trident 可以导入包含多个 LUN 的 `ontap-san-economy`卷。

以下示例显示了如何导入托管卷和非托管卷：

托管卷

对于托管卷，Trident 会将 FlexVol 卷重命名为 `pvc-<uuid>` 格式，并将 FlexVol 卷中的 LUN 重命名为 `lun0`。

以下示例导入存在于 `ontap_san_default` 后端的 `ontap-san-managed` FlexVol volume：

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

非受管卷

以下示例在 `ontap_san` 后端导入 `unmanaged_example_volume`：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

如果您的 LUN 映射到与 Kubernetes 节点 IQN 共享 IQN 的 igroup，如以下示例所示，您将收到错误： `LUN already mapped to initiator(s) in this group`。您需要删除启动器或取消映射 LUN 以导入卷。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economy 示例

以下示例演示如何为 `ontap-san-economy` 后端导入托管和非托管卷。

托管卷

当您导入托管卷时， Trident 将获得 FlexVol 的所有权并重命名它。当您从同一个 FlexVol 导入多个 LUN 时，您必须考虑此重命名。

以下示例将 `lun1`从FlexVol `toimport`导入为名为 `vol-managed-saneco`的受管卷：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

导入 `lun1`后，`Trident 将 FlexVol 重命名（例如，重命名为 `trident_lun_pool_xyz）。要从相同的 FlexVol 导入其他 LUN，请使用新的 FlexVol 名称：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

备注	`ontap-san-economy` 后端一次导入一个 LUN。您可以使用脚本自动执行多个导入。

非受管卷

当您导入非托管卷时，Trident 不拥有 FlexVol。但是，FlexVol 和 LUN 必须遵循 Trident 命名约定。

FlexVol 命名格式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX` 是后端配置中 `storagePrefix` 的值。默认值为 `trident`。
- `RANDOMSTRING` 是您选择的任何字符串。

LUN 命名要求

必须为 LUN 命名 `lun0`。

示例

如果您的 `storagePrefix`是 `xyz`，则 LUN 的完整路径为：

```
trident_lun_pool_xyz_randomstring/lun0
```

Element

Trident 支持 NetApp Element 软件和 NetApp HCI 卷导入，使用 `solidfire-san`驱动程序。

备注	Element 驱动程序支持重复的卷名。但是，如果存在重复的卷名，Trident 返回错误。作为一种解决方法，请克隆该卷，提供唯一的卷名，然后导入克隆的卷。

以下示例将在后端 `element_default`导入 `element-managed`卷。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident 支持使用 `azure-netapp-files`驱动程序进行卷导入。

备注	若要导入 Azure NetApp Files 卷，请通过其卷路径识别卷。卷路径是卷的导出路径在 `:/`之后的部分。例如，如果装载路径为 `10.0.0.2:/importvol1`，则卷路径为 `importvol1`。

以下示例将在后端 `azurenetappfiles_40517`上导入一个 `azure-netapp-files`卷，卷路径为 `importvol1`。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetApp Volumes

Trident 支持使用 `google-cloud-netapp-volumes`驱动程序进行卷导入。

以下示例在后端 `backend-tbc-gcnv1` 上导入卷 `testvoleasiaeast1`。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

当两个卷位于同一区域时，以下示例导入 `google-cloud-netapp-volumes`卷：

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

自定义卷名和标签

使用 Trident，您可以为创建的卷分配有意义的名称和标签。这有助于您识别卷并轻松映射到相应的 Kubernetes 资源 (PVC)。您还可以在后端级别定义模板，用于创建自定义卷名和自定义标签；您创建、导入或克隆的任何卷都将遵守模板。

开始之前

可定制的卷名和标签支持：

- 卷创建、导入和克隆操作。
- 对于 `ontap-nas-economy` 驱动程序，只有 Qtree 卷的名称符合名称模板。
- 对于 `ontap-san-economy` 驱动程序，只有 LUN 名称符合名称模板。

限制

- 自定义卷名仅与 ONTAP 本地驱动程序兼容。
- 只有 `ontap-san`、 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序才支持自定义标签。
- 自定义卷名称不适用于现有卷。

可定制卷名的关键行为

- 如果由于名称模板中的语法无效而发生故障，则后端创建将失败。但是，如果模板应用失败，将根据现有的命名约定命名卷。
- 当卷使用后端配置中的名称模板命名时，存储前缀不适用。任何所需的前缀值都可以直接添加到模板中。

使用名称模板和标签的后端配置示例

可以在根和/或池级别定义自定义名称模板。

根级别示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

Pool 级别示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

命名模板示例

示例 **1**：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

示例 **2**：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

需要考虑的要点

1. 对于卷导入，仅当现有卷具有特定格式的标签时，才会更新标签。例如：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 对于托管卷导入，卷名遵循后端定义中根级别定义的名称模板。
3. Trident 不支持使用带有存储前缀的切片操作符。
4. 如果模板没有产生唯一的卷名，Trident 将附加几个随机字符以创建唯一的卷名。
5. 如果 NAS 经济卷的自定义名称长度超过 64 个字符，Trident 将根据现有的命名约定命名卷。对于所有其他 ONTAP 驱动程序，如果卷名超过名称限制，则卷创建过程将失败。

跨命名空间共享 **NFS** 卷

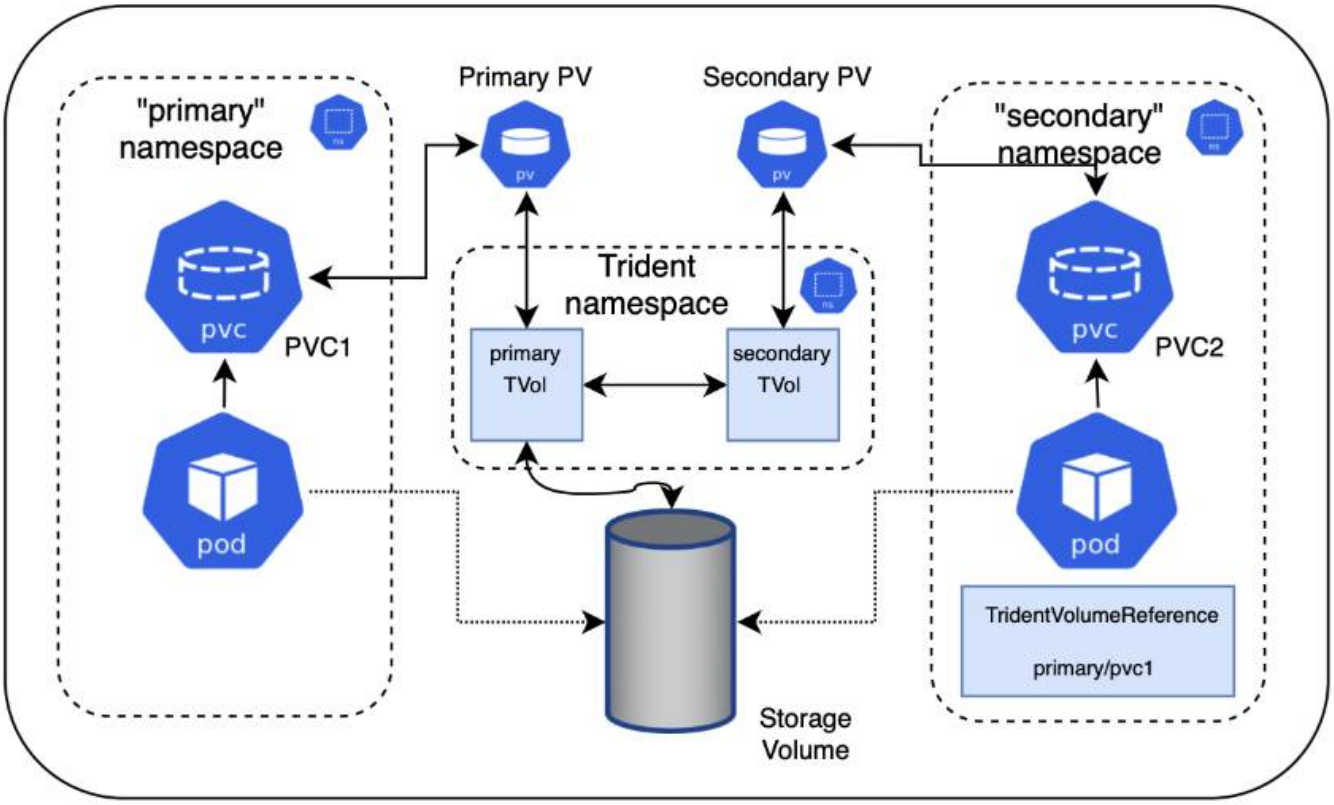
使用 Trident，您可以在主命名空间中创建卷，并在一个或多个辅助命名空间中共享它。

功能

TridentVolumeReference CR 允许您在一个或多个 Kubernetes 命名空间之间安全地共享 ReadWriteMany (RWX) NFS 卷。此 Kubernetes 原生解决方案具有以下优势：

- 多级访问控制，确保安全
- 适用于所有 Trident NFS 卷驱动程序
- 不依赖 tridentctl 或任何其他非本地 Kubernetes 功能

此图说明了两个 Kubernetes 命名空间之间的 NFS 卷共享。



快速入门

只需几个步骤即可设置 NFS 卷共享。

1 配置源 **PVC** 以共享卷

源命名空间所有者授予访问源 PVC 中的数据的权限。

2 授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 TridentVolumeReference CR 的权限。

3 在目标命名空间中创建 **TridentVolumeReference**

目标命名空间的所有者创建 TridentVolumeReference CR 以引用源 PVC。

4 在目标命名空间中创建从属 **PVC**

目标命名空间的所有者创建从属 PVC 以使用来自源 PVC 的数据源。

配置源和目标命名空间

为了确保安全性，跨命名空间共享需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。在每个步骤中指定用户角色。

步骤

1. 源命名空间所有者： 在源命名空间中创建 PVC (`pvc1`)，通过使用 `shareToNamespace`注解，授予与目标命名空间 (`namespace2`)共享的权限。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident 创建 PV 及其后端 NFS 存储卷。

备注	◦ 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。例如，`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。 ◦ 您可以使用 `*` 共享到所有命名空间。例如，`trident.netapp.io/shareToNamespace: *` ◦ 您可以随时更新 PVC 以包含 `shareToNamespace`注释。

2. *集群管理员：*确保已设置正确的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限。
3. 目标命名空间所有者： 在目标命名空间中创建引用源命名空间的 TridentVolumeReference CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 目标命名空间所有者： 在目标命名空间 (`namespace2`) 中创建一个 PVC (`pvc2`)，并使用 `shareFromPVC` 注解来指定源 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

备注	目标 PVC 的大小必须小于或等于源 PVC。

结果

Trident 读取目标 PVC 上的 `shareFromPVC`注释，并将目标 PV 创建为没有自己的存储资源的从属卷，该卷指向源 PV 并共享源 PV 存储资源。目标 PVC 和 PV 显示为正常绑定。

删除共享卷

您可以删除跨多个命名空间共享的卷。Trident 将删除对源命名空间上卷的访问权限，并保留对共享该卷的其他命名空间的访问权限。当引用该卷的所有命名空间都被删除时，Trident 会删除该卷。

使用 `tridentctl get` 查询从属卷

使用该[`tridentctl`实用程序，您可以运行该 `get`命令来获取从属卷。有关详细信息，请参见 `tridentctl` 命令和选项。

```
Usage:
  tridentctl get [option]
```

标记：

- `-h, --help`: 卷的帮助。
- `--parentOfSubordinate string`: 将查询限制为从属源卷。
- `--subordinateOf string`: 将查询限制为卷的下属。

限制

- Trident 无法阻止目标命名空间写入共享卷。您应该使用文件锁定或其他进程来防止覆盖共享卷数据。

- 您不能通过删除 `shareToNamespace`或 `shareFromNamespace`注释或删除 `TridentVolumeReference` CR 来撤销对源 PVC 的访问权限。要撤销访问权限，您必须删除从属 PVC。
- 无法在从属卷上执行快照、克隆和镜像。

了解更多信息

要了解有关跨命名空间卷访问的更多信息：

- 观看 "NetAppTV" 上的演示。

跨命名空间克隆卷

使用 Trident，您可以使用来自同一 Kubernetes 集群内不同命名空间的现有卷或卷快照创建新卷。

前提条件

在克隆卷之前，请确保源和目标后端的类型相同并且具有相同的存储类。

备注	只有 `ontap-san` 和 `ontap-nas` 存储驱动程序才支持跨命名空间克隆。不支持只读克隆。

快速入门

只需几步即可设置卷克隆。

1. 配置源 **PVC** 以克隆卷

源命名空间所有者授予访问源 PVC 中的数据的权限。

2. 授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 TridentVolumeReference CR 的权限。

3. 在目标命名空间中创建 **TridentVolumeReference**

目标命名空间的所有者创建 TridentVolumeReference CR 以引用源 PVC。

4. 在目标命名空间中创建克隆 **PVC**

目标命名空间的所有者创建 PVC 以从源命名空间克隆 PVC。

配置源和目标命名空间

为确保安全性，跨命名空间克隆卷需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。在每个步骤中指定用户角色。

步骤

1. 源命名空间所有者： 在源命名空间(`namespace1`中创建 `PVC(`pvc1`，通过使用 `cloneToNamespace`注解，授予与目标命名空间(`namespace2`共享的权限。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident 创建 PV 及其后端存储卷。

备注

- 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。例如，`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用 `*` 共享到所有命名空间。例如，`trident.netapp.io/cloneToNamespace: *`
- 您可以随时更新 PVC 以包含 `cloneToNamespace`注释。

2. 集群管理员： 确保已设置正确的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限(`namespace2`）。
3. 目标命名空间所有者： 在目标命名空间中创建引用源命名空间的 TridentVolumeReference CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 目标命名空间所有者： 在目标命名空间 (`namespace2`) 中创建一个 PVC (`pvc2`)，使用 `cloneFromPVC` 或 `cloneFromSnapshot`，以及 `cloneFromNamespace` 注解来指定源 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

限制

- 对于使用 ontap-nas-economy 驱动程序配置的 PVC，不支持只读克隆。

使用 SnapMirror 复制卷

Trident 支持一个集群上的源卷与对等集群上的目标卷之间的镜像关系，用于复制数据以进行灾难恢复。 您可以使用命名空间的自定义资源定义 (CRD)，称为 Trident Mirror Relationship (TMR) 来执行以下操作：

- 创建卷 (PVC) 之间的镜像关系
- 删除卷之间的镜像关系
- 中断镜像关系
- 在灾难情况（故障转移）期间提升辅助卷
- 执行应用程序从集群到集群的无损转换（在计划的故障转移或迁移期间）

复制先决条件

在开始之前，请确保满足以下先决条件：

ONTAP 集群

- **Trident**：Trident 版本 22.10 或更高版本必须存在于使用 ONTAP 作为后端的源和目标 Kubernetes 集群上。
- 许可证：必须在源和目标 ONTAP 集群上启用使用数据保护捆绑包的 ONTAP SnapMirror 异步许可证。请参阅 "ONTAP 中的 SnapMirror 许可概述"以获取更多信息。

  从 ONTAP 9.10.1 开始，所有许可证都以 NetApp 许可证文件 (NLF) 的形式交付，该文件是启用多个功能的单个文件。有关详细信息，请参见 "ONTAP One 附带的许可证"。

备注	仅支持 SnapMirror 异步保护。

对等

- 集群和 **SVM**：必须对等 ONTAP 存储后端。请参阅 "集群和 SVM 对等概述"以获取更多信息。

重要说明	确保在两个 ONTAP 集群之间的复制关系中使用的 SVM 名称是唯一的。

- **Trident** 和 **SVM**：对等远程 SVM 必须可用于目标集群上的 Trident。

支持的驱动程序

NetApp Trident 支持使用由以下驱动程序支持的存储类的 NetApp SnapMirror 技术进行卷复制： **`ontap-nas`: NFS** `ontap-san`: iSCSI **`ontap-san`: FC** `ontap-san`: NVMe/TCP（需要最低 ONTAP 版本 9.15.1）

备注	ASA r2 系统不支持使用 SnapMirror 进行卷复制。有关 ASA r2 系统的信息，请参见 "了解 ASA r2 存储系统"。

创建镜像 **PVC**

按照以下步骤并使用 CRD 示例在主卷和二级卷之间创建镜像关系。

步骤

1. 在主 Kubernetes 集群上执行以下步骤：
   a. 使用 `trident.netapp.io/replication: true` 参数创建 StorageClass 对象。

      示例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 使用先前创建的 StorageClass 创建 PVC。

示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. 使用本地信息创建 MirrorRelationship CR。

示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident 获取卷的内部信息和卷的当前数据保护 (DP) 状态，然后填充 MirrorRelationship 的 status 字段。

d. 获取 TridentMirrorRelationship CR 以获取 PVC 的内部名称和 SVM。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. 请在辅助 Kubernetes 集群上执行以下步骤：

   a. 使用 trident.netapp.io/replication: true 参数创建 StorageClass。

   示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

   b. 创建包含目标和源信息的 MirrorRelationship CR。

   示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Trident 将创建一个具有已配置关系策略名称（或 ONTAP 默认值）的 SnapMirror 关系并对其进行初始化。

c. 使用先前创建的 StorageClass 创建 PVC 以充当辅助（SnapMirror 目标）。

示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Trident 将检查 TridentMirrorRelationship CRD，如果关系不存在，则无法创建卷。如果存在此关系，Trident 将确保将新 FlexVol 卷放置到与 MirrorRelationship 中定义的远程 SVM 对等的 SVM 上。

卷复制状态

Trident 镜像关系（TMR）是表示 PVC 之间复制关系的一端的 CRD。目标 TMR 有一个状态，它告诉 Trident 所需的状态是什么。目标 TMR 具有以下状态：

- **Established**：本地 PVC 是镜像关系的目标卷，这是一种新的关系。
- **Promoted**：本地 PVC 是 ReadWrite 且可挂载的，目前没有镜像关系生效。
- 已重新建立：本地 PVC 是镜像关系的目标卷，此前也处于该镜像关系中。
  - 如果目标卷与源卷之间曾经存在关系，则必须使用重新建立的状态，因为它会覆盖目标卷的内容。
  - 如果卷以前未与源建立关系，则重新建立状态将失败。

在计划外故障转移期间提升辅助 **PVC**

在辅助 Kubernetes 集群上执行以下步骤：

- 将 TridentMirrorRelationship 的 *spec.state* 字段更新为 `promoted`。

在计划的故障转移期间推广辅助 **PVC**

在计划的故障转移（迁移）期间，请执行以下步骤来升级辅助 PVC：

步骤

1. 在主 Kubernetes 集群上，创建 PVC 的快照，并等待创建快照。

2. 在主 Kubernetes 集群上，创建 SnapshotInfo CR 以获取内部详细信息。

   示例

```
kind: SnapshotInfo
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  snapshot-name: csi-nas-snapshot
```

3. 在辅助 Kubernetes 集群上，将 *TridentMirrorRelationship* CR 的 *spec.state* 字段更新为 *promoted*，将 *spec.promotedSnapshotHandle* 更新为快照的 internalName。
4. 在辅助 Kubernetes 集群上，确认 TridentMirrorRelationship 的状态（status.state 字段）为 promoted。

故障转移后恢复镜像关系

在恢复镜像关系之前，请选择要作为新主要关系的一侧。

步骤

1. 在辅助 Kubernetes 集群上，请确保更新了 TridentMirrorRelationship 上的 *spec.remoteVolumeHandle* 字段的值。
2. 在辅助 Kubernetes 集群上，将 TridentMirrorRelationship 的 *spec.mirror* 字段更新为 `reestablished`。

其他操作

Trident 支持主卷和二级卷上的以下操作：

将主要 **PVC** 复制到新的次要 **PVC**

请确保已有一个主要 PVC 和一个次要 PVC。

步骤

1. 从已建立的辅助（目标）集群中删除 PersistentVolumeClaim 和 TridentMirrorRelationship CRD。
2. 从主（源）集群中删除 TridentMirrorRelationship CRD。
3. 在要建立的新辅助（目标） PVC 的主（源）集群上创建新的 TridentMirrorRelationship CRD。

调整镜像、主 **PVC** 或辅助 **PVC** 的大小

PVC 可以正常调整大小，如果数据量超过当前大小，ONTAP 将自动扩展任何目标 flexvols。

从 **PVC** 中删除复制

要删除复制，请在当前辅助卷上执行以下操作之一：

- 删除辅助 PVC 上的 MirrorRelationship。这会破坏复制关系。
- 或者，将 spec.state 字段更新为 *promoted*。

删除 **PVC**（先前已镜像）

Trident 检查复制的 PVC，并在尝试删除卷之前释放复制关系。

删除 **TMR**

删除镜像关系一侧的 TMR 会导致剩余的 TMR 在 Trident 完成删除之前过渡到 *promoted* 状态。如果选择删除的 TMR 已经处于 *promoted* 状态，则不存在现有的镜像关系，TMR 将被删除，Trident 会将本地 PVC 提升为 *ReadWrite*。此删除将释放 ONTAP 中本地卷的 SnapMirror 元数据。如果将来在镜像关系中使用此卷，则在创建新的镜像关系时，必须使用具有 *established* 卷复制状态的新 TMR。

ONTAP 联机时更新镜像关系

镜像关系建立后可以随时更新。您可以使用 `state: promoted`或 `state: reestablished`字段更新关系。将目标卷升级到常规 ReadWrite 卷时，您可以使用 *promotedSnapshotHandle* 指定要将当前卷还原到的特定快照。

ONTAP 离线时更新镜像关系

您可以使用 CRD 执行 SnapMirror 更新，而无需 Trident 直接连接到 ONTAP 集群。请参阅以下 TridentActionMirrorUpdate 格式示例：

示例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` 反映了 TridentActionMirrorUpdate CRD 的状态。它可以取自 *Succeeded*、*In Progress* 或 *Failed* 的值。

使用 **CSI** 拓扑

Trident 可以通过使用 "CSI 拓扑功能"选择性地创建卷并将其附加到 Kubernetes 集群中的节点。

概述

使用 CSI 拓扑功能，可以根据区域和可用性区域将对卷的访问限制为节点的子集。如今，云提供商使 Kubernetes 管理员能够生成基于区域的节点。节点可以位于一个区域内的不同可用区，也可以位于不同的区域。为了便于在多区域架构中为工作负载配置卷，Trident 使用了 CSI 拓扑。

提示	详细了解 CSI Topology 功能 "此处"。

Kubernetes 提供两种独特的卷绑定模式：

- 当 `VolumeBindingMode`设置为 `Immediate`时，Trident 会在没有任何拓扑感知的情况下创建卷。创建 PVC 时会处理卷绑定和动态配置。这是默认 `VolumeBindingMode`，适用于不强制执行拓扑约束的集群。创建持久卷时不依赖于请求 Pod 的调度要求。
- 将 `VolumeBindingMode` 设置为 `WaitForFirstConsumer` 时，PVC 的 Persistent Volume 的创建和绑定会被延迟，直到使用该 PVC 的 pod 被调度和创建。通过这种方式，创建的卷可以满足拓扑要求所实施的调度约束。

备注	The `WaitForFirstConsumer` 绑定模式不需要拓扑标签。这可以独立于 CSI 拓扑功能使用。

您需要什么

要使用 CSI 拓扑，您需要以下内容：

- 运行 "支持的 Kubernetes 版本" 的 Kubernetes 集群

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 集群中的节点应具有引入拓扑感知(`topology.kubernetes.io/region`和 `topology.kubernetes.io/zone`)的标签。在安装 Trident 之前，这些标签*应该存在于集群中的节点上*，以便 Trident 具有拓扑感知能力。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

步骤 **1**：创建可感知拓扑的后端

Trident 存储后端可以设计为基于可用性区域有选择地配置卷。每个后端都可以携带一个可选 `supportedTopologies`块，该块表示受支持的区域和地区列表。对于使用此类后端的 StorageClasses，只有在支持的地区/区域中调度的应用程序请求时，才会创建卷。

以下是后端定义示例：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

备注	`supportedTopologies` 用于为每个后端提供区域和可用区的列表。这些区域和可用区表示可以在 StorageClass 中提供的允许值列表。对于包含后端提供的区域和可用区子集的 StorageClasses，Trident 会在后端上创建卷。

您也可以为每个存储池定义 `supportedTopologies`。请参见以下示例：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

在此示例中， `region` 和 `zone` 标签代表存储池的位置。 `topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone` 指示可以从哪里使用存储池。

步骤 2：定义拓扑感知的 **StorageClasses**

根据提供给集群中节点的拓扑标签，StorageClasses 可以定义为包含拓扑信息。这将确定用作 PVC 请求候选项的存储池，以及可以使用 Trident 配置的卷的节点子集。

请参见以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

在上面提供的 StorageClass 定义中， `volumeBindingMode` 设置为 `WaitForFirstConsumer`。使用此 StorageClass 请求的 PVC 在 pod 中引用之前不会被执行。并且， `allowedTopologies` 提供要使用的区域和地区。 `netapp-san-us-east1` StorageClass 在上面定义的 `san-backend-us-east1` 后端上创建 PVC。

步骤 **3**：创建和使用 **PVC**

创建 StorageClass 并将其映射到后端后，您现在可以创建 PVC。

请参见以下示例 `spec`：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

使用此清单创建 PVC 将导致以下结果：

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

要使 Trident 创建卷并将其绑定到 PVC，请在 Pod 中使用 PVC。请参见以下示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

这个 podSpec 指示 Kubernetes 在 `us-east1` 区域中的节点上调度 pod，并从 `us-east1-a` 或 `us-east1-b` 可用区中的任何节点中进行选择。

请参阅以下输出：

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

更新后端以包括 `supportedTopologies`

可以更新预先存在的后端，以包含 `supportedTopologies`使用 `tridentctl backend update`的列表。这不会影响已配置的卷，仅用于后续 PVC。

查找更多信息

- "管理容器的资源"
- "nodeSelector"
- "亲和性和反亲和性"
- "污点和容忍"

使用快照

持久卷 (PV) 的 Kubernetes 卷快照支持卷的时间点副本。您可以创建使用 Trident 创建的卷的快照，导入在 Trident 外部创建的快照，从现有快照创建新卷，并从快照中恢复卷数据。

概述

`ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `azure-netapp-files` 和 `google-cloud-netapp-volumes` 驱动程序支持卷快照。

开始之前

必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，请参阅 [部署卷快照控制器]。

备注	如果在 GKE 环境中创建按需卷快照，请勿创建快照控制器。GKE 使用内置的隐藏快照控制器。

创建卷快照

步骤

1. 创建一个 `VolumeSnapshotClass`。有关详细信息，请参见 "VolumeSnapshotClass"。
   - `driver` 指向 Trident CSI 驱动程序。
   - `deletionPolicy` 可以为 `Delete`或 `Retain`。当设置为 `Retain`时，即使删除 `VolumeSnapshot` 对象，也会保留存储集群上的底层物理快照。

   示例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   deletionPolicy: Delete
   ```

2. 创建现有 PVC 的快照。

   示例

   - 此示例创建现有 PVC 的快照。

   ```
   cat snap.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: pvc1-snap
   spec:
     volumeSnapshotClassName: csi-snapclass
     source:
       persistentVolumeClaimName: pvc1
   ```

   - 此示例为名为 `pvc1`的 `PVC` 创建卷快照对象，快照的名称设置为 `pvc1-snap`。VolumeSnapshot 类似于 PVC，并与表示实际快照的 `VolumeSnapshotContent`对象相关联。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 您可以通过描述 `VolumeSnapshotContent`对象来识别 `pvc1-snap` VolumeSnapshot。`Snapshot Content Name`标识为此快照提供服务的 VolumeSnapshotContent 对象。 `Ready To Use`参数表示快照可用于创建新的 PVC。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

从卷快照创建 **PVC**

您可以使用 `dataSource`创建 PVC，使用名为 `<pvc-name>`的 VolumeSnapshot 作为数据源。创建 PVC 后，它可以连接到 pod 并像任何其他 PVC 一样使用。

警告	PVC 将在与源卷相同的后端创建。请参见 "KB：无法在备用后端从 Trident PVC 快照创建 PVC"。

以下示例使用 `pvc1-snap` 作为数据源创建 PVC。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

导入卷快照

Trident 支持 "Kubernetes 预置 Snapshot 流程" 使集群管理员能够创建 `VolumeSnapshotContent` 对象并导入在 Trident 外部创建的快照。

开始之前

Trident 必须已创建或导入快照的父卷。

步骤

1. 集群管理员： 创建引用后端快照的 `VolumeSnapshotContent` 对象。这将启动 Trident 中的快照工作流。
   - 将 `annotations` 中的后端快照的名称指定为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 在 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`中指定 `snapshotHandle`。这是外部快照程序在 `ListSnapshots`调用中提供给 Trident 的唯一信息。

     备注 由于 CR 命名限制， `<volumeSnapshotContentName>` 无法始终匹配后端快照名称。

     示例

     以下示例创建一个 `VolumeSnapshotContent`对象，该对象引用后端快照 `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. 集群管理员： 创建引用 VolumeSnapshot`对象的 `VolumeSnapshotContent CR。这将请求在指定命名空间中使用 `VolumeSnapshot`的访问权限。

   示例

   以下示例创建了一个 VolumeSnapshot`名为 `import-snap`的 CR，并引用了名为 `import-snap-content`的 `VolumeSnapshotContent。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. 内部处理（无需执行任何操作）： 外部快照器识别新创建的 VolumeSnapshotContent`并运行 `ListSnapshots`调用。Trident 创建 `TridentSnapshot。
   - 外部快照器将 VolumeSnapshotContent 设置为 readyToUse，将 VolumeSnapshot 设置为 true。
   - Trident 返回 readyToUse=true。
4. 任何用户： 创建 PersistentVolumeClaim`以引用新 `VolumeSnapshot，其中 spec.dataSource（或 spec.dataSourceRef）名称是 `VolumeSnapshot`名称。

   示例

以下示例创建了一个 PVC，引用名为 `VolumeSnapshot`的 `import-snap。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

使用快照恢复卷数据

默认情况下隐藏 snapshot 目录，以促进使用 `ontap-nas`和 `ontap-nas-economy`驱动程序配置的卷的最大兼容性。启用 `.snapshot`目录以直接从快照恢复数据。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

备注	还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 `TridentActionSnapshotRestore` (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

Trident 支持在 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `google-cloud-netapp-volumes` 和 `solidfire-san` 驱动程序上还原快照。

开始之前

您必须具有绑定的 PVC 和可用的卷快照。

- 确认 PVC 状态已绑定。

```
kubectl get pvc
```

- 验证卷快照是否可以使用。

```
kubectl get vs
```

步骤

1. 创建 TASR CR。此示例为 PVC `pvc1` 和卷快照 `pvc1-snapshot`创建 CR。

   备注 | TASR CR 必须位于存在 PVC 和 VS 的命名空间中。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. 应用 CR 从快照中恢复。此示例从快照中还原 `pvc1`。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

结果

Trident 从快照中恢复数据。您可以验证快照还原状态：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

备注

- 在大多数情况下，出现故障时，Trident 不会自动重试此操作。您需要再次执行此操作。
- 没有管理员访问权的 Kubernetes 用户可能需要管理员授予权限才能在其应用程序命名空间中创建 TASR CR。

删除具有关联快照的 PV

删除具有关联快照的永久卷时，相应的 Trident 卷将更新为"删除状态"。删除卷快照以删除 Trident 卷。

部署卷快照控制器

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

步骤

1. 创建卷快照 CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

备注	如有必要，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`到您的命名空间。

相关链接

- "卷快照"
- "VolumeSnapshotClass"

处理卷组快照

持久卷（PV）的 Kubernetes 卷组快照 NetApp Trident 提供了创建多个卷快照（一组卷快照）的功能。此卷组快照表示在同一时间点拍摄的多个卷的副本。

备注	VolumeGroupSnapshot 是 Kubernetes 中带有 beta API 的 beta 功能。Kubernetes 1.32 是 VolumeGroupSnapshot 所需的最低版本。

创建卷组快照

以下存储驱动程序支持卷组快照：

- `ontap-san` 驱动程序 - 仅适用于 iSCSI 和 FC 协议，不适用于 NVMe/TCP 协议。
- `ontap-san-economy` - 仅适用于 iSCSI 协议。
- `ontap-nas`

备注	NetApp ASA r2 或 AFX 存储系统不支持卷组快照。

开始之前

- 请确保您的 Kubernetes 版本为 K8s 1.32 或更高版本。
- 必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

  如果您的 Kubernetes 发行版不包括外部快照控制器和 CRD，请参阅 [部署卷快照控制器]。

备注	如果在 GKE 环境中创建按需卷组快照，请勿创建快照控制器。GKE 使用内置的隐藏快照控制器。

- 在快照控制器 YAML 中，将 `CSIVolumeGroupSnapshot` 功能门设置为 'true'，以确保启用卷组快照。
- 在创建卷组快照之前创建所需的卷组快照类。
- 确保所有 PVC/卷都在同一个 SVM 上，以便能够创建 VolumeGroupSnapshot。

步骤

- 在创建 VolumeGroupSnapshot 之前，先创建 VolumeGroupSnapshotClass。有关详细信息，请参阅 "VolumeGroupSnapshotClass"。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- 使用现有存储类创建具有所需标签的 PVC，或将这些标签添加到现有 PVC。

  以下示例使用 `pvc1-group-snap` 作为数据源和标签 `consistentGroupSnapshot: groupA` 创建 PVC。根据您的要求定义标签键和值。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- 创建具有相同标签 (`consistentGroupSnapshot: groupA` 的 VolumeGroupSnapshot，该标签在 PVC 中指定。

  此示例创建卷组快照：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

使用组快照恢复卷数据

您可以使用作为 Volume Group Snapshot 一部分创建的单个快照来恢复单个 Persistent Volume。您无法将 Volume Group Snapshot 作为一个单位进行恢复。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

备注	还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 `TridentActionSnapshotRestore` (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

有关详细信息，请参见 "从快照就地还原卷"。

删除具有关联组快照的 **PV**

删除组卷快照时：

- 您可以整体删除 VolumeGroupSnapshots，而不是删除组中的单个快照。
- 如果在存在该 PersistentVolume 的快照时删除 PersistentVolumes，Trident 会将该卷移动到“deleting”状态，因为必须先删除快照，才能安全地删除该卷。
- 如果已使用分组快照创建克隆，然后要删除该组，则将开始克隆拆分操作，并且在完成拆分之前无法删除该组。

部署卷快照控制器

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

步骤

1. 创建卷快照 CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.
yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

备注	如有必要，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`到您的命名空间。

相关链接

- "VolumeGroupSnapshotClass"
- "卷快照"