Trident 25.06 文档

Trident

NetApp
March 04, 2026

目录

Trident 25.06 文档

发行说明

什么是新的

发行说明提供了有关NetApp Trident最新版本的新功能、增强功能和错误修复的信息。

> 这 `tridentctl`安装程序 zip 文件中提供的 Linux 二进制文件是经过测试和支持的版本。请注意，`macos`提供的二进制文件 `/extras`压缩文件中的部分内容未经测试或支持。

25.06.2 中的新增功能

“新增功能”摘要提供了有关Trident和Trident Protect 版本增强功能、修复程序和弃用项的详细信息。

Trident

修复

- **Kubernetes**：修复了从 Kubernetes 节点分离卷时发现不正确的 iSCSI 设备的严重问题。

25.06.1 中的变更

Trident

> 对于使用SolidFire的客户，请不要升级到 25.06.1，因为取消发布卷时存在已知问题。25.06.2 即将发布以解决此问题。

修复

- **Kubernetes**：
  - 修复了在从子系统取消映射之前未检查 NQN 的问题。
  - 修复了多次尝试关闭 LUKS 设备导致无法分离卷的问题。
  - 修复了当设备路径自创建以来发生变化时 iSCSI 卷取消暂存的问题。
  - 阻止跨存储类的卷克隆。
- **OpenShift**：修复了 OCP 4.19 中 iSCSI 节点准备失败的问题。
- 增加了使用SolidFire后端克隆卷时的超时时间（"第1008期"）。

25.06 中的变化

Trident

增强功能

- **Kubernetes**：
  - 新增对 CSI 卷组快照的支持 `v1beta1`适用于ONTAP的卷组快照 Kubernetes API - SAN iSCSI 驱动程序。看"使用卷组快照"。

VolumeGroupSnapshot 是 Kubernetes 中的一项测试版功能，其 API 也处于测试版阶段。 VolumeGroupSnapshot 所需的最低 Kubernetes 版本为 1.32。

- 除了 iSCSI 之外，还增加了对ONTAP ASA r2 的 NVMe/TCP 支持。看link:"ONTAP SAN 配置选项和示例"。
- 为ONTAP-NAS 和ONTAP-NAS-Economy 卷添加了安全的 SMB 支持。现在可以将 Active Directory 用户和组与 SMB 卷一起使用，以增强安全性。看"启用安全的 SMB"。
- 增强Trident节点并发性，以提高 iSCSI 卷节点操作的可扩展性。
- 额外 `--allow-discards`打开 LUKS 卷时，允许执行 discard/TRIM 命令以回收空间。
- 提高格式化 LUKS 加密卷时的性能。
- 增强了对失败但部分格式化的 LUKS 设备的 LUKS 清理功能。
- 增强了Trident节点对NVMe卷挂载和分离的幂等性。
- 额外 `internalID`字段到ONTAP -SAN-Economy 驱动程序的Trident卷配置。
- 为 NVMe 后端添加了对SnapMirror卷复制的支持。看"使用SnapMirror复制卷"。

实验性增强

不可用于生产环境。

- [技术预览] 通过以下方式启用并发的Trident控制器操作 `--enable-concurrency`功能标志。这样一来，控制器操作就可以并行运行，从而提高繁忙或大型环境的性能。

  此功能为实验性功能，目前仅支持使用ONTAP-SAN 驱动程序（iSCSI 和 FCP 协议）的有限并行工作流程。

- 【技术预览】为 ANF 驱动程序添加了手动 QOS 支持。

修复

- **Kubernetes**：
  - 修复了 CSI NodeExpandVolume 的一个问题，即当底层 SCSI 磁盘不可用时，多路径设备的大小可能会不一致。
  - 修复了ONTAP-NAS 和ONTAP-NAS-Economy 驱动程序的重复导出策略清理失败的问题。
  - 修复了 GCNV 卷默认使用 NFSv3 的问题 `nfsMountOptions`未设置；现在同时支持 NFSv3 和 NFSv4 协议。如果 `nfsMountOptions`如果未提供，则将使用主机的默认 NFS 版本（NFSv3 或 NFSv4）。
  - 修复了使用 Kustomize 安装Trident时出现的部署问题（"第831期"）。
  - 修复了从快照创建的 PVC 缺少导出策略的问题（"第1016期"）。
  - 修复了 ANF 卷大小无法自动按 1 GiB 增量对齐的问题。
  - 修复了使用 Bottlerocket 时 NFSv3 出现的问题。
- 修复了使用SolidFire后端克隆卷时出现的超时问题（"第1008期"）。
- 修复了ONTAP-NAS-Economy 卷在调整大小失败的情况下仍可扩展至 300 TB 的问题。

- 修复了使用ONTAP REST API 时克隆拆分操作同步执行的问题。

弃用：

- **Kubernetes**：已将最低支持的 Kubernetes 版本更新至 v1.27。

Trident保护

NetApp Trident Protect 提供高级应用程序数据管理功能，增强了由NetApp ONTAP存储系统和NetApp Trident CSI 存储供应器支持的有状态 Kubernetes 应用程序的功能和可用性。

增强功能

- 提高了恢复速度，并提供了更频繁执行完整备份的选项。
- 改进了应用程序定义的粒度，并可通过组版本类型 (GVK) 筛选进行选择性恢复。
- 使用 AppMirrorRelationship (AMR) 和NetApp SnapMirror时，可实现高效的重新同步和反向复制，从而避免完全 PVC 复制。
- 新增了使用 EKS Pod Identity 创建 AppVault 存储桶的功能，无需为 EKS 集群指定存储桶凭据的密钥。
- 增加了在恢复命名空间时跳过恢复标签和注释的功能（如果需要）。
- AppMirrorRelationship (AMR) 现在将检查源 PVC 扩展，并根据需要对目标 PVC 执行相应的扩展。

修复

- 修复了之前快照的快照注释值被应用到新快照的错误。所有快照注释现已正确应用。
- 默认情况下，为数据移动器加密（Kopia / Restic）定义了一个密钥，如果未定义则不会定义。
- 为 S3 应用库创建添加了改进的验证和错误消息。
- AppMirrorRelationship (AMR) 现在只复制处于绑定状态的 PV，以避免复制失败。
- 修复了在具有大量备份的 AppVault 上获取 AppVaultContent 时显示错误的问题。
- 为避免故障，KubeVirt VMSnapshots 被排除在恢复和故障转移操作之外。
- 修复了 Kopia 的一个问题，即由于 Kopia 的默认保留计划覆盖了用户在计划中设置的内容，导致快照被过早删除。

25.02.1 中的变更

Trident

修复

- **Kubernetes**：
  - 修复了 trident-operator 在使用非默认镜像仓库时 sidecar 镜像名称和版本填充不正确的问题 ("第983期" ）。
  - 修复了ONTAP故障转移恢复期间多路径会话无法恢复的问题（"第961期" ）。

25.02 中的变化

从Trident 25.02 开始，“新增功能”摘要提供了Trident和Trident Protect 版本的增强功能、修复和弃用详情。

Trident

增强功能

- **Kubernetes**：
  - 增加了对ONTAP ASA r2 的 iSCSI 支持。
  - 增加了对在非正常节点关闭情况下强制分离ONTAP-NAS 卷的支持。新的ONTAP-NAS 卷现在将使用由Trident管理的按卷导出策略。为现有卷提供升级路径，使其在取消发布时过渡到新的导出策略模型，而不会影响正在运行的工作负载。
  - 添加了 cloneFromSnapshot 注解。
  - 增加了对跨命名空间卷克隆的支持。
  - 增强 iSCSI 自愈扫描修复功能，可通过精确的主机、通道、目标和 LUN ID 启动重新扫描。
  - 增加了对 Kubernetes 1.32 的支持。
- **OpenShift**：
  - 为 ROSA 集群上的 RHCOS 添加了自动 iSCSI 节点准备支持。
  - 为ONTAP驱动程序添加了对 OpenShift 虚拟化的支持。
- ONTAP-SAN驱动程序新增了光纤通道支持。
- 新增NVMe LUKS支持。
- 所有基础图像均已切换为临时图像。
- 添加了 iSCSI 连接状态发现和日志记录功能，用于记录 iSCSI 会话应该登录但实际未登录的情况（"第961期"）。
- 添加了对使用 google-cloud-netapp-volumes 驱动程序的 SMB 卷的支持。
- 增加了对ONTAP卷在删除时跳过恢复队列的支持。
- 新增了使用 SHA 值而非标签覆盖默认图像的功能。
- 为 tridentctl 安装程序添加了 image-pull-secrets 标志。

修复

- **Kubernetes**：
  - 修复了自动导出策略中缺失的节点 IP 地址（"第965期"）。
  - 修复了ONTAP-NAS-Economy 的自动导出策略过早切换到按卷策略的问题。
  - 修复后端配置凭证，以支持所有可用的 AWS ARN 分区（"第913期"）。
  - 在Trident操作符中添加了禁用自动配置器协调的选项（"第924期"）。
  - 为 csi-resizer 容器添加了 securityContext（"第976期"）。

Trident保护

NetApp Trident Protect 提供高级应用程序数据管理功能，增强了由NetApp ONTAP存储系统和NetApp Trident CSI 存储供应器支持的有状态 Kubernetes 应用程序的功能和可用性。

增强功能

- 为 KubeVirt / OpenShift 虚拟化虚拟机添加了备份和恢复支持，支持 volumeMode: File 和 volumeMode: Block（原始设备）存储。此支持与所有Trident驱动程序兼容，并且在使用NetApp SnapMirror和Trident Protect 复制存储时增强了现有的保护功能。
- 为 Kubevirt 环境增加了在应用程序级别控制冻结行为的功能。
- 增加了对配置AutoSupport代理连接的支持。
- 增加了为数据移动加密（Kopia / Restic）定义密钥的功能。
- 增加了手动运行执行钩子的功能。
- 增加了在Trident Protect 安装过程中配置安全上下文约束 (SCC) 的功能。
- 增加了在Trident Protect 安装过程中配置 nodeSelector 的支持。
- 为 AppVault 对象添加了对 HTTP / HTTPS 出口代理的支持。
- 扩展资源过滤器，以排除集群范围的资源。
- 为 S3 AppVault 凭证添加了对 AWS 会话令牌的支持。
- 增加了对快照执行钩子之后资源收集的支持。

修复

- 改进了临时卷的管理，使其跳过ONTAP卷恢复队列。
- SCC 注释现已恢复为原始值。
- 支持并行操作，提高了恢复效率。
- 增强了对大型应用程序执行钩子超时的支持。

24.10.1 中的变更

增强功能

- **Kubernetes**：新增对 Kubernetes 1.32 的支持。
- 添加了 iSCSI 连接状态发现和日志记录功能，用于记录 iSCSI 会话应该登录但实际未登录的情况（"第961期"）。

修复

- 修复了自动导出策略中缺失的节点 IP 地址（"第965期"）。
- 修复了ONTAP-NAS-Economy 的自动导出策略过早切换到按卷策略的问题。
- 更新了Trident和 Trident-ASUP 依赖项，以解决 CVE-2024-45337 和 CVE-2024-45310。
- 在 iSCSI 自愈期间，移除间歇性不健康的非 CHAP 门户的注销操作 ("第961期")。

24.10 中的变化

增强功能

- Google Cloud NetApp Volumes驱动程序现已正式推出，适用于 NFS 卷，并支持区域感知配置。
- GCP Workload Identity 将用作Google Cloud NetApp Volumes与 GKE 的云身份。
- 额外 `formatOptions`向ONTAP-SAN 和ONTAP-SAN-Economy 驱动程序添加配置参数，允许用户指定 LUN 格式选项。
- 将Azure NetApp Files卷的最小大小减少到 50 GiB。 Azure 新的最小系统规模预计将于 11 月正式推出。
- 额外 `denyNewVolumePools`配置参数，用于将ONTAP-NAS-Economy 和ONTAP-SAN-Economy 驱动程序限制为预先存在的 Flexvol 池。
- 增加了对所有ONTAP驱动程序中 SVM 聚合的添加、删除或重命名的检测。
- 向 LUKS LUN 添加了 18 MiB 开销，以确保报告的 PVC 大小可用。
- 改进了ONTAP-SAN 和ONTAP-SAN-Economy 节点阶段和取消阶段错误处理，允许取消阶段操作在阶段失败后移除设备。
- 添加了自定义角色生成器，允许客户在ONTAP中为Trident创建极简角色。
- 增加了额外的故障排除日志记录 `lsscsi`（"第792期"）。

Kubernetes

- 为 Kubernetes 原生工作流添加了新的Trident功能：
  - 数据保护
  - 数据迁移
  - 灾难恢复
  - 应用移动性

    "了解更多关于Trident Protect的信息"。
- 新增标志 `--k8s-api-qps`指示安装程序设置Trident用于与 Kubernetes API 服务器通信的 QPS 值。
- 额外 `--node-prep`向安装程序发出信号，以便自动管理 Kubernetes 集群节点上的存储协议依赖项。已测试并验证与 Amazon Linux 2023 iSCSI 存储协议的兼容性
- 增加了对ONTAP-NAS-Economy 卷在非正常节点关闭场景下强制分离的支持。
- 新的ONTAP-NAS-Economy NFS 卷在使用时将采用基于 qtree 的导出策略 `autoExportPolicy`后端选项。Qtree 仅在发布时才会映射到节点限制性导出策略，以提高访问控制和安全性。当Trident从所有节点取消发布卷时，现有的 qtree 将切换到新的导出策略模型，这样就不会影响正在进行的工作负载。
- 增加了对 Kubernetes 1.31 的支持。

实验性增强

- ONTAP-SAN驱动程序新增了对光纤通道支持的技术预览。

修复

- **Kubernetes**：
  - 修复了 Rancher 准入 webhook 阻止Trident Helm 安装的问题（"第839期"）。
  - 固定 Helm Chart 值中的亲和力键（"第898期"）。
  - 已修复 tridentControllerPluginNodeSelector/tridentNodePluginNodeSelector 无法处理“true”值（"第899期"）。
  - 克隆过程中创建的已删除临时快照（"第901期"）。
- 新增对 Windows Server 2019 的支持。
- 修复了Trident仓库中的 `go mod tidy`（"第767期"）。

弃用

- **Kubernetes:**
  - 已将支持的最低 Kubernetes 版本更新至 1.25。
  - 已移除对 POD 安全策略的支持。

产品品牌重塑

从 24.10 版本开始， Astra Trident更名为Trident （Netapp Trident）。此次品牌重塑不会影响Trident的任何功能、支持的平台或互操作性。

24.06 中的变化

增强功能

- 重要提示： `limitVolumeSize`该参数现在限制ONTAP经济型驱动程序中的 qtree/LUN 大小。使用新的 `limitVolumePoolSize`用于控制这些驱动程序中 Flexvol 大小的参数。 （"第341期"）。
- 增加了 iSCSI 自愈功能，以便在使用已弃用的 igroup 时通过精确的 LUN ID 启动 SCSI 扫描 ("第883期"）。
- 增加了对卷克隆和调整大小操作的支持，即使后端处于挂起模式也允许执行这些操作。
- 增加了将Trident控制器的用户配置日志设置传播到Trident节点 pod 的功能。
- Trident增加了对ONTAP 9.15.1 及更高版本默认使用 REST 而不是 ONTAPI (ZAPI) 的支持。
- ONTAP存储后端新增了对自定义卷名称和元数据的支持，用于创建新的持久卷。
- 增强了 `azure-netapp-files`（ANF）驱动程序，当 NFS 挂载选项设置为使用 NFS 版本 4.x 时，默认自动启用快照目录。
- 增加了对 NFS 卷的 Bottlerocket 支持。
- 新增对Google Cloud NetApp Volumes的技术预览支持。

Kubernetes

- 增加了对 Kubernetes 1.30 的支持。
- 为Trident DaemonSet 添加了在启动时清理僵尸挂载点和残留跟踪文件的功能（"第883期"）。

- 添加了PVC注释 `trident.netapp.io/luksEncryption`用于动态导入 LUKS 卷（"第849期"）。
- 为 ANF 驱动程序添加了拓扑感知功能。
- 新增对 Windows Server 2022 节点的支持。

修复

- 修复了由于过期交易导致的Trident安装失败问题。
- 修复了 tridentctl 忽略来自 Kubernetes 的警告消息的问题（"第892期"）。
- 更换了Trident控制器 `SecurityContextConstraint`优先 `0`（"第887期"）。
- ONTAP驱动程序现在接受小于 20 MiB 的卷大小（"问题[#885"）。
- 修复了Trident，以防止在ONTAP -SAN 驱动程序调整大小操作期间FlexVol卷缩小。
- 修复了 NFS v4.1 中 ANF 卷导入失败的问题。

24.02 中的变化

增强功能

- 新增对云身份的支持。
  - 将使用 AKS 和 ANF - Azure 工作负载标识作为云标识。
  - EKS 与 FSxN - 将使用 AWS IAM 角色作为云身份。
- 增加了从 EKS 控制台将Trident作为插件安装到 EKS 集群的支持。
- 增加了配置和禁用 iSCSI 自愈功能 ("第864期"）。
- 为ONTAP驱动程序添加了Amazon FSx特性，以启用与 AWS IAM 和 SecretsManager 的集成，并使Trident能够删除带有备份的 FSx 卷 ("第453期"）。

Kubernetes

- 新增对 Kubernetes 1.29 的支持。

修复

- 修复了未启用 ACP 时的 ACP 警告消息（"第866期"）。
- 在ONTAP驱动程序中，当克隆与快照关联时，在删除快照期间执行克隆拆分之前，增加了 10 秒的延迟。

弃用

- 从多平台镜像清单中移除了 in-toto 认证框架。

23.10 中的变化

修复

- 对于 ontap-nas 和 ontap-nas-flexgroup 存储驱动程序，如果新请求的大小小于总卷大小，则固定卷扩展（"第834期"）。

- 固定卷大小，以便在导入 ontap-nas 和 ontap-nas-flexgroup 存储驱动程序时仅显示卷的可用大小（"第722期"）。
- 修复了ONTAP -NAS-Economy 的FlexVol名称转换问题。
- 修复了 Windows 节点重启后Trident初始化出现的问题。

增强功能

Kubernetes

增加了对 Kubernetes 1.28 的支持。

Trident

- 增加了对使用 Azure 托管标识 (AMI) 和 azure-netapp-files 存储驱动程序的支持。
- 为ONTAP-SAN 驱动程序添加了对 NVMe over TCP 的支持。
- 增加了当后端被用户设置为暂停状态时暂停卷配置的功能（"第558期"）。

23.07.1 中的变更

*Kubernetes：*修复了守护进程集删除问题，以支持零停机升级（"第740期"）。

23.07 中的变化

修复

Kubernetes

- 修复了Trident升级，使其忽略处于终止状态的旧 pod（"第740期"）。
- 为“transient-trident-version-pod”定义添加了容忍度（"第795期"）。

Trident

- 修复了 ONTAPI (ZAPI) 请求，以确保在获取 LUN 属性时查询 LUN 序列号，从而在节点暂存操作期间识别和修复幽灵 iSCSI 设备。
- 修复了存储驱动程序代码中的错误处理（"第816期"）。
- 修复了使用带有 use-rest=true 的ONTAP驱动程序时配额调整的问题。
- 修复了 ontap-san-economy 中的 LUN 克隆创建问题。
- 恢复发布信息字段 `rawDevicePath`到 `devicePath`添加了用于填充和恢复（某些情况下）的逻辑 `devicePath`场地。

增强功能

Kubernetes

- 新增对导入预配置快照的支持。
- 最小化部署和守护进程集 Linux 权限（"第817期"）。

Trident

- 不再报告“在线”卷和快照的状态字段。
- 如果ONTAP后端离线，则更新后端状态（"第 801 期"，"#543"）。
- LUN 序列号始终在 ControllerVolumePublish 工作流程中检索和发布。
- 增加了额外的逻辑来验证 iSCSI 多路径设备的序列号和大小。
- 对 iSCSI 卷进行额外验证，以确保正确的多路径设备已取消暂存。

实验增强

为ONTAP-SAN 驱动程序添加了对 NVMe over TCP 的技术预览支持。

文档

在组织结构和格式方面都进行了许多改进。

弃用

Kubernetes

- 已移除对 v1beta1 快照的支持。
- 移除对 CSI 之前的卷和存储类的支持。
- 已将支持的最低 Kubernetes 版本更新至 1.22。

23.04 中的变化

> 仅当 Kubernetes 版本启用了非优雅节点关闭功能门时，才支持对ONTAP-SAN-* 卷强制分离。必须在安装时使用以下方式启用强制分离： `--enable-force-detach` Trident安装程序标志。

修复

- 修复了Trident Operator 在规范中指定时使用 IPv6 localhost 进行安装的问题。
- 修复了Trident Operator 集群角色权限，使其与捆绑包权限保持同步（"第799期"）。
- 修复了在 RWX 模式下将原始块卷附加到多个节点的问题。
- 修复了FlexGroup克隆支持和 SMB 卷的卷导入问题。
- 修复了Trident控制器无法立即关闭的问题（"第811期"）。
- 添加了修复程序，用于列出与使用 ontap-san-* 驱动程序配置的指定 LUN 关联的所有 igroup 名称。
- 添加了允许外部进程运行完成的修复程序。
- 修复了 s390 架构的编译错误（"第537期"）。
- 修复了卷挂载操作期间日志级别不正确的问题"第781期"）。
- 修复了潜在的类型断言错误（"第802期"）。

增强功能

- Kubernetes：
  - 新增对 Kubernetes 1.27 的支持。
  - 新增对导入 LUKS 卷的支持。
  - 增加了对 ReadWriteOncePod PVC 访问模式的支持。
  - 增加了对在非正常节点关闭场景下强制分离ONTAP-SAN-* 卷的支持。
  - 所有ONTAP-SAN-* 卷现在都将使用按节点 igroup。只有在 LUN 主动发布到这些节点时，才会将其映射到 igroup，以提高我们的安全态势。当Trident认为在不影响活动工作负载的情况下安全地将现有卷切换到新的 igroup 方案时，将会择机进行切换（"第758期"）。
  - 通过从ONTAP -SAN-* 后端清理未使用的 Trident 管理的 igroup，提高了Trident 的安全性。
- 为 ontap-nas-economy 和 ontap-nas-flexgroup 存储驱动程序添加了对Amazon FSx SMB 卷的支持。
- 增加了对 ontap-nas、ontap-nas-economy 和 ontap-nas-flexgroup 存储驱动程序的 SMB 共享的支持。
- 增加了对 arm64 节点的支持（"第732期"）。
- 改进了Trident的关闭流程，首先停用API服务器（"第811期"）。
- 在 Makefile 中添加了对 Windows 和 arm64 主机的跨平台构建支持；请参阅 BUILD.md。

弃用

Kubernetes：配置 ontap-san 和 ontap-san-economy 驱动程序时，将不再创建后端范围的 igroups（"第758期"）。

23.01.1 中的变更

修复

- 修复了Trident Operator 在规范中指定时使用 IPv6 localhost 进行安装的问题。
- 修复了Trident Operator 集群角色权限，使其与捆绑包权限保持同步。"第799期"。
- 添加了允许外部进程运行完成的修复程序。
- 修复了在 RWX 模式下将原始块卷附加到多个节点的问题。
- 修复了FlexGroup克隆支持和 SMB 卷的卷导入问题。

23.01 中的变化

Trident现已支持 Kubernetes 1.27。请先升级Trident，再升级 Kubernetes。

修复

- Kubernetes：添加了排除 Pod 安全策略创建的选项，以修复通过 Helm 安装Trident 的问题（"第 783 期、第 794 期"）。

增强功能

Kubernetes

- 增加了对 Kubernetes 1.26 的支持。
- 提高了Trident RBAC资源的整体利用率（"第757期"）。
- 增加了自动化功能，用于检测和修复主机节点上损坏或过期的 iSCSI 会话。
- 增加了对扩展 LUKS 加密卷的支持。
- Kubernetes：为 LUKS 加密卷添加了凭证轮换支持。

Trident

- 为 ontap-nas 存储驱动程序添加了对Amazon FSx for NetApp ONTAP 的SMB 卷的支持。
- 增加了在使用 SMB 卷时对 NTFS 权限的支持。
- 为具有 CVS 服务级别的 GCP 卷添加了存储池支持。
- 在使用 ontap-nas-flexgroup 存储驱动程序创建 FlexGroups 时，增加了对可选使用 flexgroupAggregateList 的支持。
- 改进了 ontap-nas-economy 存储驱动程序在管理多个FlexVol卷时的性能
- 已为所有ONTAP NAS 存储驱动程序启用 dataLIF 更新。
- 更新了Trident Deployment 和 DaemonSet 的命名约定，以反映主机节点操作系统。

弃用

- Kubernetes：已将支持的最低 Kubernetes 版本更新为 1.21。
- 配置时不应再指定 DataLIF。 `ontap-san`或者 `ontap-san-economy`司机。

22.10 中的变化

升级到**Trident 22.10**之前，请务必阅读以下重要信息。

关于Trident 22.10 的关键信息

- Trident现已支持 Kubernetes 1.25。在升级到 Kubernetes 1.25 之前，必须先将Trident升级到 22.10。
- Trident现在严格强制要求在 SAN 环境中使用多路径配置，建议值为 `find_multipaths: no`在 multipath.conf 文件中。

  使用非多路径配置或使用 `find_multipaths: yes`或者 `find_multipaths: smart`multipath.conf 文件中的值会导致挂载失败。 Trident建议使用 `find_multipaths: no`自 21.07 版本发布以来。

修复

- 修复了使用ONTAP后端创建的特定问题 `credentials`22.07.0 升级期间字段无法上线 ("第759期" ）。
- **Docker**：修复了导致 Docker 卷插件在某些环境下无法启动的问题（"第548期"和"第760期"）。
- 修复了ONTAP SAN 后端特有的 SLM 问题，以确保仅发布属于报告节点的 dataLIF 子集。

- 修复了附加卷时发生不必要的 iSCSI LUN 扫描的性能问题。
- 移除了Trident iSCSI 工作流程中的细粒度重试，以便快速失败并减少外部重试间隔。
- 修复了当相应的多路径设备已被刷新时，刷新 iSCSI 设备会返回错误的问题。

增强功能

- Kubernetes：
  - 新增对 Kubernetes 1.25 的支持。在升级到 Kubernetes 1.25 之前，必须先将Trident升级到 22.10。
  - 为Trident部署和 DaemonSet 添加了单独的 ServiceAccount、ClusterRole 和 ClusterRoleBinding，以便将来进行权限增强。
  - 增加了对"跨命名空间卷共享"。
- 所有Trident `ontap-*`存储驱动程序现在可以与ONTAP REST API 配合使用。
- 新增运算符 yaml(`bundle_post_1_25.yaml`) 没有 `PodSecurityPolicy`支持 Kubernetes 1.25。
- 额外"支持 LUKS 加密卷"为了 `ontap-san`和 `ontap-san-economy`存储驱动程序。
- 新增对 Windows Server 2019 节点的支持。
- 额外"支持 Windows 节点上的 SMB 卷"通过 `azure-netapp-files`存储驱动程序。
- ONTAP驱动程序的自动MetroCluster切换检测功能现已普遍可用。

弃用

- **Kubernetes：** 已将支持的最低 Kubernetes 版本更新为 1.20。
- 已移除Astra数据存储 (ADS) 驱动程序。
- 已移除对以下功能的支持 `yes`和 `smart`选项 `find_multipaths`配置 iSCSI 工作节点多路径时。

22.07 中的变化

修复

Kubernetes

- 修复了在使用 Helm 或Trident Operator 配置Trident时处理节点选择器的布尔值和数值的问题。 （"GitHub 问题 #700" ）
- 修复了处理非 CHAP 路径错误的问题，以便 kubelet 在失败时重试。"GitHub 问题 #736" ）

增强功能

- 将 CSI 镜像的默认注册表从 k8s.gcr.io 过渡到 registry.k8s.io
- ONTAP-SAN 卷现在将使用每个节点的 igroup，并且仅在主动发布到这些节点时才将 LUN 映射到 igroup，以提高我们的安全态势。当Trident认为在不影响当前工作负载的情况下安全地将现有卷切换到新的 igroup 方案时，将会择机进行。
- 在Trident安装中包含 ResourceQuota，以确保在 PriorityClass 消耗默认受到限制时， Trident DaemonSet 能够被调度。
- 为Azure NetApp Files驱动程序添加了对网络功能的支持。 （"GitHub 问题 #717" ）

- ONTAP驱动程序新增了技术预览版自动MetroCluster切换检测功能。 （"GitHub 问题 #228" ）

弃用

- **Kubernetes**： 已将支持的最低 Kubernetes 版本更新为 1.19。
- 后端配置不再允许在单个配置中使用多种身份验证类型。

搬家

- AWS CVS 驱动程序（自 22.04 版本起已弃用）已被移除。
- Kubernetes
  - 从节点 pod 中移除不必要的 SYS_ADMIN 功能。
  - 将 nodeprep 简化为简单的主机信息和主动服务发现，以尽力确认 NFS/iSCSI 服务在工作节点上可用。

文档

一个新的"舱体安全标准"（PSS）部分已添加，详细说明了Trident在安装时启用的权限。

22.04 中的变化

NetApp不断改进和增强其产品和服务。以下是Trident的一些最新功能。有关先前版本，请参阅 "早期版本的文档"。

> 如果您是从任何先前的Trident版本升级，并且使用Azure NetApp Files，则`location`配置参数现在是必填的单例字段。

修复

- 改进了 iSCSI 发起程序名称的解析。 （"GitHub 问题 #681" ）
- 修复了不允许使用 CSI 存储类参数的问题。 （"GitHub 问题 #598" ）
- 修复了Trident CRD 中重复的键声明。 （"GitHub 问题 #671" ）
- 修复了不准确的 CSI 快照日志。 （"GitHub 问题 #629" ））
- 修复了在已删除节点上取消发布卷的问题。 （"GitHub 问题 #691" ）
- 增加了对块设备上文件系统不一致性的处理。 （"GitHub 问题 #656" ）
- 修复了设置时自动拉取支撑图像的问题 `imageRegistry`安装过程中标记。 （"GitHub 问题 #715" ）
- 修复了Azure NetApp Files驱动程序无法克隆具有多个导出规则的卷的问题。

增强功能

- 现在，与 Trident 安全端点的入站连接至少需要 TLS 1.3。 （"GitHub 问题 #698" ）
- Trident现在会在其安全端点的响应中添加 HSTS 标头。
- Trident现在会尝试自动启用Azure NetApp FilesUnix 权限功能。
- **Kubernetes**： Trident daemonset 现在以 system-node-critical 优先级运行。 （"GitHub 问题 #694" ）

搬家

E系列驱动程序（自20.07版本起已禁用）已被移除。

22.01.1 中的变更

修复

- 修复了在已删除节点上取消发布卷的问题。 （"GitHub 问题 #691" ）
- 修复了在ONTAP API 响应中访问聚合空间的 nil 字段时发生的 panic 问题。

22.01.0 中的变更

修复

- **Kubernetes:** 增加大型集群的节点注册退避重试时间。
- 修复了 azure-netapp-files 驱动程序可能被多个同名资源混淆的问题。
- 如果用方括号指定， ONTAP SAN IPv6 DataLIF 现在可以正常工作。
- 修复了尝试导入已导入卷时返回 EOF 导致 PVC 处于待处理状态的问题。 （"GitHub 问题 #489" ）
- 修复了在SolidFire卷上创建超过 32 个快照时Trident性能变慢的问题。
- 在创建 SSL 证书时，将 SHA-1 替换为 SHA-256。
- 修复了Azure NetApp Files驱动程序，使其允许重复的资源名称，并将操作限制在单个位置。
- 修复了Azure NetApp Files驱动程序，使其允许重复的资源名称，并将操作限制在单个位置。

增强功能

- Kubernetes 增强功能：
  - 新增对 Kubernetes 1.23 的支持。
  - 通过Trident Operator 或 Helm 安装Trident pod 时，添加调度选项。 （"GitHub 问题 #651" ）
- 允许 GCP 驱动程序中的跨区域卷。 （"GitHub 问题 #633" ）
- 为Azure NetApp Files卷添加了对“unixPermissions”选项的支持。 （"GitHub 问题 #666" ）

弃用

Trident REST 接口只能监听和提供服务于 127.0.0.1 或 [::1] 地址

21.10.1 中的变更

> v21.10.0 版本存在一个问题，当节点被移除然后又被添加到 Kubernetes 集群时， Trident控制器可能会进入 CrashLoopBackOff 状态。此问题已在 v21.10.1 中修复（GitHub 问题 669）。

修复

- 修复了在 GCP CVS 后端导入卷时可能出现的竞争条件，该条件会导致导入失败。

- 修复了当节点被移除然后又被添加到 Kubernetes 集群时， Trident控制器可能进入 CrashLoopBackOff 状态的问题（GitHub 问题 669）。
- 修复了未指定 SVM 名称时无法发现 SVM 的问题（GitHub 问题 612）。

21.10.0 中的变更

修复

- 修复了 XFS 卷的克隆无法挂载到与源卷相同的节点上的问题（GitHub 问题 514）。
- 修复了Trident在关闭时记录致命错误的问题（GitHub 问题 597）。
- 与 Kubernetes 相关的修复：
  - 使用以下命令创建快照时，将卷的已用空间作为最小恢复大小返回： `ontap-nas`和 `ontap-nas-flexgroup`驱动程序（GitHub 问题 645）。
  - 修复了以下问题 `Failed to expand filesystem`卷大小调整后记录了错误（GitHub 问题 560）。
  - 修复了舱体可能卡住的问题 `Terminating`状态（GitHub 问题 572）。
  - 修复了以下情况： `ontap-san-economy` FlexVol可能充满了快照 LUN（GitHub 问题 533）。
  - 修复了使用不同镜像时自定义 YAML 安装程序的问题（GitHub 问题 613）。
  - 修复了快照大小计算（GitHub 问题 611）。
  - 修复了所有Trident安装程序都能将普通 Kubernetes 识别为 OpenShift 的问题（GitHub 问题 639）。
  - 修复了Trident操作符，使其在 Kubernetes API 服务器无法访问时停止协调（GitHub 问题 599）。

增强功能

- 增加了对 `unixPermissions`可选择 GCP-CVS 性能卷。
- 为 GCP 中 600 GiB 到 1 TiB 范围内的规模优化 CVS 卷添加了支持。
- Kubernetes 相关增强功能：
  - 新增对 Kubernetes 1.22 的支持。
  - 使Trident operator 和 Helm chart 能够与 Kubernetes 1.22 一起使用（GitHub 问题 628）。
  - 已添加操作员图像 `tridentctl`images 命令（GitHub 问题 570）。

实验性改进

- 增加了对卷复制的支持 `ontap-san`司机。
- 新增了对 REST 的*技术预览*支持 `ontap-nas-flexgroup`， `ontap-san` ， 和 `ontap-nas-economy`司机。

已知问题

已知问题是指可能妨碍您成功使用产品的问题。

- 当将已安装Trident 的Kubernetes 集群从 1.24 版本升级到 1.25 或更高版本时，必须更新 values.yaml 文件进行设置。 `excludePodSecurityPolicy`到 `true`或添加 `--set excludePodSecurityPolicy=true`到 `helm

upgrade`升级集群前必须先执行此命令。

- Trident现在强制执行空白 `fsType` (`fsType=""`对于没有卷的 `fsType`在其存储类中指定。使用 Kubernetes 1.17 或更高版本时， Trident支持提供一个空白的 `fsType`适用于 NFS 卷。对于 iSCSI 卷，您需要设置 `fsType`在强制执行 StorageClass 时 `fsGroup`使用安全上下文。
- 当在多个Trident实例中使用后端时，每个后端配置文件都应该有不同的配置。 `storagePrefix` ONTAP后端的值，或者使用其他值 `TenantName`适用于SolidFire后端。 Trident无法检测到其他Trident实例创建的卷。尝试在ONTAP或SolidFire后端创建现有卷都会成功，因为Trident将卷创建视为幂等操作。如果 `storagePrefix`或者 `TenantName`即使名称相同，在同一后端创建的卷也可能出现名称冲突。
- 安装Trident时（使用 `tridentctl`或Trident运算符）并使用 `tridentctl`要管理Trident，您应该确保 `KUBECONFIG`环境变量已设置。这是为了指明 Kubernetes 集群。 `tridentctl`应该起到反作用。当使用多个 Kubernetes 环境时，您应该确保： `KUBECONFIG`文件来源准确无误。
- 要对 iSCSI PV 执行在线空间回收，工作节点上的底层操作系统可能需要将挂载选项传递给卷。对于 RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 实例来说，情况确实如此，这些实例需要 `discard` "挂载选项"确保你的配置中包含 discard mountOption。[`StorageClass` ^] 支持在线块丢弃。
- 如果每个 Kubernetes 集群中有多个Trident实例， Trident将无法与其他实例通信，也无法发现它们创建的其他卷，如果集群中运行多个实例，则会导致意外和不正确的行为。每个 Kubernetes 集群应该只有一个Trident实例。
- 如果基于三叉戟 `StorageClass`当Trident离线时，Kubernetes 中的对象会被删除；当Trident重新上线时，它不会从数据库中删除相应的存储类。您应该使用以下命令删除这些存储类 `tridentctl`或者使用 REST API。
- 如果用户在删除相应的 PVC 之前删除了Trident提供的 PV， Trident不会自动删除支持卷。您应该通过以下方式移除音量： `tridentctl`或者使用 REST API。
- 除非每个配置请求的聚合集都是唯一的，否则ONTAP不能同时配置多个FlexGroup 。
- 使用 IPv6 上的Trident时，您应该指定 `managementLIF`和 `dataLIF`在方括号内的后端定义中。例如 ，`[fd20:8b1e:b258:2000:f816:3eff:feec:0]` 。

> 您无法指定 `dataLIF`基于ONTAP SAN 后端。 Trident会发现所有可用的 iSCSI LIF，并使用它们来建立多路径会话。

- 如果使用 `solidfire-san`使用 OpenShift 4.5 的驱动程序时，请确保底层工作节点使用 MD5 作为 CHAP 认证算法。 Element 12.7 提供符合 FIPS 标准的 CHAP 安全算法 SHA1、SHA-256 和 SHA3-256。

查找更多信息

- "Trident GitHub"
- "Trident博客"

早期版本的文档

如果您运行的不是Trident 25.06 版本，则可以根据以下信息获取先前版本的文档："Trident支持生命周期" 。

- "Trident25.02"
- "Trident24.10"
- "Trident24.06"

- "Trident24.02"
- "Trident23.10"
- "Trident23.07"
- "Trident23.04"
- "Trident23.01"
- "Trident22.10"

已知问题

已知问题指明了可能会阻止您成功使用此版本产品的问题。

以下已知问题会影响当前版本：

恢复 **Restic** 备份的大文件可能会失败

从使用 Restic 创建的 Amazon S3 备份中恢复 30GB 或更大的文件时，恢复操作可能会失败。作为变通方法，可以使用 Kopia 作为数据移动工具备份数据（Kopia 是备份的默认数据移动工具）。请参阅 "使用Trident Protect保护应用程序"以获取说明。

开始使用

了解Trident

了解Trident

Trident是由NetApp维护的完全支持的开源项目。它旨在帮助您使用行业标准接口（例如容器存储接口 (CSI)）来满足容器化应用程序的持久性需求。

什么是Trident?

NetApp Trident支持在所有流行的NetApp存储平台（包括公有云和本地）上使用和管理存储资源，例如本地ONTAP集群（AFF、 FAS和ASA）、 ONTAP Select、 Cloud Volumes ONTAP、Element 软件（NetApp HCI、 SolidFire）、 Azure NetApp Files、 Amazon FSx for NetApp ONTAP和 Google Cloud 上的Cloud Volumes Service 。

Trident是一个符合容器存储接口 (CSI) 标准的动态存储编排器，可与以下系统原生集成："Kubernetes" 。Trident以单个 Controller Pod 和集群中每个工作节点上的 Node Pod 的形式运行。参考 "Trident架构" 了解详情。

Trident还为NetApp存储平台提供与 Docker 生态系统的直接集成。 NetApp Docker Volume Plugin (nDVP) 支持从存储平台到 Docker 主机的存储资源配置和管理。参考 "部署适用于 Docker 的Trident" 了解详情。

如果您是第一次使用 Kubernetes，您应该先熟悉一下它。"Kubernetes 概念和工具" 。

Kubernetes 与NetApp产品集成

NetApp的存储产品组合与 Kubernetes 集群的许多方面集成，提供高级数据管理功能，从而增强 Kubernetes 部署的功能、能力、性能和可用性。

Amazon FSx for NetApp ONTAP

"Amazon FSx for NetApp ONTAP"是一项完全托管的 AWS 服务，可让您启动和运行由NetApp ONTAP存储操作系统支持的文件系统。

Azure NetApp Files

"Azure NetApp Files"是由NetApp提供支持的企业级 Azure 文件共享服务。您可以在 Azure 中原生运行要求最高的基于文件的工作负载，并获得您期望从NetApp获得的高性能和丰富的数据管理功能。

Cloud Volumes ONTAP

"Cloud Volumes ONTAP"是一款纯软件存储设备，可在云端运行ONTAP数据管理软件。

Google Cloud NetApp Volumes

"Google Cloud NetApp Volumes"是 Google Cloud 中完全托管的文件存储服务，提供高性能、企业级的文件存储。

Element软件

"Element"通过保证性能并实现简化和精简的存储布局，使存储管理员能够整合工作负载。

NetApp HCI

"NetApp HCI"通过自动化日常任务，简化数据中心的管理和扩展，使基础设施管理员能够专注于更重要的功能。

Trident可以直接针对底层NetApp HCI存储平台，为容器化应用程序配置和管理存储设备。

NetApp ONTAP

"NetApp ONTAP"NetApp是 NetApp 的多协议统一存储操作系统，可为任何应用程序提供高级数据管理功能。

ONTAP系统具有全闪存、混合或全 HDD 配置，并提供多种不同的部署模型：本地FAS、AFA 和ASA集群、 ONTAP Select和Cloud Volumes ONTAP。 Trident支持这些ONTAP部署模型。

Trident架构

Trident以单个 Controller Pod 和集群中每个工作节点上的 Node Pod 的形式运行。节点 pod 必须运行在您希望挂载Trident卷的任何主机上。

了解控制器 **pod** 和节点 **pod**

Trident以单个部署。Trident控制器舱以及一个或多个Trident节点 Pod在 Kubernetes 集群上，并使用标准的 Kubernetes *CSI Sidecar Containers* 来简化 CSI 插件的部署。"Kubernetes CSI Sidecar 容器"由 Kubernetes 存储社区维护。

Kubernetes"节点选择器"和"容忍与污点"用于限制 pod 在特定或首选节点上运行。在Trident安装过程中，您可以为控制器和节点 pod 配置节点选择器和容差。

- 控制器插件负责卷的配置和管理，例如快照和调整大小。
- 节点插件负责将存储连接到节点。

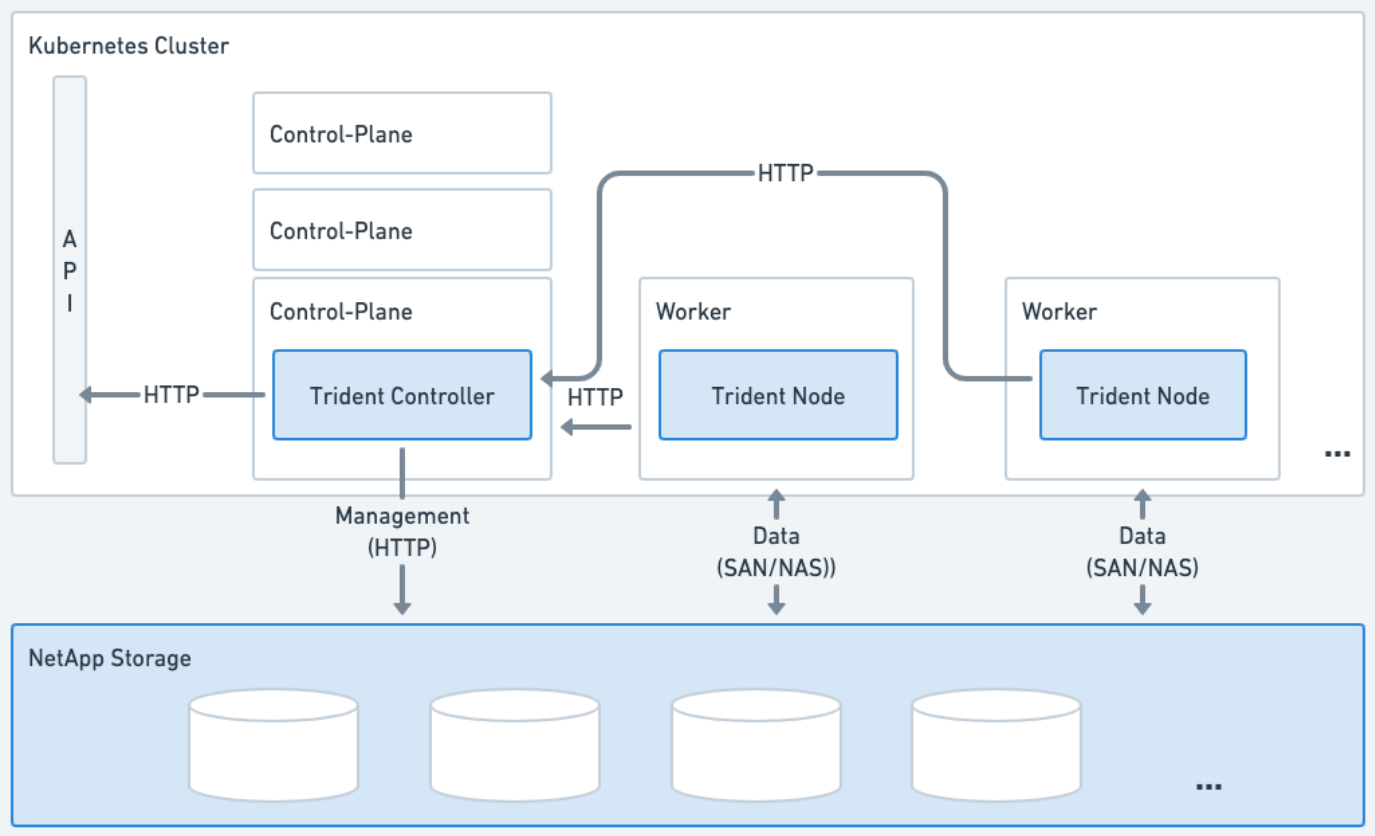


图 **1. Trident**已部署在 **Kubernetes** 集群上

Trident控制器舱

Trident Controller Pod 是一个运行 CSI Controller 插件的单个 Pod。

- 负责在NetApp存储中配置和管理卷
- 由 Kubernetes 部署管理
- 根据安装参数的不同，可以在控制平面或工作节点上运行。

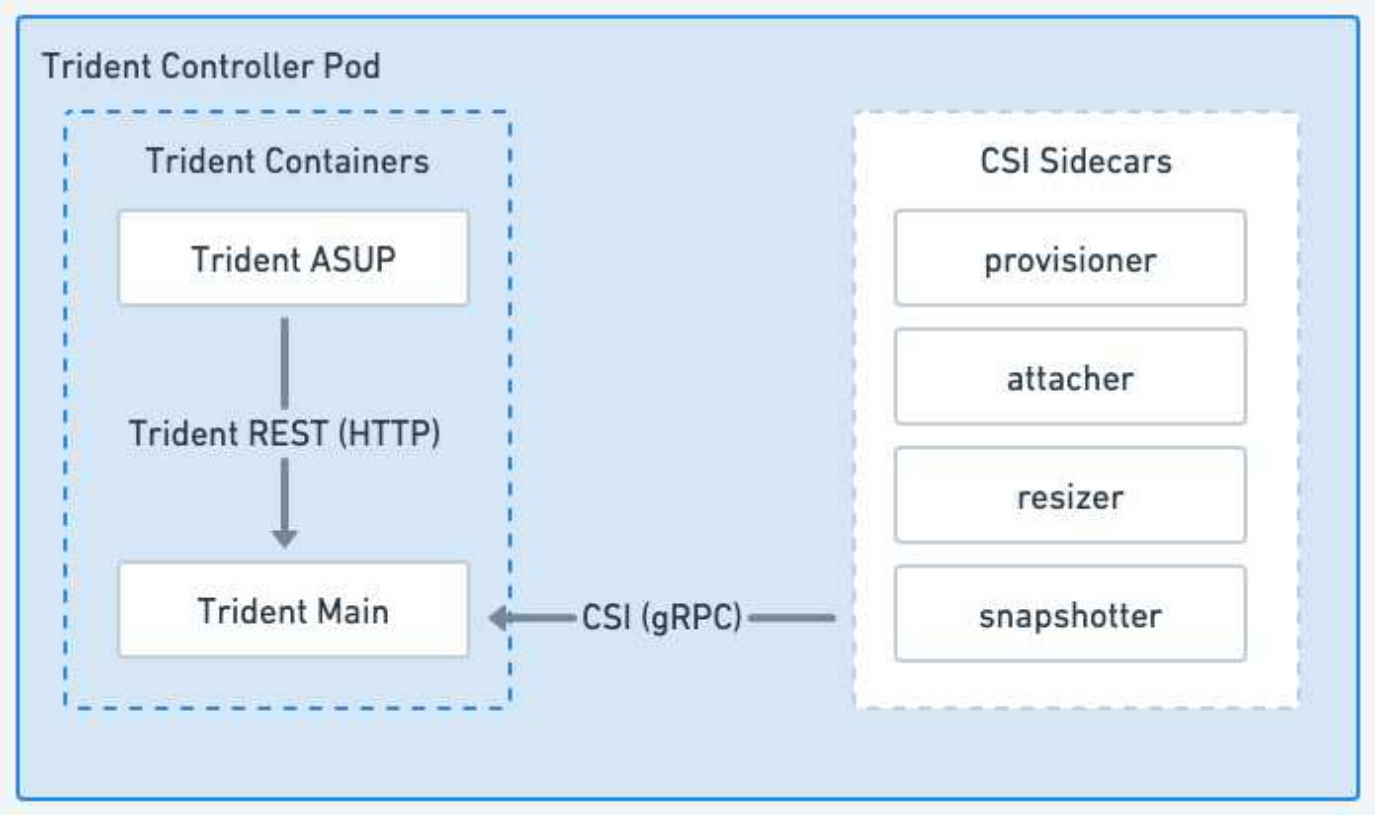


图 **2. Trident**控制器舱示意图

Trident节点 **Pod**

Trident Node Pod 是运行 CSI Node 插件的特权 Pod。

- 负责挂载和卸载主机上运行的 Pod 的存储设备
- 由 Kubernetes DaemonSet 管理
- 必须在任何能够挂载NetApp存储的节点上运行

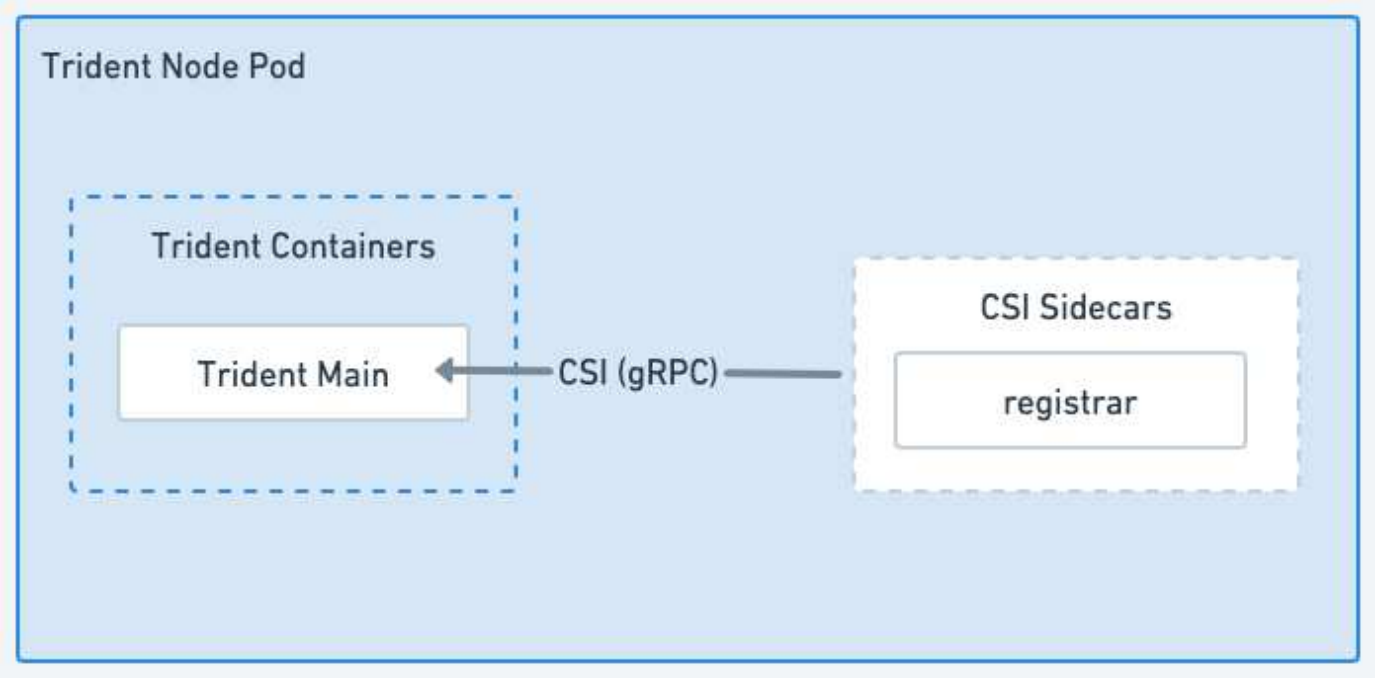


图 **3. Trident**节点舱示意图

支持的 **Kubernetes** 集群架构

Trident支持以下 Kubernetes 架构：

Kubernetes集群架构	支持	默认安装
单主控计算	是	是
多主计算	是	是
掌握， `etcd`计算	是	是
主控、基础设施、计算	是	是

概念

配置

Trident的配置过程分为两个主要阶段。第一阶段将存储类与一组合适的后端存储池关联起来，这是在进行配置之前必须进行的准备工作。第二阶段包括卷的创建本身，需要从与待创建卷的存储类别关联的存储池中选择一个存储池。

存储类关联

将后端存储池与存储类关联起来，取决于存储类的请求属性及其 `storagePools`， `additionalStoragePools` ， 和 `excludeStoragePools`列表。创建存储类时， Trident会将其每个后端提供的属性和池与存储类请求的属性和池进行比较。如果存储池的属性和名称与所有请求的属性和池名称匹配，Trident会将该存储池添加到该存储类的适用存储池集合中。此外， Trident还添加了列表中列出的所有存储池。

`additionalStoragePools`即使它们的属性不满足存储类的所有或任何请求属性，也要将其添加到该集合中。你应该使用 `excludeStoragePools`列出要覆盖和移除存储类使用的存储池。每次添加新的后端时， Trident都会执行类似的过程，检查其存储池是否满足现有存储类的要求，并删除任何被标记为排除的存储池。

创建卷

Trident然后利用存储类和存储池之间的关联来确定在哪里配置卷。创建卷时， Trident首先获取该卷存储类别的存储池集合，如果您为该卷指定了协议， Trident会删除那些无法提供所请求协议的存储池（例如， NetApp HCI/ SolidFire后端无法提供基于文件的卷，而ONTAP NAS 后端无法提供基于块的卷）。 Trident会随机化所得集合的顺序，以方便卷的均匀分布，然后遍历该集合，依次尝试在每个存储池上配置卷。如果一次成功，则返回成功结果，并将过程中遇到的任何失败记录下来。 Trident仅在无法为请求的存储类别和协议配置所有可用的存储池时才会返回失败。

卷快照

了解更多关于Trident如何处理其驱动程序的卷快照创建的信息。

了解如何创建卷快照

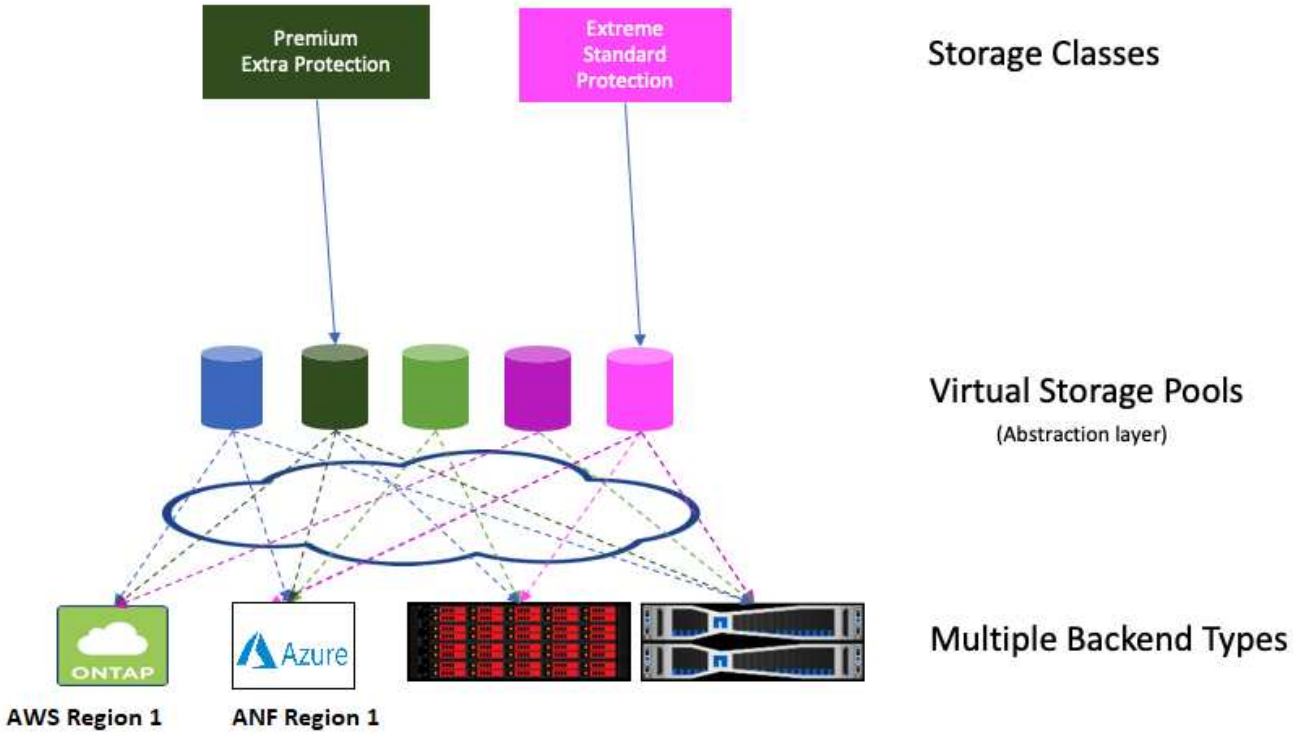
- 对于 `ontap-nas`， `ontap-san` ， `gcp-cvs` ， 和 `azure-netapp-files`驱动程序中，每个持久卷 (PV) 都映射到一个FlexVol volume。因此，卷快照被创建为NetApp快照。 NetApp快照技术比同类快照技术具有更高的稳定性、可扩展性、可恢复性和性能。这些快照副本在创建所需时间和存储空间方面都非常高效。
- 对于 `ontap-nas-flexgroup`驱动程序中，每个持久卷 (PV) 都映射到一个FlexGroup。因此，卷快照将创建为NetApp FlexGroup快照。 NetApp快照技术比同类快照技术具有更高的稳定性、可扩展性、可恢复性和性能。这些快照副本在创建所需时间和存储空间方面都非常高效。
- 对于 `ontap-san-economy`驱动程序，PV 映射到在共享FlexVol卷上创建的 LUN。通过对关联的 LUN 执行 FlexClone 操作，即可获得 PV 的 VolumeSnapshot。 ONTAP FlexClone技术能够几乎瞬间创建即使是最大数据集的副本。副本与其父级共享数据块，除了元数据所需的空间外，不占用任何存储空间。
- 对于 `solidfire-san`驱动程序中，每个 PV 映射到在NetApp Element软件/ NetApp HCI集群上创建的 LUN。VolumeSnapshots 由底层 LUN 的 Element 快照表示。这些快照是特定时间点的副本，仅占用少量系统资源和空间。
- 当与 `ontap-nas`和 `ontap-san`驱动程序中， ONTAP快照是FlexVol的某个时间点的副本，会占用FlexVol本身的空间。随着快照的创建/计划，这会导致卷中的可写空间随时间减少。解决此问题的一个简单方法是通过 Kubernetes 调整大小来增加卷。另一种方法是删除不再需要的快照。当通过 Kubernetes 创建的 VolumeSnapshot 被删除时， Trident将删除关联的ONTAP快照。并非通过 Kubernetes 创建的ONTAP快照也可以删除。

使用Trident，您可以利用 VolumeSnapshots 从中创建新的 PV。使用FlexClone技术从这些快照创建 PV，适用于受支持的ONTAP和 CVS 后端。从快照创建 PV 时，后备卷是快照父卷的FlexClone 。这 `solidfire-san`驱动程序使用 Element 软件卷克隆从快照创建 PV。它在这里从 Element 快照创建一个克隆。

虚拟池

虚拟池在Trident存储后端和 Kubernetes 之间提供了一个抽象层。 `StorageClasses` 。它们允许管理员以通用的、与后端无关的方式定义每个后端的各个方面，例如位置、性能和保护，而无需创建单独的配置。 `StorageClass`指定要使用的物理后端、后端池或后端类型，以满足所需条件。

存储管理员可以在任何Trident后端上，通过 JSON 或 YAML 定义文件定义虚拟池。



在虚拟池列表之外指定的任何方面对于后端都是全局的，并将应用于所有虚拟池；而每个虚拟池可以单独指定一个或多个方面（覆盖任何后端全局方面）。

- 定义虚拟池时，不要尝试重新排列后端定义中现有虚拟池的顺序。
- 我们不建议修改现有虚拟池的属性。您需要定义一个新的虚拟池来进行更改。

大多数方面都是用后端特有的术语来规定的。至关重要的是，这些方面值不会暴露在后端驱动程序之外，也无法用于匹配。 `StorageClasses`相反，管理员可以为每个虚拟池定义一个或多个标签。每个标签都是一个键值对，并且标签可能在不同的后端中是相同的。与方面类似，标签可以针对每个池进行指定，也可以全局指定到后端。与具有预定义名称和值的方面不同，管理员可以完全自主地根据需要定义标签键和值。为了方便起见，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。

可以使用以下字符来定义虚拟池标签：

- 大写字母 `A-Z`
- 小写字母 `a-z`
- 数字 `0-9`
- 下划线 `_`
- 连字符 `-`

一个 `StorageClass`通过引用选择器参数中的标签来确定要使用的虚拟池。虚拟池选择器支持以下运算符：

运算符	示例	池的标签值必须：
=	性能=优质	匹配
!=	性能！=极限	不匹配
in	位置在（东，西）	属于价值集合。
notin	表演奖（银奖、铜奖）	不在值集中
<key>	保护	存在且具有任意值
!<key>	！保护	不存在

卷访问组

了解更多关于Trident如何使用 "卷访问组"。

> 如果您使用的是 CHAP，请忽略此部分。建议使用 CHAP 以简化管理并避免下文所述的扩展限制。此外，如果您在 CSI 模式下使用Trident ，则可以忽略此部分。 Trident在作为增强型 CSI 配置器安装时使用 CHAP。

了解卷访问组

Trident可以使用卷访问组来控制对其所配置卷的访问。如果 CHAP 被禁用，它会寻找一个名为“访问组”的访问组。 `trident`除非您在配置中指定一个或多个访问组 ID。

Trident会将新卷与已配置的访问组关联起来，但它本身并不创建或以其他方式管理访问组。在将存储后端添加到Trident之前，访问组必须存在，并且它们需要包含 Kubernetes 集群中每个节点的 iSCSI IQN，这些节点可能会挂载由该后端配置的卷。在大多数安装中，这包括集群中的每个工作节点。

对于节点数超过 64 个的 Kubernetes 集群，您应该使用多个访问组。每个访问组最多可包含 64 个 IQN，每个卷可属于四个访问组。配置最多四个访问组后，集群中最多 256 个节点中的任何节点都将能够访问任何卷。有关卷访问组的最新限制，请参阅 "此处"。

如果您要修改的配置是基于默认配置的，那么请注意以下事项。 `trident`访问组必须与其他组一起使用，并且必须包含 ID。 `trident`访问列表中的组。

Trident快速入门

只需几个步骤，即可安装Trident并开始管理存储资源。开始之前，请先回顾一下"Trident的要求"。

> 有关 Docker 的信息，请参阅"Trident for Docker"。

1 准备工作节点

Kubernetes 集群中的所有工作节点都必须能够挂载您为 Pod 配置的卷。

"准备工作节点"

安装**Trident**

Trident提供多种安装方法和模式，针对各种环境和组织进行了优化。

"安装Trident"

创建后端

后端定义了Trident与存储系统之间的关系。它告诉Trident如何与该存储系统通信，以及Trident应该如何从中配置卷。

"配置后端"适用于您的存储系统

创建 **Kubernetes** 存储类

Kubernetes StorageClass 对象指定Trident作为配置器，并允许您创建存储类以配置具有可自定义属性的卷。Trident为指定Trident配置器的 Kubernetes 对象创建匹配的存储类。

"创建存储类"

提供一定量

持久卷 (PV) 是 Kubernetes 集群上由集群管理员配置的物理存储资源。 *PersistentVolumeClaim* (PVC) 是对集群上持久卷的访问请求。

创建一个持久卷 (PV) 和一个持久卷声明 (PVC)，使用配置的 Kubernetes StorageClass 请求访问 PV。然后您可以将光伏组件安装到支架上。

"提供一定量"

下一步是什么?

现在您可以添加其他后端、管理存储类、管理后端以及执行卷操作。

要求

安装Trident之前，您应该查看这些通用系统要求。特定后端可能还有其他要求。

关于**Trident**的关键信息

您必须阅读以下关于**Trident**的重要信息。

- Trident现已支持 Kubernetes 1.34。在升级 Kubernetes 之前先升级Trident 。
- Trident严格强制要求在 SAN 环境中使用多路径配置，建议值为 `find_multipaths: no`在 multipath.conf 文件中。

  使用非多路径配置或使用 `find_multipaths: yes`或者 `find_multipaths: smart`multipath.conf 文件中的值会导致挂载失败。 Trident建议使用 `find_multipaths: no`自 21.07 版本发布以来。

支持的前端（编排器）

Trident支持多种容器引擎和编排器，包括以下几种：

- Anthos On-Prem (VMware) 和 Anthos on Bare Metal 1.16
- Kubernetes 1.27 - 1.34
- OpenShift 4.12、4.14 - 4.19（如果您计划使用 OpenShift 4.19 进行 iSCSI 节点准备，则支持的最低Trident版本为 25.06.1。）

  Trident继续支持旧版本的 OpenShift，以符合……"Red Hat 扩展更新支持 (EUS) 发布生命周期"即使他们依赖于上游不再官方支持的 Kubernetes 版本。在这种情况下安装Trident时，您可以放心地忽略有关 Kubernetes 版本的任何警告消息。

- Rancher Kubernetes Engine 2（RKE2）v1.27.x - 1.34.x

  虽然Trident在 Rancher Kubernetes Engine 2 (RKE2) 版本 1.27.x - 1.34.x 上受支持，但Trident目前仅在 RKE2 v1.28.5+rke2r1 上获得认证。

Trident还与许多其他完全托管和自托管的 Kubernetes 产品合作，包括 Google Kubernetes Engine (GKE)、Amazon Elastic Kubernetes Services (EKS)、Azure Kubernetes Service (AKS)、Mirantis Kubernetes Engine (MKE) 和 VMWare Tanzu Portfolio。

Trident和ONTAP可用作存储提供程序"KubeVirt"。

在将已安装Trident的 Kubernetes 集群从 1.25 版本升级到 1.26 或更高版本之前，请参阅以下内容："升级 Helm 安装" 。

支持的后端（存储）

要使用Trident，您需要以下一个或多个受支持的后端：

- Amazon FSx for NetApp ONTAP
- Azure NetApp Files
- Cloud Volumes ONTAP
- Google Cloud NetApp Volumes

- NetApp全 SAN 阵列 (ASA)
- 在 NetApp 完全或有限支持下运行 ONTAP 版本的本地 FAS、AFF 或 ASA r2（iSCSI、NVMe/TCP 和 FC ）。请参阅 "软件版本支持"。
- NetApp HCI/Element 软件 11 或更高版本

Trident对 KubeVirt 和 OpenShift 虚拟化的支持

支持的存储驱动程序：

Trident支持以下适用于 KubeVirt 和 OpenShift 虚拟化的ONTAP驱动程序：

- ontap-nas
- ontap-nas-economy
- ontap-san（基于 TCP 的 iSCSI、FCP、NVMe）
- ontap-san-economy（仅限 iSCSI）

需要考虑的要点：

- 更新存储类以使其具有 `fsType`参数（例如： `fsType: "ext4"`在 OpenShift 虚拟化环境中。如果需要，请使用以下方式显式设置音量模式为阻止模式： `volumeMode=Block`参数 `dataVolumeTemplates`通知 CDI 创建块数据卷。
- 块存储驱动程序的 *RWX* 访问模式: ontap-san (iSCSI、NVMe/TCP、FC) 和 ontap-san-economy (iSCSI) 驱动程序仅支持“volumeMode: Block”（原始设备）。对于这些司机来说， `fstype`由于卷是以原始设备模式提供的，因此无法使用该参数。
- 对于需要 RWX 访问模式的实时迁移工作流程，支持以下组合：
  - NFS+ `volumeMode=Filesystem`
  - iSCSI+ `volumeMode=Block` （原始设备）
  - NVMe/TCP + `volumeMode=Block` （原始设备）
  - FC+ `volumeMode=Block` （原始设备）

功能需求

下表总结了此版本Trident提供的功能及其支持的 Kubernetes 版本。

功能	**Kubernetes** 版本	需要设置特色门吗?
Trident	1.27 - 1.34	否
卷 Snapshot	1.27 - 1.34	否
PVC 来自卷快照	1.27 - 1.34	否
iSCSI PV 调整大小	1.27 - 1.34	否
ONTAP双向 CHAP	1.27 - 1.34	否

功能	**Kubernetes** 版本	需要设置特色门吗?
动态出口政策	1.27 - 1.34	否
Trident操作员	1.27 - 1.34	否
CSI拓扑	1.27 - 1.34	否

测试过的主机操作系统

虽然Trident官方并未正式支持特定操作系统，但已知以下操作系统可以正常工作：

- OpenShift 容器平台在 AMD64 和 ARM64 架构上支持的 Red Hat Enterprise Linux CoreOS (RHCOS) 版本
- Red Hat Enterprise Linux (RHEL) 8 或更高版本，支持 AMD64 和 ARM64 架构

NVMe/TCP 需要 RHEL 9 或更高版本。

- Ubuntu 22.04 LTS 或更高版本，支持 AMD64 和 ARM64 架构
- Windows Server 2022
- SUSE Linux Enterprise Server (SLES) 15 或更高版本

默认情况下， Trident在容器中运行，因此可以在任何 Linux 工作节点上运行。但是，这些工作人员需要能够使用标准 NFS 客户端或 iSCSI 发起程序挂载Trident提供的卷，具体取决于您使用的后端。

这 `tridentctl`该实用程序也可在上述任何 Linux 发行版上运行。

主机配置

Kubernetes 集群中的所有工作节点都必须能够挂载您为 Pod 配置的卷。要准备工作节点，您必须根据所选驱动程序安装 NFS、iSCSI 或 NVMe 工具。

"准备工作节点"

存储系统配置

Trident可能需要对存储系统进行更改，后端配置才能使用它。

"配置后端"

Trident港口

Trident需要访问特定端口才能进行通信。

"Trident港口"

容器镜像和相应的 **Kubernetes** 版本

对于物理隔离安装，以下列表是安装Trident所需的容器镜像参考。使用 `tridentctl images`用于验证所需容器镜像列表的命令。

Trident 25.06.2 所需的容器镜像

Kubernetes 版本	容器图像
v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0、v1.32.0、v1.33.0、v1.34.0	• docker.io/netapp/trident:25.06.2 • docker.io/netapp/trident-autosupport:25.06 • registry.k8s.io/sig-storage/csi-provisioner:v5.2.0 • registry.k8s.io/sig-storage/csi-attacher:v4.8.1 • registry.k8s.io/sig-storage/csi-resizer:v1.13.2 • registry.k8s.io/sig-storage/csi-snapshotter:v8.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.13.0 • docker.io/netapp/trident-operator:25.06.2（可选）

Trident 25.06 所需的容器镜像

Kubernetes 版本	容器图像
v1.27.0、v1.28.0、v1.29.0、v1.30.0、v1.31.0、v1.32.0、v1.33.0、v1.34.0	• docker.io/netapp/trident:25.06.0 • docker.io/netapp/trident-autosupport:25.06 • registry.k8s.io/sig-storage/csi-provisioner:v5.2.0 • registry.k8s.io/sig-storage/csi-attacher:v4.8.1 • registry.k8s.io/sig-storage/csi-resizer:v1.13.2 • registry.k8s.io/sig-storage/csi-snapshotter:v8.2.1 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.13.0 • docker.io/netapp/trident-operator:25.06.0（可选）

安装Trident

使用Trident操作员进行安装

使用 tridentctl 安装

使用 OpenShift 认证操作员进行安装

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挂载您为 Pod 配置的卷。要准备工作节点，您必须根据所选驱动程序安装 NFS、iSCSI、NVMe/TCP 或 FC 工具。

选择合适的工具

如果您使用的是多种驱动程序，则应安装所有驱动程序所需的工具。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"如果您正在使用： `ontap-nas` ， `ontap-nas-economy` ， `ontap-nas-flexgroup` ， `azure-netapp-files` ， `gcp-cvs` 。

iSCSI 工具

"安装 iSCSI 工具"如果您正在使用： `ontap-san` ， `ontap-san-economy` ， `solidfire-san` 。

NVMe 工具

"安装 NVMe 工具"如果你正在使用 `ontap-san`用于基于 TCP 的非易失性存储器高速接口 (NVMe) (NVMe/TCP) 协议。

NetApp建议 NVMe/TCP 使用ONTAP 9.12 或更高版本。

通过光纤通道 **(FC)** 进行 **SCSI** 的工具

请参阅"配置 FC 和 FC-NVMe SAN 主机的方法"有关配置 FC 和 FC-NVMe SAN 主机的更多信息。

"安装 FC 工具"如果你正在使用 `ontap-san`使用 `sanType` `fcp` （通过光纤通道进行 SCSI 通信）。

需要考虑的要点：* OpenShift 和 KubeVirt 环境支持通过 FC 进行 SCSI 通信。 * Docker 不支持通过 FC 传输 SCSI。 * iSCSI 自愈功能不适用于基于 FC 的 SCSI。

节点服务发现

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

节点服务发现功能可以识别已发现的服务，但不能保证服务配置正确。反之，未发现服务并不保证卷挂载一定会失败。

回顾事件

Trident会为节点创建事件，以识别已发现的服务。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识别Trident节点 CR 上每个节点启用的服务。要查看已发现的服务，请运行：

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

NFS卷

使用适用于您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 工具后重启工作节点，以防止将卷附加到容器时发生故障。

iSCSI 卷

Trident可以自动建立 iSCSI 会话、扫描 LUN、发现多路径设备、格式化设备并将其挂载到 pod 中。

iSCSI自愈能力

对于ONTAP系统， Trident每五分钟运行一次 iSCSI 自愈程序，以：

1. *确定*所需的 iSCSI 会话状态和当前的 iSCSI 会话状态。
2. 将理想状态与当前状态进行比较，以确定需要进行的维修。 Trident确定维修优先级以及何时进行抢修。
3. 执行必要的修复，使当前的 iSCSI 会话状态恢复到所需的 iSCSI 会话状态。

自愈活动的日志位于…… `trident-main`相应 Daemonset pod 上的容器。要查看日志，您必须已设置 `debug`在Trident安装过程中设置为“true”。

Trident iSCSI 的自愈功能可以帮助预防：

- 网络连接问题后可能出现过期或不健康的 iSCSI 会话。如果会话已过期， Trident会等待七分钟，然后注销并重新与门户建立连接。

例如，如果存储控制器上轮换了 CHAP 密钥，并且网络失去连接，则旧的（过时的）CHAP 密钥可能会保留下来。自愈功能可以识别这一点，并自动重新建立会话以应用更新后的 CHAP 密钥。

- 缺少 iSCSI 会话
- 缺少 LUN

升级**Trident**之前需要考虑的要点

- 如果仅使用每个节点的 igroup（在 23.04+ 中引入），则 iSCSI 自愈功能将启动 SCSI 总线上所有设备的 SCSI 重新扫描。
- 如果仅使用后端范围的 igroup（自 23.04 版本起已弃用），则 iSCSI 自愈功能将启动 SCSI 重新扫描，以查找 SCSI 总线上的确切 LUN ID。
- 如果同时使用节点级 igroup 和后端级 igroup，iSCSI 自愈功能将启动 SCSI 重新扫描，以查找 SCSI 总线上的精确 LUN ID。

安装 **iSCSI** 工具

使用适用于您操作系统的命令安装 iSCSI 工具。

开始之前

- Kubernetes 集群中的每个节点都必须有一个唯一的 IQN。这是必要的前提条件。
- 如果使用 RHCOS 4.5 或更高版本，或其他与 RHEL 兼容的 Linux 发行版，则需要执行以下操作：`solidfire-san`对于驱动程序和Element OS 12.5 或更早版本，请确保将 CHAP 身份验证算法设置为 MD5。 `/etc/iscsi/iscsid.conf` Element 12.7 提供符合 FIPS 标准的 CHAP 安全算法 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) 且带有 iSCSI PV 的工作节点时，请指定 `discard`StorageClass 中的 mountOption 用于执行内联空间回收。参考 "红帽文档"。
- 请确保您已升级到最新版本。 `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. 请检查 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. 将扫描方式设置为手动：

```
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
```

> 确保 `/etc/multipath.conf`包含 `find_multipaths no`在下面 `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. 请检查 open-iscsi 版本是否为 2.0.874-5ubuntu2.10 或更高版本（适用于 bionic）或 2.0.874-7.1ubuntu6.1 或更高版本（适用于 focal）：

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

3. 将扫描方式设置为手动：

```
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
```

> 确保 /etc/multipath.conf`包含 `find_multipaths no`在下面 `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，您必须使用以下命令发现目标端口： `iscsiadm`开始之前 `open-iscsi`启动 iSCSI 守护进程。您也可以修改 `iscsi`服务启动 `iscsid`自动地。

配置或禁用 **iSCSI** 自愈

您可以配置以下Trident iSCSI 自愈设置来修复过期的会话：

- **iSCSI** 自愈间隔：确定 iSCSI 自愈的调用频率（默认值：5 分钟）。你可以通过设置较小的数字来增加运行频率，或者通过设置较大的数字来降低运行频率。

> 将 iSCSI 自愈间隔设置为 0 将完全停止 iSCSI 自愈功能。我们不建议禁用 iSCSI 自愈功能；只有在 iSCSI 自愈功能无法按预期工作或出于调试目的时，才应禁用该功能。

- **iSCSI** 自愈等待时间：确定 iSCSI 自愈在注销不健康的会话并尝试再次登录之前等待的时间（默认值：7 分

钟）。您可以将其配置为更大的数字，以便将识别为不健康的会话在注销之前等待更长时间，然后再尝试重新登录；或者配置为更小的数字，以便更快地注销和登录。

舵

要配置或更改 iSCSI 自愈设置，请传递以下参数： `iscsiSelfHealingInterval`和 `iscsiSelfHealingWaitTime`Helm 安装或 Helm 更新期间的参数。

以下示例将 iSCSI 自愈间隔设置为 3 分钟，自愈等待时间设置为 6 分钟：

```
helm install trident trident-operator-100.2506.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

三叉戟

要配置或更改 iSCSI 自愈设置，请传递以下参数： `iscsi-self-healing-interval`和 `iscsi-self-healing-wait-time`tridentctl 安装或更新期间的参数。

以下示例将 iSCSI 自愈间隔设置为 3 分钟，自愈等待时间设置为 6 分钟：

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

NVMe/TCP 卷

使用适用于您操作系统的命令安装 NVMe 工具。

- NVMe 需要 RHEL 9 或更高版本。
- 如果您的 Kubernetes 节点的内核版本太旧，或者您的内核版本没有 NVMe 软件包，则您可能需要将节点的内核版本更新为包含 NVMe 软件包的版本。

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

安装完成后，使用以下命令验证 Kubernetes 集群中的每个节点是否都具有唯一的 NQN：

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

Trident修改了 `ctrl_device_tmo`确保 NVMe 在路径中断时不会放弃路径的值。请勿更改此设置。

通过 FC 卷进行 SCSI 连接

现在您可以使用光纤通道 (FC) 协议和Trident在ONTAP系统上配置和管理存储资源。

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"了解更多信息。
2. 获取发起方（主机）上接口的 WWPN。

   请参考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 交换机上配置区域。

   有关信息，请参阅相应交换机供应商的文档。

   详情请参阅以下ONTAP文档：

   - "光纤通道和FCoE分区概述"
   - "配置 FC 和 FC-NVMe SAN 主机的方法"

安装 FC 工具

使用适用于您操作系统的命令安装 FC 工具。

- 当使用运行 RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 且带有 FC PV 的工作节点时，请指定 `discard`StorageClass 中的 mountOption 用于执行内联空间回收。参考 "红帽文档"。

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
```

> 确保 /etc/multipath.conf`包含 `find_multipaths no`在下面 `defaults。

3. 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
```

> 确保 /etc/multipath.conf`包含 `find_multipaths no`在下面 `defaults。

3. 确保 `multipath-tools`已启用并正在运行：

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

配置和管理后端

配置后端

后端定义了Trident与存储系统之间的关系。它告诉Trident如何与该存储系统通信，以及Trident应该如何从中配置卷。

Trident会自动提供符合存储类定义要求的后端存储池。了解如何配置存储系统的后端。

- "配置Azure NetApp Files后端"
- "配置Google Cloud NetApp Volumes后端"
- "为 Google Cloud Platform 后端配置Cloud Volumes Service"
- "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"

Azure NetApp Files

配置**Azure NetApp Files**后端

您可以将Azure NetApp Files配置为Trident的后端。您可以使用Azure NetApp Files后端附加 NFS 和 SMB 卷。 Trident还支持使用托管标识对 Azure Kubernetes 服务 (AKS) 集群进行凭据管理。

Azure NetApp Files驱动程序详细信息

Trident提供以下Azure NetApp Files存储驱动程序，以便与集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

驱动程序	协议	音量模式	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`

注意事项

- Azure NetApp Files服务不支持小于 50 GiB 的卷。如果请求的卷较小， Trident会自动创建 50GiB 的卷。
- Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。

为 **AKS** 管理身份

Trident支持"托管身份"适用于 Azure Kubernetes 服务集群。要利用托管身份提供的简化凭证管理功能，您必须具备以下条件：

- 使用 AKS 部署的 Kubernetes 集群
- 在 AKS Kubernetes 集群上配置的托管身份
- 已安装的Trident包括 `cloudProvider`指定 `"Azure"`。

Trident操作员

要使用Trident操作员安装Trident，请编辑 `tridentorchestrator_cr.yaml`设置 `cloudProvider`到 `"Azure"`。例如：

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

舵

以下示例安装Trident套装 `cloudProvider`使用环境变量连接到 `Azure` `$CP`：

```
helm install trident trident-operator-100.2506.0.tgz --create
-namespace --namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

以下示例安装Trident并进行设置 `cloudProvider`标记 `Azure`：

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

AKS 的云身份

云身份使 Kubernetes pod 能够通过作为工作负载身份进行身份验证来访问 Azure 资源，而无需提供显式的 Azure 凭据。

要在 Azure 中利用云身份功能，您必须具备以下条件：

- 使用 AKS 部署的 Kubernetes 集群
- 在 AKS Kubernetes 集群上配置工作负载身份和 oidc-issuer
- 已安装的Trident包括 `cloudProvider`指定 `"Azure"`和 `cloudIdentity`指定工作负载标识

Trident操作员

要使用Trident操作员安装Trident，请编辑 `tridentorchestrator_cr.yaml`设置 `cloudProvider`到 `"Azure"`并设置 `cloudIdentity`到 `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx`。

例如：

```
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
```

舵

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-
xxxxxxxxxx'"
```

以下示例安装Trident并进行设置 `cloudProvider`使用环境变量连接到 `Azure` `$CP`并设置 `cloudIdentity`使用环境变量 `$CI`：

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

<code>tridentctl</code>

请使用以下环境变量设置 云提供商 和 云身份 标志的值：

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

以下示例安装Trident并进行设置 `cloud-provider`标记 `$CP`，和 `cloud-identity`到 `$CI`：

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

在配置Azure NetApp Files后端之前，需要确保满足以下要求。

NFS 和 SMB 卷的先决条件

如果您是第一次使用Azure NetApp Files或在新的位置使用，则需要进行一些初始配置来设置 Azure NetApp文件并创建 NFS 卷。参考 "Azure：设置Azure NetApp Files并创建 NFS 卷"。

配置和使用 "Azure NetApp Files"后端需要以下组件：

- `subscriptionID`， `tenantID`， `clientID`， `location`， 和 `clientSecret`在 AKS 集群上使用托管身份时，这些是可选的。
- `tenantID`， `clientID`， 和 `clientSecret`在 AKS 集群上使用云身份时，这些是可选的。

- 容量池。参考"微软：为Azure NetApp Files创建容量池"。
- 委派给Azure NetApp Files 的子网。参考"Microsoft：将子网委派给Azure NetApp Files"。
- `subscriptionID`来自已启用Azure NetApp Files功能的 Azure 订阅。
- `tenantID`， `clientID`， 和 `clientSecret`来自"应用程序注册"在 Azure Active Directory 中拥有对Azure NetApp Files服务的足够权限。应用程序注册应使用以下任一方式：
  - 所有者或贡献者角色"Azure 预定义"。
  - 一个"自定义贡献者角色"订阅级别(`assignableScopes`）但权限仅限于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": []
      }
    ]
  }
}
```

- 蔚蓝 `location`包含至少一个 "委托子网"。截至Trident 22.01版本， `location`参数是后端配置文件顶层的一个必填字段。虚拟池中指定的位置值将被忽略。
- 使用 `Cloud Identity`得到 `client ID`从一个 "用户分配的托管身份"并指定该 `azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx` ID 。

中小企业卷的附加要求

要创建 SMB 卷，您必须具备以下条件：

- Active Directory 已配置并连接到Azure NetApp Files。参考"Microsoft：创建和管理Azure NetApp Files管理器的 Active Directory 连接"。
- 一个 Kubernetes 集群，包含一个 Linux 控制器节点和至少一个运行 Windows Server 2022 的 Windows 工作节点。 Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。
- 至少需要一个包含 Active Directory 凭据的Trident密钥，以便Azure NetApp Files可以向 Active Directory 进行身份验证。生成秘密 `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代理"或者"GitHub：适用于 Windows 的 CSI 代理"适用于在 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使用您的后端配置（子网、虚拟网络、服务级别和位置），在请求的位置中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相匹配。

* 从NetApp Trident 25.06 版本开始，手动 QoS 容量池作为技术预览版受到支持。*

Azure NetApp Files后端提供以下配置选项。

参数	描述	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	"azure-netapp-files"
`backendName`	自定义名称或存储后端	司机姓名 + "_" + 随机字符
`subscriptionID`	Azure 订阅的订阅 ID（在 AKS 群集上启用托管标识时为可选）。	
`tenantID`	在 AKS 集群上使用托管身份或云身份时，应用程序注册中的租户 ID 是可选的。	
`clientID`	在 AKS 集群上使用托管身份或云身份时，应用程序注册中的客户端 ID 是可选的。	
`clientSecret`	在 AKS 集群上使用托管身份或云身份时，应用程序注册中的客户端密钥是可选的。	
`serviceLevel`	之一 `Standard`， `Premium` ， 或者 `Ultra`	““ （随机的）
`location`	将在 Azure 中创建新卷的位置名称（在 AKS 群集上启用托管标识时为可选）。	
`resourceGroups`	用于筛选已发现资源的资源组列表	（无滤镜）
`netappAccounts`	用于筛选已发现资源的NetApp帐户列表	（无滤镜）
`capacityPools`	用于筛选已发现资源的容量池列表	（无过滤，随机）
`virtualNetwork`	具有委派子网的虚拟网络的名称	""
`subnet`	委派给的子网名称 `Microsoft.Netapp/volumes`	""
`networkFeatures`	卷的 VNet 功能集，可能是 `Basic`` 或者 `Standard`。网络功能并非在所有地区都可用，可能需要通过订阅才能启用。指定 `networkFeatures`未启用该功能会导致卷配置失败。	""

参数	描述	默认
`nfsMountOptions`	对 NFS 挂载选项进行精细控制。对于 SMB 卷，此设置将被忽略。要使用 NFS 版本 4.1 挂载卷，请包含以下内容 `nfsvers=4`在以逗号分隔的挂载选项列表中选择 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`	如果请求的卷大小大于此值，则配置失败。	（默认情况下不强制执行）
`debugTraceFlags`	故障排除时要使用的调试标志。例子， `\{"api": false, "method": true, "discovery": true}` 。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	无效的
`nasType`	配置 NFS 或 SMB 卷的创建。选项有 `nfs`， `smb`或空值。设置为 null 则默认使用 NFS 卷。	`nfs`
`supportedTopologies`	表示此后端支持的区域和区域列表。更多信息，请参阅"使用 CSI 拓扑"。	
`qosType`	表示 QoS 类型：自动或手动。* Trident 25.06 技术预览版*	自动
`maxThroughput`	设置允许的最大吞吐量，单位为 MiB/秒。仅支持手动 QoS 容量池。* Trident 25.06 技术预览版*	`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"。

所需权限和资源

如果在创建 PVC 时收到“未找到容量池”错误，则可能是您的应用程序注册没有关联的必要权限和资源（子网、虚拟网络、容量池）。如果启用调试功能， Trident将记录在创建后端时发现的 Azure 资源。请确认是否使用了合适的角色。

值 `resourceGroups`， `netappAccounts` ， `capacityPools` ， `virtualNetwork` ， 和 `subnet`可以使用简称或完全限定名称来指定。大多数情况下建议使用完全限定名称，因为短名称可能会匹配多个同名资源。

这 `resourceGroups`， `netappAccounts` ， 和 `capacityPools`值是过滤器，用于将发现的资源集限制为该存储后端可用的资源，并且可以以任意组合指定。完全限定名称遵循以下格式：

类型	格式
资源组	<资源组>
NetApp账户	<资源组>/<NetApp 帐户>
容量池	<资源组>/<NetApp 帐户>/<容量池>

类型	格式
虚拟网络	<资源组>/<虚拟网络>
子网	<资源组>/<虚拟网络>/<子网>

卷配置

您可以通过在配置文件的特定部分中指定以下选项来控制默认卷配置。参考 [示例配置] 了解详情。

参数	描述	默认
`exportRule`	新卷的出口规则。 `exportRule`必须是以逗号分隔的 IPv4 地址或 IPv4 子网的任意组合列表，采用 CIDR 表示法。对于 SMB 卷，此设置将被忽略。	“0.0.0.0/0”
`snapshotDir`	控制 .snapshot 目录的可见性	NFSv4 为“true”，NFSv3 为“false”。
`size`	新卷的默认大小	100G
`unixPermissions`	新卷的 Unix 权限（4 位八进制数字）。对于 SMB 卷，此设置将被忽略。	（预览功能，需订阅并加入白名单）

示例配置

以下示例展示了基本配置，其中大多数参数都保留默认值。这是定义后端最简单的方法。

最小配置

这是最基本的后端配置。通过此配置， Trident会发现配置位置中所有委派给Azure NetApp Files存储的NetApp帐户、容量池和子网，并将新卷随机放置在其中一个池和子网上。因为 `nasType`省略了 `nfs`默认设置生效，后端将为 NFS 卷进行配置。

如果您刚开始使用Azure NetApp Files并进行尝试，此配置是理想的选择，但在实践中，您需要为预配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
```

为 **AKS** 管理身份

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

```
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
```

AKS 的云身份

此后端配置省略了 `tenantID`， `clientID` ， 和 `clientSecret`在使用云身份时，这些是可选的。

```
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
```

具有容量池过滤器的特定服务级别配置

此后端配置将卷放置在 Azure 中 `eastus`位置 `Ultra`容量池。 Trident会自动发现该位置中委派给Azure NetApp Files的所有子网，并随机在其中一个子网上放置一个新卷。

```
---
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

此后端配置将卷放置在 Azure 中 `eastus`具有手动 QoS 容量池的位置。* NetApp Trident 25.06 中的技术预览*。

```
---
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

此后端配置进一步缩小了卷放置范围到单个子网，并且还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: my-virtual-network
subnet: 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了多个存储池。当您有多个容量池支持不同的服务级别，并且想要在 Kubernetes 中创建代表这些级别的存储类时，这将非常有用。虚拟池标签用于根据以下因素区分池子：`performance` 。

```
---
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:
      - ultra-1
      - ultra-2
    networkFeatures: Standard
  - labels:
      performance: silver
    serviceLevel: Premium
    capacityPools:
      - premium-1
  - labels:
      performance: bronze
    serviceLevel: Standard
    capacityPools:
      - standard-1
      - standard-2
```

支持的拓扑配置

Trident可根据区域和可用区为工作负载提供卷。这 `supportedTopologies`此后端配置中的块用于提供每个后端的区域和区域列表。此处指定的区域和区域值必须与每个 Kubernetes 集群节点上的标签中的区域和区域值相匹配。这些区域和分区代表存储类中可以提供的允许值的列表。对于包含后端提供的区域和可用区子集的存储类， Trident会在所述区域和可用区中创建卷。更多信息，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

下列 `StorageClass`上述定义指的是存储池。

使用示例定义 **`parameter.selector`**场地

使用 `parameter.selector`你可以为每个对象指定。 `StorageClass`用于托管卷的虚拟池。该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卷的示例定义

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

默认命名空间上的基本配置

```
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: "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`筛选支持 SMB 卷的存储池。 `nasType: nfs`或者 `nasType: null`NFS池过滤器。

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>
```

如果后端创建失败，则后端配置存在问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在您发现并纠正配置文件中的问题后，您可以再次运行创建命令。

Google Cloud NetApp Volumes

配置Google Cloud NetApp Volumes后端

现在您可以将Google Cloud NetApp Volumes配置为Trident的后端。您可以使用Google Cloud NetApp Volumes后端来附加 NFS 和 SMB 卷。

Google Cloud NetApp Volumes驱动程序详情

Trident提供 `google-cloud-netapp-volumes`驱动程序与集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

驱动程序	协议	音量模式	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`

GKE 的云身份

云身份使 Kubernetes pod 能够通过作为工作负载身份进行身份验证来访问 Google Cloud 资源，而无需提供显式的 Google Cloud 凭据。

要在 Google Cloud 中利用云身份功能，您必须具备以下条件：

- 使用 GKE 部署的 Kubernetes 集群。
- 在 GKE 集群上配置工作负载标识，并在节点池上配置 GKE 元数据服务器。
- 具有Google Cloud NetApp Volumes管理员 (roles/netapp.admin) 角色或自定义角色的 GCP 服务帐户。
- Trident已安装，其中包括 cloudProvider（指定“GCP”）和 cloudIdentity（指定新的 GCP 服务帐户）。下面给出一个例子。

Trident操作员

要使用Trident操作员安装Trident，请编辑 `tridentorchestrator_cr.yaml`设置 `cloudProvider`到 `"GCP"`并设置 `cloudIdentity`到 `iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com`。

例如：

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

舵

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

```
export CP="GCP"
export ANNOTATION="'iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com'"
```

以下示例安装Trident并进行设置 `cloudProvider`使用环境变量连接到 `GCP` `$CP`并设置 `cloudIdentity`使用环境变量 `$ANNOTATION`：

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

<code>tridentctl</code>

请使用以下环境变量设置 云提供商 和 云身份 标志的值：

```
export CP="GCP"
export ANNOTATION="'iam.gke.io/gcp-service-account: cloudvolumes-admin-sa@mygcpproject.iam.gserviceaccount.com'"
```

以下示例安装Trident并进行设置 `cloud-provider`标记 `$CP`，和 `cloud-identity`到 `$ANNOTATION`：

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

准备配置Google Cloud NetApp Volumes后端

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

NFS 卷的先决条件

如果您是第一次使用Google Cloud NetApp Volumes或在新的位置使用，则需要进行一些初始配置来设置Google Cloud NetApp Volumes并创建 NFS 卷。参考"开始之前"。

配置Google Cloud NetApp Volumes后端之前，请确保您已具备以下条件：

- 已配置Google Cloud NetApp Volumes服务的 Google Cloud 帐户。参考"Google Cloud NetApp Volumes"。
- 您的 Google Cloud 帐户项目编号。参考"确定项目"。
- 拥有NetApp Volumes 管理员权限的 Google Cloud 服务帐户(`roles/netapp.admin`) 角色。参考"身份和访问管理角色和权限"。
- GCNV账户的API密钥文件。请参阅"创建服务帐户密钥"
- 储水池。参考"存储池概览"。

有关如何设置对Google Cloud NetApp Volumes 的访问权限的更多信息，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

每个后端都在单个 Google Cloud 区域中配置卷。要在其他区域创建卷，您可以定义其他后端。

参数	描述	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	价值 `storageDriverName` 必须指定为“google-cloud-netapp-volumes”。
`backendName`	（可选）存储后端自定义名称	驱动程序名称 + "_" + API 密钥的一部分
`storagePools`	用于指定卷创建存储池的可选参数。	
`projectNumber`	Google Cloud 帐户项目编号。该值可在 Google Cloud 门户网站首页找到。	
`location`	Trident创建 GCNV 卷的 Google Cloud 位置。创建跨区域 Kubernetes 集群时，在以下位置创建的卷：`location`可用于跨多个 Google Cloud 区域的节点上调度的工作负载。跨区域运输会产生额外费用。	

参数	描述	默认
apiKey	用于 Google Cloud 服务帐户的 API 密钥 netapp.admin`角色。它包含 Google Cloud 服务帐户私钥文件的 JSON 格式内容（原封不动地复制到后端配置文件中）。这 `apiKey`必须包含以下键的键值对： `type ， project_id ， client_email ， client_id ， auth_uri ， token_uri ， auth_provider_x509_cert_url ， 和 client_x509_cert_url。	
nfsMountOptions	对 NFS 挂载选项进行精细控制。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小大于此值，则配置失败。	（默认情况下不强制执行）
serviceLevel	存储池的服务级别及其容量。这些值是 flex， standard ， premium ， 或者 extreme。	
labels	要应用于卷的任意 JSON 格式标签集	""
network	Google Cloud 网络用于 GCNV 卷。	
debugTraceFlags	故障排除时要使用的调试标志。例子， {"api":false, "method":true} 。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	无效的
nasType	配置 NFS 或 SMB 卷的创建。选项有 nfs， `smb`或空值。设置为 null 则默认使用 NFS 卷。	nfs
supportedTopologies	表示此后端支持的区域和区域列表。更多信息，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

您可以控制默认卷配置 `defaults`配置文件部分。

参数	描述	默认
exportRule	新卷的出口规则。必须是以逗号分隔的 IPv4 地址列表，地址可以任意组合。	“0.0.0.0/0”
snapshotDir	访问 `.snapshot`目录	NFSv4 为“true”，NFSv3 为“false”。
snapshotReserve	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）。	""

以下示例展示了基本配置，其中大多数参数都保留默认值。这是定义后端最简单的方法。

最小配置

这是最基本的后端配置。通过这种配置， Trident会发现您已委派给配置位置中Google Cloud NetApp Volumes的所有存储池，并随机将新卷放置在其中一个存储池上。因为 `nasType`省略了 `nfs`默认设置生效，后端将为 NFS 卷进行配置。

如果您刚开始使用Google Cloud NetApp Volumes并进行尝试，这种配置是理想的，但在实践中，您很可能需要为配置的卷提供额外的范围。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec
  private_key: |
    -----BEGIN PRIVATE KEY-----\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m\n
    XsYg6gyxy4zq7OlwWgLwGa==\n
    -----END PRIVATE KEY-----\n

---
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
```

使用存储池过滤器进行配置

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----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了多个虚拟池。虚拟池在以下位置定义： `storage`部分。当您有多个支持不同服务级别的存储池，并且想要在 Kubernetes 中创建代表这些存储级别的存储类时，它们非常有用。虚拟池标签用于区分不同的池子。例如，在下面的例子中 `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。在下面的例子中，`snapshotReserve`和 `exportRule`作为所有虚拟池的默认值。

更多信息，请参阅"虚拟池"。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: f2cb6ed6d7cc10c453f7d3406fc700c5df0ab9ec
  private_key: |
    -----BEGIN PRIVATE KEY-----
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    znHczZsrrtHisIsAbOguSaPIKeyAZNchRAGzlzZE4jK3bl/qp8B4Kws8zX5ojY9m
    XsYg6gyxy4zq7OlwWgLwGa==
    -----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
```

GKE 的云身份

```
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可根据区域和可用区为工作负载提供卷。这 `supportedTopologies`此后端配置中的块用于提供每个后端的区域和区域列表。此处指定的区域和区域值必须与每个 Kubernetes 集群节点上的标签中的区域和区域值相匹配。这些区域和分区代表存储类中可以提供的允许值的列表。对于包含后端提供的区域和可用区子集的存储类， Trident会在所述区域和可用区中创建卷。更多信息，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>
```

要验证后端是否创建成功，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，则后端配置存在问题。您可以使用以下方式描述后端： `kubectl get tridentbackendconfig <backend-name>`运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在您发现并纠正配置文件中的问题后，您可以删除后端并再次运行创建命令。

存储类定义

以下是一个基本内容 `StorageClass`上述后端所指的定义。

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

使用以下示例定义 **`parameter.selector`**场地：

使用 `parameter.selector`你可以为每个对象指定。 `StorageClass`这"虚拟池"用于托管卷。该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卷的示例定义

使用 `nasType`， `node-stage-secret-name` ， 和 `node-stage-secret-namespace`您可以指定 SMB 卷并提供所需的 Active Directory 凭据。任何 Active Directory 用户/密码，无论拥有任何权限或没有权限，都可以用作节点阶段密钥。

默认命名空间上的基本配置

```
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: "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`筛选支持 SMB 卷的存储池。 `nasType: nfs`或者 `nasType: null`NFS池过滤器。

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 后端配置Cloud Volumes Service

了解如何使用提供的示例配置，将NetApp Cloud Volumes Service for Google Cloud 配置为Trident安装的后端。

Google Cloud 驱动程序详情

Trident提供 `gcp-cvs`驱动程序与集群通信。支持的访问模式有：*ReadWriteOnce* (RWO)、*ReadOnlyMany* (ROX)、*ReadWriteMany* (RWX)、*ReadWriteOncePod* (RWOP)。

驱动程序	协议	音量模式	支持的访问模式	支持的文件系统
`gcp-cvs`	NFS	Filesystem	RWO、ROX、RWX、RWOP	`nfs`

了解Trident对 Google Cloud Cloud Volumes Service 的支持

Trident可以在以下两种格式之一中创建Cloud Volumes Service卷："服务类型"：

- **CVS-Performance**： Trident 的默认服务类型。这种性能优化型服务类型最适合重视性能的生产工作负载。CVS-Performance 服务类型是一种硬件选项，支持最小 100 GiB 大小的卷。您可以选择其中之一"三个服务级别"：
  - `standard`

- premium
- extreme

- **CVS**：CVS 服务类型提供较高的区域可用性，但性能水平有限或中等。 CVS 服务类型是一种软件选项，它使用存储池来支持小至 1 GiB 的卷。存储池最多可包含 50 个卷，所有卷共享池的容量和性能。您可以选择其中之一"两种服务级别"：
  - standardsw
  - zoneredundantstandardsw

你需要什么

配置和使用 "适用于 Google Cloud 的Cloud Volumes Service"后端需要以下组件：

- 已配置NetApp Cloud Volumes Service的 Google Cloud 帐户
- 您的 Google Cloud 帐户的项目编号
- 拥有 Google Cloud 服务帐户 `netappcloudvolumes.admin`角色
- 您的Cloud Volumes Service帐户的 API 密钥文件

后端配置选项

每个后端都在单个 Google Cloud 区域中配置卷。要在其他区域创建卷，您可以定义其他后端。

参数	描述	默认
version		始终为 1
storageDriverName	存储驱动程序的名称	"gcp-cvs"
backendName	自定义名称或存储后端	驱动程序名称 + "_" + API 密钥的一部分
storageClass	用于指定 CVS 服务类型的可选参数。使用 software`选择 CVS 服务类型。否则， Trident 会假定为 CVS-Performance 服务类型（`hardware）。	
storagePools	仅限CVS服务类型。用于指定卷创建存储池的可选参数。	
projectNumber	Google Cloud 帐户项目编号。该值可在 Google Cloud 门户网站首页找到。	
hostProjectNumber	如果使用共享 VPC 网络，则必须执行此操作。在这种情况下， `projectNumber`这是一个服务项目，而且 `hostProjectNumber`是宿主项目。	
apiRegion	Trident创建Cloud Volumes Service卷的 Google Cloud 区域。创建跨区域 Kubernetes 集群时，在以下位置创建的卷： `apiRegion`可用于跨多个 Google Cloud 区域的节点上调度的工作负载。跨区域运输会产生额外费用。	

参数	描述	默认
apiKey	用于 Google Cloud 服务帐户的 API 密钥 `netappcloudvolumes.admin`角色。它包含 Google Cloud 服务帐户私钥文件的 JSON 格式内容（原封不动地复制到后端配置文件中）。	
proxyURL	如果需要代理服务器才能连接到 CVS 帐户，请提供代理 URL。代理服务器可以是HTTP代理，也可以是HTTPS代理。对于 HTTPS 代理，会跳过证书验证，以允许在代理服务器中使用自签名证书。不支持启用身份验证的代理服务器。	
nfsMountOptions	对 NFS 挂载选项进行精细控制。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小大于此值，则配置失败。	（默认情况下不强制执行）
serviceLevel	CVS-Performance 或 CVS 服务级别（适用于新卷）。CVS-Performance 值是 standard， premium ， 或者 extreme。 CVS 值是 standardsw`或者 `zoneredundantstandardsw。	CVS-Performance 默认值为“标准”。 CVS 默认值为“standardsw”。
network	Google Cloud 网络用于Cloud Volumes Service卷。	“默认”
debugTraceFlags	故障排除时要使用的调试标志。例子，\{"api":false, "method":true} 。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	无效的
allowedTopologies	要启用跨区域访问，您的 StorageClass 定义如下：allowedTopologies`必须包含所有地区。例如：`- key: topology.kubernetes.io/region values: - us-east1 - europe-west1	

卷配置选项

您可以控制默认卷配置 `defaults`配置文件部分。

参数	描述	默认
exportRule	新卷的出口规则。必须是以逗号分隔的 IPv4 地址或 IPv4 子网的列表，采用 CIDR 表示法。	“0.0.0.0/0”
snapshotDir	访问 `.snapshot`目录	"false"
snapshotReserve	快照预留的卷百分比	（接受 CVS 默认值 0）
size	新卷的规模。 CVS-Performance 最低要求为 100 GiB。 CVS 最小容量为 1 GiB。	CVS-Performance 服务类型默认为“100GiB”。 CVS 服务类型不设置默认值，但要求至少 1 GiB。

CVS-Performance 服务类型示例

以下示例提供了 CVS-Performance 服务类型的示例配置。

示例 1：最小配置

这是使用默认 CVS-Performance 服务类型和默认“标准”服务级别的最小后端配置。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: "012345678901"
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: <id_value>
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: "123456789012345678901"
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
```

示例 **2**：服务级别配置

此示例展示了后端配置选项，包括服务级别和卷默认值。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
proxyURL: http://proxy-server-hostname/
nfsMountOptions: vers=3,proto=tcp,timeo=600
limitVolumeSize: 10Ti
serviceLevel: premium
defaults:
  snapshotDir: 'true'
  snapshotReserve: '5'
  exportRule: 10.0.0.0/24,10.0.1.0/24,10.0.2.100
  size: 5Ti
```

示例 **3**：虚拟池配置

此示例使用 `storage`配置虚拟池和 `StorageClasses`指的是他们。请参阅[存储类定义]查看存储类的定义方式。

这里为所有虚拟池设置了特定的默认值，这些默认值决定了： `snapshotReserve` 5%和 `exportRule` 至 `0.0.0.0/0`。虚拟池在以下位置定义： `storage`部分。每个虚拟池都定义了自己的规则。 `serviceLevel`并且有些池会覆盖默认值。虚拟池标签用于根据以下因素区分池子： `performance` 和 `protection`。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
apiRegion: us-west2
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
nfsMountOptions: vers=3,proto=tcp,timeo=600
defaults:
  snapshotReserve: '5'
  exportRule: 0.0.0.0/0
labels:
  cloud: gcp
region: us-west2
storage:
- labels:
    performance: extreme
    protection: extra
  serviceLevel: extreme
  defaults:
```

```
    snapshotDir: 'true'
    snapshotReserve: '10'
    exportRule: 10.0.0.0/24
- labels:
    performance: extreme
    protection: standard
  serviceLevel: extreme
- labels:
    performance: premium
    protection: extra
  serviceLevel: premium
  defaults:
    snapshotDir: 'true'
    snapshotReserve: '10'
- labels:
    performance: premium
    protection: standard
  serviceLevel: premium
- labels:
    performance: standard
  serviceLevel: standard
```

存储类定义

以下 StorageClass 定义适用于虚拟池配置示例。使用 `parameters.selector`您可以为每个 StorageClass 指定用于托管卷的虚拟池。该卷将具有所选池中定义的方面。

存储类示例

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-extra-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=extreme; protection=extra
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-extreme-standard-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium; protection=standard
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium-extra-protection
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium; protection=extra
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-premium
provisioner: csi.trident.netapp.io
parameters:
  selector: performance=premium; protection=standard
allowVolumeExpansion: true
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cvs-standard
provisioner: csi.trident.netapp.io
parameters:
```

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

- 第一个存储类(`cvs-extreme-extra-protection`) 映射到第一个虚拟池。这是唯一一个提供极致性能且快照储备为 10% 的存储池。
- 最后一个存储类(`cvs-extra-protection`) 调用任何提供 10% 快照保留的存储池。 Trident决定选择哪个虚拟池，并确保满足快照储备要求。

CVS 服务类型示例

以下示例提供了 CVS 服务类型的示例配置。

示例 **1**：最小配置

这是使用最简后端配置 `storageClass`指定 CVS 服务类型和默认值 `standardsw`服务水平。

```
---
version: 1
storageDriverName: gcp-cvs
projectNumber: '012345678901'
storageClass: software
apiRegion: us-east4
apiKey:
  type: service_account
  project_id: my-gcp-project
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@my-gcp-
project.iam.gserviceaccount.com
  client_id: '123456789012345678901'
  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/cloudvolumes-admin-
sa%40my-gcp-project.iam.gserviceaccount.com
serviceLevel: standardsw
```

示例 **2**：存储池配置

此示例后端配置使用 `storagePools`配置存储池。

```
---
version: 1
storageDriverName: gcp-cvs
backendName: gcp-std-so-with-pool
projectNumber: '531265380079'
apiRegion: europe-west1
apiKey:
  type: service_account
  project_id: cloud-native-data
  private_key_id: "<id_value>"
  private_key: |-
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
  client_email: cloudvolumes-admin-sa@cloud-native-
data.iam.gserviceaccount.com
  client_id: '107071413297115343396'
  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/cloudvolumes-admin-
sa%40cloud-native-data.iam.gserviceaccount.com
storageClass: software
zone: europe-west1-b
network: default
storagePools:
- 1bc7f380-3314-6005-45e9-c7dc8c2d7509
serviceLevel: Standardsw
```

下一步是什么?

创建后端配置文件后，运行以下命令：

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

如果后端创建失败，则后端配置存在问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在您发现并纠正配置文件中的问题后，您可以再次运行创建命令。

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。

元素驱动程序详情

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

这 `solidfire-san`存储驱动程序支持_文件_和_块_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	音量模式	支持的访问模式	支持的文件系统
`solidfire-san`	iSCSI	块	RWO、ROX、RWX、RWOP	没有文件系统。原始块设备。
`solidfire-san`	iSCSI	Filesystem	RWO，RWOP	`xfs`， `ext3`，`ext4`

开始之前

创建 Element 后端之前，您需要以下内容。

- 一个支持运行 Element 软件的存储系统。
- 拥有NetApp HCI/ SolidFire集群管理员或租户用户权限，可以管理卷。
- 所有 Kubernetes 工作节点都应该安装相应的 iSCSI 工具。参考"工作节点准备信息"。

后端配置选项

请参阅下表了解后端配置选项：

参数	描述	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	始终是“solidfire-san”
`backendName`	自定义名称或存储后端	"solidfire_" + 存储 (iSCSI) IP 地址
`Endpoint`	针对SolidFire集群的 MVIP，包含租户凭证	
`SVIP`	存储（iSCSI）IP 地址和端口	

参数	描述	默认
`labels`	要应用于卷的任意 JSON 格式标签集。	""
`TenantName`	要使用的租户名称（如果找不到则创建）	
`InitiatorIFace`	将 iSCSI 流量限制到特定主机接口	“默认”
`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`	如果请求的卷大小超过此值，则配置失败。	（默认情况下不强制执行）
`debugTraceFlags`	故障排除时要使用的调试标志。例如，{"api":false, "method":true}	无效的

请勿使用 `debugTraceFlags`除非您正在进行故障排除并且需要详细的日志转储。

示例 **1**：后端配置 **`solidfire-san`**具有三种音量类型的驱动器

本示例展示了一个使用 CHAP 认证的后端文件，并对三种具有特定 QoS 保证的卷类型进行了建模。最有可能的情况是，您会定义存储类来使用它们中的每一个。 `IOPS`存储类参数。

```
---
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：后端和存储类配置 **`solidfire-san`**带虚拟池的驱动程序

此示例显示了配置了虚拟池的后端定义文件以及引用这些虚拟池的存储类。

Trident在配置时将存储池中存在的标签复制到后端存储 LUN。为了方便起见，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。

在下方所示的示例后端定义文件中，所有存储池都设置了特定的默认值，这些默认值决定了： `type`银级。虚拟池在以下位置定义： `storage`部分。在这个例子中，一些存储池设置了自己的类型，而一些存储池则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 定义与上述虚拟池相关。使用 `parameters.selector`字段中，每个 StorageClass 都会指定哪些虚拟池可用于托管卷。卷将具有所选虚拟池中定义的各个方面。

第一个存储类(`solidfire-gold-four`将映射到第一个虚拟池。这是唯一一家提供黄金级性能的泳池。

`Volume Type QoS`黄金。最后一个存储类（`solidfire-silver`）指出任何提供银级性能的存储池。Trident将决定选择哪个虚拟池，并确保满足存储需求。

```
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 驱动程序详情

Trident提供以下 SAN 存储驱动程序，用于与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	音量模式	支持的访问模式	支持的文件系统
`ontap-san`	iSCSI 通过光纤通道提供 SCSI 服务	块	RWO、ROX、RWX、RWOP	无文件系统；原始块设备
`ontap-san`	iSCSI 通过光纤通道提供 SCSI 服务	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 参考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 参考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`

驱动程序	协议	音量模式	支持的访问模式	支持的文件系统
`ontap-san-economy`	iSCSI	块	RWO、ROX、RWX、RWOP	无文件系统；原始块设备
`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`

- 使用 `ontap-san-economy`仅当预计持续卷使用量高于……时"支持的ONTAP容量限制"。
- 使用 `ontap-nas-economy`仅当预计持续卷使用量高于……时"支持的ONTAP容量限制"以及 `ontap-san-economy`驱动程序无法使用。
- 请勿使用 `ontap-nas-economy`如果您预计需要数据保护、灾难恢复或移动办公。
- NetApp不建议在所有ONTAP驱动程序中使用 Flexvol 自动增长，ontap-san 除外。作为一种变通方法， Trident支持使用快照储备，并相应地调整 Flexvol 容量。

用户权限

Trident预期以ONTAP或 SVM 管理员身份运行，通常使用以下方式： `admin`集群用户或 `vsadmin`SVM 用户，或者具有相同角色但名称不同的用户。对于Amazon FSx for NetApp ONTAP部署， Trident需要以ONTAP或 SVM 管理员身份运行，并使用集群。 `fsxadmin`用户或 `vsadmin`SVM 用户，或者具有相同角色但名称不同的用户。这 `fsxadmin`用户是集群管理员用户的有限替代品。

如果你使用 `limitAggregateUsage`需要参数和集群管理员权限。当使用Amazon FSx for NetApp ONTAP和Trident时， `limitAggregateUsage`参数将无法与 `vsadmin`和 `fsxadmin`用户账户。如果指定此参数，配置操作将失败。

虽然可以在ONTAP中创建一个Trident驱动程序可以使用的限制性更强的角色，但我们不建议这样做。 Trident的大多数新版本都会调用额外的 API，这些 API 必须加以考虑，这使得升级变得困难且容易出错。

NVMe/TCP 的其他注意事项

Trident支持使用非易失性存储器高速接口 (NVMe) 协议 `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外部创建的 NVMe 卷，以便Trident可以管理其生命周期。
- NVMe原生多路径
- K8s节点的优雅关闭或非优雅关闭（24.06）

Trident不支持：

- NVMe 原生支持的 DH-HMAC-CHAP
- 设备映射器 (DM) 多路径

- LUKS 加密

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

准备配置后端**ONTAP SAN** 驱动程序

了解配置ONTAP后端和ONTAP SAN 驱动程序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。

"ASA r2 系统"与其他ONTAP系统（ASA、 AFF和FAS）在存储层的实现上有所不同。在ASA r2 系统中，使用存储可用区而不是聚合。请参阅"这"知识库文章，介绍如何在ASA r2 系统中将聚合分配给 SVM。

请记住，您还可以运行多个驱动程序，并创建指向其中一个或另一个驱动程序的存储类。例如，您可以配置一个 `san-dev`使用类 `ontap-san`司机和 `san-default`使用类 `ontap-san-economy`一。

所有 Kubernetes 工作节点都必须安装相应的 iSCSI 工具。参考 "准备工作节点" 了解详情。

对**ONTAP**后端进行身份验证

Trident提供两种ONTAP后端身份验证方式。

- 基于凭证：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`或者 `vsadmin`确保与ONTAP版本最大程度兼容。
- 基于证书： Trident还可以使用安装在后端的证书与ONTAP集群通信。此处，后端定义必须包含客户端证书、密钥和受信任 CA 证书（如果使用，建议使用）的 Base64 编码值。

您可以更新现有后端，以在基于凭据的方法和基于证书的方法之间进行切换。但是，一次只能支持一种身份验证方法。要切换到不同的身份验证方法，必须从后端配置中删除现有方法。

如果您尝试同时提供凭据和证书，则后端创建将失败，并出现错误，提示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 SVM 范围/集群范围管理员的凭据才能与ONTAP后端通信。建议使用标准的、预定义的角色，例如： `admin`或者 `vsadmin`。这样可以确保与未来ONTAP版本向前兼容，这些版本可能会公开一些功能 API，供未来的Trident版本使用。虽然可以创建自定义安全登录角色并将其与Trident一起使用，但不建议这样做。

后端定义示例如下所示：

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"
}
```

请注意，后端定义是唯一以纯文本形式存储凭据的地方。后端创建完成后，用户名/密码将使用 Base64 进行编码，并存储为 Kubernetes 密钥。只有创建或更新后端时才需要了解凭据。因此，这是一项仅限管理员执行的操作，由 Kubernetes/存储管理员执行。

启用基于证书的身份验证

新的和现有的后端都可以使用证书与ONTAP后端通信。后端定义需要三个参数。

- clientCertificate：客户端证书的 Base64 编码值。
- clientPrivateKey：关联私钥的 Base64 编码值。
- trustedCACertificate：受信任 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。如果没有使用受信任的证书颁发机构，则可以忽略此步骤。

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，将通用名称 (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. 向ONTAP集群添加受信任的 CA 证书。这可能已经由存储管理员处理了。如果没有使用受信任的证书颁发机构，则忽略此操作。

```
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
```

> 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. 使用生成的证书进行身份验证。请将 < ONTAP管理 LIF> 和 <vserver 名称> 替换为管理 LIF IP 地址和 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、密钥和受信任的 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。这种方法是双向的：使用用户名/密码的后端可以更新为使用证书；使用证书的后端可以更新为基于用户名/密码的后端。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。然后使用包含所需参数的更新后的 backend.json 文件来执行 `tridentctl backend update`。

```
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 |
+------------+----------------+--------------------------------------
+--------+---------+
```

轮换密码时，存储管理员必须首先更新ONTAP上用户的密码。接下来将进行后端更新。轮换证书时，可以为用户添加多个证书。然后更新后端以使用新证书，之后即可从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。后端更新成功表明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

您可以创建一个具有最低权限的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将使用您创建的ONTAP集群角色来执行操作。

请参阅"Trident自定义角色生成器"有关创建Trident自定义角色的更多信息。

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\>
```

使用系统管理器

在ONTAP系统管理器中执行以下步骤：

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“集群 > 设置”。

      （或者）要在 SVM 级别创建自定义角色，请选择“存储”>“存储虚拟机”> `required SVM` > 设置 > 用户和角色*。

   b. 选择“用户和角色”旁边的箭头图标（→）。
   c. 在“角色”下选择“+添加”。
   d. 定义角色规则，然后点击“保存”。
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. 在“用户”下方选择“添加”图标 **+**。
   b. 选择所需的用户名，然后在“角色”下拉菜单中选择角色。
   c. 单击“保存”。

更多信息请参阅以下页面：

- "用于管理ONTAP的自定义角色"或者"定义自定义角色"
- "与角色和用户协作"

使用双向 **CHAP** 验证连接

Trident可以使用双向 CHAP 对 iSCSI 会话进行身份验证。 `ontap-san`和 `ontap-san-economy`司机。这需要启用 `useCHAP`在后端定义中添加选项。设置为 `true`Trident将 SVM 的默认发起程序安全配置为双向 CHAP，并从后端文件中设置用户名和密钥。 NetApp建议使用双向 CHAP 协议对连接进行身份验证。请参见以下示例配置：

```
---
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
```

这 `useCHAP`该参数是一个布尔选项，只能配置一次。默认值为 false。一旦将其设置为 true，就无法再将其设置为 false。

此外 `useCHAP=true`， 这 `chapInitiatorSecret`， `chapTargetInitiatorSecret` ， `chapTargetUsername` ， 和 `chapUsername`字段必须包含在后端定义中。创建后端后，可以通过运行以下命令来更改密钥： `tridentctl update`。

工作原理

通过设置 `useCHAP`如果设置为 true，则存储管理员指示Trident在存储后端配置 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“无”（默认设置）*并且*卷中不存在任何预先存在的 LUN，Trident会将默认安全类型设置为“无”。 `CHAP`然后继续配置 CHAP 发起程序和目标用户名及密钥。
  - 如果 SVM 包含 LUN， Trident将不会在 SVM 上启用 CHAP。这样可以确保对 SVM 上已存在的 LUN 的访问不受限制。
- 配置 CHAP 发起程序和目标用户名和密钥；这些选项必须在后端配置中指定（如上所示）。

后端创建完成后， Trident会创建一个相应的实例。 `tridentbackend` CRD 并将 CHAP 密钥和用户名存储为 Kubernetes 密钥。 Trident在此后端创建的所有 PV 都将通过 CHAP 进行安装和连接。

轮换凭证并更新后端

您可以通过更新 CHAP 参数来更新 CHAP 凭据。 `backend.json`文件。这将需要更新 CHAP 密钥并使用 `tridentctl update`命令反映这些更改。

更新后端 CHAP 密钥时，必须使用 `tridentctl`更新后端。请勿使用ONTAP CLI 或ONTAP系统管理器更新存储集群上的凭据，因为Trident将无法检测到这些更改。

```
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 上更新凭据，则这些连接将继续保持活动状态。新连接使用更新后的凭据，现有连接将继续保持活动状态。断开并重新连接旧的PV将使它们使用更新后的凭据。

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。本节提供后端配置示例以及将后端映射到 StorageClasses 的详细信息。

"ASA r2 系统"与其他ONTAP系统（ASA、 AFF和FAS）在存储层的实现上有所不同。这些变化会影响某些参数的使用，如注释中所述。"了解更多关于ASA r2 系统与其他ONTAP系统之间的区别"。

只有 `ontap-san`ASA r2 系统支持驱动程序（支持 iSCSI 和 NVMe/TCP 协议）。

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`Trident可自动检测ASA r2 或传统的ONTAP系统。如下表所示，某些后端配置参数不适用于ASA r2 系统。

请参阅下表了解后端配置选项：

参数	描述	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`ontap-san`或者 `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。 可以指定一个完全限定域名（FQDN）。 如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]` 。 为了实现MetroCluster 的无缝切换，请参阅MetroCluster示例。 ⓘ 如果您使用的是“vsadmin”凭据，`managementLIF`必须是SVM的凭据；如果使用“admin”凭据，`managementLIF`必须是集群的那个。	"10.0.0.1", "[2001:1234:abcd::fefe]"
`dataLIF`	LIF协议的IP地址。如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]` 。 *请勿指定使用 iSCSI。*Trident的使用"ONTAP选择性 LUN 地图"发现建立多路径会话所需的 iSCSI LIF。如果出现以下情况，则会生成警告： `dataLIF`已明确定义。 *Metrocluster 除外。*查看MetroCluster示例。	由支持向量机导出
`svm`	要使用的存储虚拟机 *Metrocluster 除外。*查看MetroCluster示例。	如果是 SVM 则推导而来 `managementLIF`已指定
`useCHAP`	使用 CHAP 对ONTAP SAN 驱动程序的 iSCSI 进行身份验证 [布尔值]。设置为 `true`让Trident配置并使用双向 CHAP 作为后端给定 SVM 的默认身份验证。参考 "准备配置后端ONTAP SAN 驱动程序" 了解详情。 不支持**FCP**或**NVMe/TCP**协议。	`false`
`chapInitiatorSecret`	CHAP 发起者密钥。如果是必填项 `useCHAP=true`	""
`labels`	要应用于卷的任意 JSON 格式标签集	""

参数	描述	默认
chapTargetInitiatorSecret	CHAP 目标发起者密钥。如果是必填项 useCHAP=true	""
chapUsername	入站用户名。如果是必填项 useCHAP=true	""
chapTargetUsername	目标用户名。如果是必填项 useCHAP=true	""
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。用于基于凭证的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	""
password	与ONTAP集群通信所需的密码。用于基于凭证的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	""
svm	使用的存储虚拟机	如果是 SVM 则推导而来 `managementLIF`已指定
storagePrefix	在 SVM 中配置新卷时使用的前缀。之后无法修改。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。 当 SVM 中的聚合数据更新时， Trident 会自动轮询 SVM 进行更新，而无需重启Trident控制器。当您在Trident中配置特定聚合以配置卷时，如果该聚合被重命名或移出 SVM，则在轮询 SVM 聚合时， Trident中的后端将变为失败状态。您必须将聚合更改为 SVM 上存在的聚合，或者将其完全删除，才能使后端恢复联机。 请勿指定用于**ASA r2** 系统。	""

参数	描述	默认
`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，请勿指定 `limitAggregateUsage`。提供的 `fsxadmin`和 `vsadmin`不包含检索汇总使用情况和使用Trident限制它所需的权限。 请勿指定用于**ASA r2** 系统。	（默认情况下不强制执行）
`limitVolumeSize`	如果请求的卷大小大于此值，则配置失败。同时限制其管理的 LUN 卷的最大大小。	（默认情况下不强制执行）
`lunsPerFlexvol`	每个 Flexvol 的最大 LUN 数量必须在 [50, 200] 范围内	`100`
`debugTraceFlags`	故障排除时要使用的调试标志。例如，{"api":false, "method":true} 除非您正在进行故障排除并且需要详细的日志转储，否则请勿使用此方法。	`null`
`useREST`	使用ONTAP REST API 的布尔参数。 `useREST`设置为 `true`Trident 使用ONTAP REST API 与后端通信；当设置为 `false`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登录角色必须具有访问权限。 `ontapi`应用。预定义项满足了这一点。 `vsadmin`和 `cluster-admin` 角色。从Trident 24.06 版本和ONTAP 9.15.1 或更高版本开始， `useREST`设置为 `true`默认；更改 `useREST`到 `false`使用 ONTAPI (ZAPI) 调用。 `useREST`完全符合 NVMe/TCP 标准。 NVMe 仅支持ONTAP REST API，不支持 ONTAPI (ZAPI)。 如果指定，则始终设置为 **`true`**适用于**ASA r2** 系统。	`true`适用于ONTAP 9.15.1 或更高版本，否则 `false`。
`sanType`	用于选择 `iscsi`对于 iSCSI， `nvme`适用于 NVMe/TCP 或 `fcp`用于光纤通道 (FC) 上的 SCSI。	`iscsi`如果为空

参数	描述	默认
formatOptions	使用 `formatOptions`为以下情况指定命令行参数 `mkfs`该命令将在每次格式化卷时应用。这样您就可以根据自己的喜好格式化音量。请确保指定与 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**`支持 **iSCSI** 协议的驱动程序。 此外，在使用 **iSCSI** 和 **NVMe/TCP** 协议时， **ASA r2** 系统也受支持。	
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`后端创建新的FlexVol卷来包含它们的 LUN。只有预先存在的 Flexvol 才能用于配置新的 PV。	

使用 **formatOptions** 的建议

Trident建议采用以下选项来加快格式化过程：

-E nodiscard:

- 保留，不要在执行 mkfs 时尝试丢弃块（最初丢弃块对固态设备和稀疏/精简配置存储很有用）。这取代了已弃用的选项“-K”，并且适用于所有文件系统（xfs、ext3 和 ext4）。

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

您可以配置Trident以使用 Active Directory (AD) 凭据对后端 SVM 进行身份验证。在 AD 帐户可以访问 SVM 之前，您必须配置 AD 域控制器对集群或 SVM 的访问权限。对于使用 AD 帐户进行集群管理，您必须创建域隧道。参考 "在ONTAP中配置 Active Directory 域控制器访问" 了解详情。

步骤

1. 为后端 SVM 配置域名系统 (DNS) 设置：

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

2. 运行以下命令在 Active Directory 中为 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 用户或组来管理集群或 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，设置 `username` 和 `password` 参数分别为 AD 用户或组名称和密码。

您可以使用以下选项控制默认配置。 `defaults`配置部分。例如，请参见下面的配置示例。

参数	描述	默认
`spaceAllocation`	LUN 的空间分配	“true” 如果指定，则设置为 **`true`**适用于**ASA r2** 系统。
`spaceReserve`	空间预留模式；“无”（细）或“大量”（粗）。 设置为 **`none`**适用于**ASA r2** 系统。	“没有任何”
`snapshotPolicy`	要使用的快照策略。 设置为 **`none`**适用于**ASA r2** 系统。	“没有任何”
`qosPolicy`	要为创建的卷分配的 QoS 策略组。每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 之一。将 QoS 策略组与Trident结合使用需要ONTAP 9.8 或更高版本。您应该使用非共享的 QoS 策略组，并确保该策略组单独应用于每个成员。共享的 QoS 策略组强制规定所有工作负载的总吞吐量上限。	""
`adaptiveQosPolicy`	要为创建的卷分配的自适应 QoS 策略组。每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 之一	""
`snapshotReserve`	为快照预留的卷百分比。 请勿指定用于**ASA r2** 系统。	如果为“0”， `snapshotPolicy`为“无”，否则为“
`splitOnClone`	创建时将克隆体从其母体中分离出来	"false"
`encryption`	在新卷上启用NetApp卷加密 (NVE)；默认设置为 `false`。要使用此选项，必须在集群上获得 NVE 许可并启用 NVE。如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。更多信息，请参阅：“Trident如何与 NVE 和 NAE 协同工作” 。	“false” 如果指定，则设置为 **`true`**适用于**ASA r2** 系统。
`luksEncryption`	启用LUKS加密。参考“使用 Linux 统一密钥设置 (LUKS)”。	设置为 **`false`**适用于**ASA r2** 系统。
`tieringPolicy`	分层策略使用“无” 请勿为**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

以下是一个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'
```

对于使用以下方式创建的所有卷 `ontap-san`驱动程序Trident为FlexVol增加了 10% 的额外容量，以容纳 LUN 元数据。 LUN 将按照用户在 PVC 中请求的确切大小进行配置。 Trident使FlexVol增加 10%（在ONTAP中显示为可用尺寸）。用户现在将获得他们所申请的可用容量。此项更改还可以防止 LUN 在可用空间未完全利用之前变为只读。这不适用于 ontap-san-economy。

对于定义后端 `snapshotReserve`Trident计算体积大小的方法如下：

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

1.1 是Trident为容纳 LUN 元数据而额外添加到FlexVol 的10%。为了 `snapshotReserve`= 5%，PVC 请求 = 5 GiB，总体积大小为 5.79 GiB，可用大小为 5.5 GiB。这 `volume show`该命令应显示与此示例类似的结果：

目前，调整大小是将新计算方法应用于现有体积的唯一途径。

以下示例展示了基本配置，其中大多数参数都保留默认值。这是定义后端最简单的方法。

> 如果您在NetApp ONTAP上使用Amazon FSx和Trident， NetApp建议您为 LIF 指定 DNS 名称而不是 IP 地址。

ONTAP SAN 示例

这是使用以下方法的基本配置： `ontap-san`司机。

```
---
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，以避免在切换和切换回后端后手动更新后端定义。"SVM复制和恢复" 。

为了实现无缝切换和切换回，请指定 SVM `managementLIF`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

在这个基本配置示例中 `clientCertificate`， `clientPrivateKey`， 和 `trustedCACertificate`（如果使用受信任的 CA，则为可选） `backend.json`分别取客户端证书、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

这些示例创建了一个后端。 `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 经济 **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

您的ONTAP后端必须配置有使用 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
```

SCSI over FC (FCP) 示例

您的ONTAP后端必须配置有 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
```

使用 **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}}"
```

ontap-san-economy 驱动程序的 **formatOptions** 示例

```
---
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，所有存储池都设置了特定的默认值，例如： `spaceReserve`没有，`spaceAllocation`为假，并且 `encryption`为假。虚拟池在存储部分中定义。

Trident在“备注”字段中设置配置标签。在FlexVol volume上设置注释。Trident 在配置时Trident虚拟池上存在的所有标签复制到存储卷。为了方便起见，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，一些存储池会设置自己的参数。 `spaceReserve` ， `spaceAllocation` ， 和 `encryption`有

些值会覆盖默认值，有些池会覆盖默认值。

ONTAP SAN 示例

```yaml
---
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"
```

将后端映射到存储类

以下 StorageClass 定义指的是[具有虚拟池的后端示例]。使用 `parameters.selector`字段中，每个 StorageClass 都会指出哪些虚拟池可用于托管卷。卷将具有所选虚拟池中定义的各个方面。

- 这 `protection-gold`StorageClass 将映射到第一个虚拟池。 `ontap-san`后端。这是唯一提供黄金级保护的泳池。

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

- 这 `protection-not-gold`StorageClass 将映射到第二个和第三个虚拟池。 `ontap-san`后端。除了黄金级别之外，只有这些金池提供其他级别的保护。

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

- 这 `app-mysqldb`StorageClass 将映射到第三个虚拟池 `ontap-san-economy`后端。这是唯一一个为mysqldb类型应用程序提供存储池配置的存储池。

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

- 这 `protection-silver-creditpoints-20k`StorageClass 将映射到第二个虚拟池 `ontap-san`后端。这是唯一提供银级保护和 20000 积分的彩池。

```
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"
```

- 这 `creditpoints-5k`StorageClass 将映射到第三个虚拟池 `ontap-san`后端和第四个虚拟池 `ontap-san-economy`后端。这是唯一提供 5000 积分的彩池产品。

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

- 这 `my-test-app-sc`StorageClass 将映射到 `testAPP`虚拟池 `ontap-san`司机 `sanType: nvme`。这是唯一一家提供泳池的公司 `testApp`。

```
---
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，并确保满足存储需求。

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 驱动程序详情

Trident提供以下 NAS 存储驱动程序，用于与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	音量模式	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`

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

- 使用 `ontap-san-economy`仅当预计持续卷使用量高于……时"支持的ONTAP容量限制"。
- 使用 `ontap-nas-economy`仅当预计持续卷使用量高于……时"支持的ONTAP容量限制"以及 `ontap-san-economy`驱动程序无法使用。
- 请勿使用 `ontap-nas-economy`如果您预计需要数据保护、灾难恢复或移动办公。
- NetApp不建议在所有ONTAP驱动程序中使用 Flexvol 自动增长，ontap-san 除外。作为一种变通方法， Trident支持使用快照储备，并相应地调整 Flexvol 容量。

用户权限

Trident预期以ONTAP或 SVM 管理员身份运行，通常使用以下方式： `admin`集群用户或 `vsadmin`SVM 用户，或者具有相同角色但名称不同的用户。

对于Amazon FSx for NetApp ONTAP部署， Trident需要以ONTAP或 SVM 管理员身份运行，并使用集群。`fsxadmin`用户或 `vsadmin`SVM 用户，或者具有相同角色但名称不同的用户。这 `fsxadmin`用户是集群管理员用户的有限替代品。

如果你使用 `limitAggregateUsage`需要参数和集群管理员权限。当使用Amazon FSx for NetApp ONTAP和Trident时， `limitAggregateUsage`参数将无法与 `vsadmin`和 `fsxadmin`用户账户。如果指定此参数，配置操作将失败。

虽然可以在ONTAP中创建一个Trident驱动程序可以使用的限制性更强的角色，但我们不建议这样做。 Trident的大多数新版本都会调用额外的 API，这些 API 必须加以考虑，这使得升级变得困难且容易出错。

准备配置带有**ONTAP NAS** 驱动程序的后端

了解配置带有ONTAP NAS 驱动程序的ONTAP后端的要求、身份验证选项和导出策略。

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，并创建指向其中一个或另一个驱动程序的存储类。例如，您可以配置一个使用以下方式的 Gold 类： `ontap-nas`驾驶员和使用青铜级的驾驶员 `ontap-nas-economy`一。
- 所有 Kubernetes 工作节点都必须安装相应的 NFS 工具。请参阅"此处"更多详情请见下文。
- Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。参考 准备配置SMB卷 了解详情。

对**ONTAP**后端进行身份验证

Trident提供两种ONTAP后端身份验证方式。

- 基于凭证：此模式需要对ONTAP后端拥有足够的权限。建议使用与预定义的安全登录角色关联的帐户，例如： `admin`或者 `vsadmin`确保与ONTAP版本最大程度兼容。
- 基于证书：此模式要求后端安装证书， Trident才能与ONTAP集群通信。此处，后端定义必须包含客户端证

书、密钥和受信任 CA 证书（如果使用，建议使用）的 Base64 编码值。

您可以更新现有后端，以在基于凭据的方法和基于证书的方法之间进行切换。但是，一次只能支持一种身份验证方法。要切换到不同的身份验证方法，必须从后端配置中删除现有方法。

如果您尝试同时提供凭据和证书，则后端创建将失败，并出现错误，提示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 SVM 范围/集群范围管理员的凭据才能与ONTAP后端通信。建议使用标准的、预定义的角色，例如：`admin`或者 `vsadmin`。这样可以确保与未来ONTAP版本向前兼容，这些版本可能会公开一些功能 API，供未来的Trident版本使用。虽然可以创建自定义安全登录角色并将其与Trident一起使用，但不建议这样做。

后端定义示例如下所示：

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"
    }
}
```

请注意，后端定义是唯一以纯文本形式存储凭据的地方。后端创建完成后，用户名/密码将使用 Base64 进行编码，并存储为 Kubernetes 密钥。后端创建/更新是唯一需要了解凭据的步骤。因此，这是一项仅限管理员执行的操作，由 Kubernetes/存储管理员执行。

启用基于证书的身份验证

新的和现有的后端都可以使用证书与ONTAP后端通信。后端定义需要三个参数。

- clientCertificate：客户端证书的 Base64 编码值。
- clientPrivateKey：关联私钥的 Base64 编码值。
- trustedCACertificate：受信任 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。如果没有使用受信任的证书颁发机构，则可以忽略此步骤。

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，将通用名称 (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. 向ONTAP集群添加受信任的 CA 证书。这可能已经由存储管理员处理了。如果没有使用受信任的证书颁发机构，则忽略此操作。

```
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. 使用生成的证书进行身份验证。请将 < ONTAP管理 LIF> 和 <vserver 名称> 替换为管理 LIF IP 地址和 SVM 名称。您必须确保 LIF 的服务策略已设置为 `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、密钥和受信任的 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。这种方法是双向的：使用用户名/密码的后端可以更新为使用证书；使用证书的后端可以更新为基于用户名/密码的后端。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。然后使用包含所需参数的更新后的 backend.json 文件来执行 `tridentctl update backend`。

```
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 |
+------------+----------------+--------------------------------------
+--------+---------+
```

> 轮换密码时，存储管理员必须首先更新ONTAP上用户的密码。接下来将进行后端更新。轮换证书时，可以为用户添加多个证书。然后更新后端以使用新证书，之后即可从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。后端更新成功表明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

您可以创建一个具有最低权限的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将使用您创建的ONTAP集群角色来执行操作。

请参阅"Trident自定义角色生成器"有关创建Trident自定义角色的更多信息。

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\>
```

使用系统管理器

在ONTAP系统管理器中执行以下步骤：

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“集群 > 设置”。

      （或者）要在 SVM 级别创建自定义角色，请选择“存储”>“存储虚拟机”> `required SVM` > 设置 > 用户和角色*。

   b. 选择“用户和角色”旁边的箭头图标（→）。
   c. 在“角色”下选择“+添加”。
   d. 定义角色规则，然后点击“保存”。
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. 在“用户”下方选择“添加”图标 **+**。
   b. 选择所需的用户名，然后在“角色”下拉菜单中选择角色。
   c. 单击“保存”。

更多信息请参阅以下页面：

- "用于管理ONTAP的自定义角色"或者"定义自定义角色"

- "与角色和用户协作"

管理 **NFS** 导出策略

Trident使用 NFS 导出策略来控制对其所配置卷的访问。

Trident在处理出口策略时提供两种选择：

- Trident可以动态管理导出策略本身；在这种操作模式下，存储管理员指定一个 CIDR 块列表，这些 CIDR 块代表可接受的 IP 地址。 Trident会在发布时自动将属于这些范围内的适用节点 IP 添加到导出策略中。或者，如果没有指定 CIDR，则会将发布卷的节点上找到的所有全局范围的单播 IP 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。除非在配置中指定了不同的导出策略名称，否则Trident将使用默认导出策略。

动态管理出口策略

Trident提供了动态管理ONTAP后端导出策略的功能。这样，存储管理员就可以指定工作节点 IP 的允许地址空间，而无需手动定义明确的规则。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人工干预。此外，这有助于将对存储集群的访问限制在仅允许挂载卷且 IP 地址在指定范围内的节点上，从而支持细粒度和自动化管理。

> 使用动态导出策略时，请勿使用网络地址转换（NAT）。使用 NAT 时，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，因此当在导出规则中找不到匹配项时，访问将被拒绝。

示例

必须使用两种配置选项。以下是一个后端定义示例：

```
---
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
```

> 使用此功能时，必须确保 SVM 中的根连接点具有先前创建的导出策略，该策略的导出规则允许节点 CIDR 块（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。

下面以上述示例为例，解释此功能的工作原理：

- `autoExportPolicy`设置为 `true`。这表明Trident会为使用此后端配置的每个卷创建一个导出策略。`svm1`使用 SVM 处理规则的添加和删除 `autoexportCIDRs`地址块。在卷连接到节点之前，该卷使用空的

导出策略，没有任何规则来防止对该卷的未经授权的访问。当卷发布到节点时， Trident会创建一个导出策略，该策略的名称与包含指定 CIDR 块内节点 IP 的底层 qtree 的名称相同。这些 IP 地址也将添加到父FlexVol volume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FlexVol的 FlexVol 创建了一个导出策略。 `trident-403b5326-8482-40db96d0-d83fb3f4daec` ，名为 qtree 的导出策略 `trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`以及一个名为“空的导出策略”的 `trident_empty`在支持向量机上。 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。任何未附加的卷都将重用空导出策略。

- `autoExportCIDRs`包含地址块列表。此字段为可选字段，默认值为 ["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具有发布的全局作用域的单播地址。

在这个例子中， `192.168.0.0/24`已提供地址空间。这意味着，位于此地址范围内且已发布 Kubernetes 节点 IP 的节点将被添加到Trident创建的导出策略中。当Trident注册它运行所在的节点时，它会检索该节点的 IP 地址，并将其与提供的地址块进行比对。 `autoExportCIDRs`在发布时， Trident在过滤 IP 地址后，会为它要发布到的节点的客户端 IP 地址创建导出策略规则。

您可以更新 `autoExportPolicy`和 `autoExportCIDRs`创建后端之后，需要进行以下操作。您可以为自动管理的后端添加新的 CIDR，或者删除现有的 CIDR。删除 CIDR 时要格外小心，确保现有连接不会断开。您也可以选择禁用 `autoExportPolicy`对于后端，如果出现问题，则回退到手动创建的导出策略。这将需要进行设置 `exportPolicy`后端配置中的参数。

Trident创建或更新后端后，您可以使用以下命令检查后端： `tridentctl`或相应的 `tridentbackend`CRD：

```
./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
```

当一个节点被移除时， Trident会检查所有导出策略，以移除与该节点对应的访问规则。通过从受管后端导出策略中移除此节点 IP， Trident可以防止恶意挂载，除非集群中的新节点重新使用此 IP。

对于先前存在的后端，使用以下方式更新后端： `tridentctl update backend`确保Trident自动管理出口策略。这样，当需要时，就会创建两个以后端 UUID 和 qtree 名称命名的新导出策略。后端存在的卷在卸载并重新挂载后，将使用新创建的导出策略。

> 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。如果后端被重新创建，它将被视为一个新的后端，并将导致创建一个新的导出策略。

如果运行中的节点的 IP 地址更新，则必须重启该节点上的Trident pod。 Trident随后将更新其管理的后端的导出策略，以反映此 IP 变更。

准备配置SMB卷

稍作准备，您就可以使用以下方式配置 SMB 卷 `ontap-nas`司机。

> 您必须在 SVM 上同时配置 NFS 和 SMB/CIFS 协议才能创建 `ontap-nas-economy`适用于ONTAP本地集群的 SMB 卷。如果未能配置这些协议中的任何一个，都会导致 SMB 卷创建失败。

> `autoExportPolicy`不支持 SMB 卷。

开始之前

在配置 SMB 卷之前，您必须具备以下条件。

- 一个 Kubernetes 集群，包含一个 Linux 控制器节点和至少一个运行 Windows Server 2022 的 Windows 工作节点。 Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。
- 至少有一个包含您的 Active Directory 凭据的Trident密钥。生成秘密 `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代理"或者"GitHub：适用于 Windows 的 CSI 代理"适用于在 Windows 上运行的 Kubernetes 节点。

步骤

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

   Amazon FSx for ONTAP需要 SMB 共享。

   您可以通过以下两种方式之一创建 SMB 管理共享："Microsoft 管理控制台"共享文件夹管理单元或使用ONTAP CLI。使用ONTAP CLI 创建 SMB 共享：

   a. 如有必要，请创建共享的目录路径结构。

      这 `vserver cifs share create`该命令检查在创建共享时 -path 选项中指定的路径。如果指定的路径不存在，则命令执行失败。

   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. 确认共享已创建：

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

   请参阅"创建 SMB 共享"了解详细信息。

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

参数	描述	示例
`smbShare`	您可以指定以下一项：使用 Microsoft 管理控制台或ONTAP CLI 创建的 SMB 共享的名称；允许Trident创建 SMB 共享的名称；或者您可以将参数留空以防止对卷的公共共享访问。对于本地部署的ONTAP，此参数是可选的。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`.*如果为空，则默认为空。 `nfs` 。	`smb`
`securityStyle`	新卷的安全样式。 必须设置为 **`ntfs`**或者 **`mixed`**适用于 **SMB** 卷。	`ntfs`或者 `mixed`适用于 SMB 卷
`unixPermissions`	新卷模式。 对于 **SMB** 卷，此项必须留空。	""

启用安全的 **SMB**

从 25.06 版本开始， NetApp Trident支持使用以下方式安全配置创建的 SMB 卷： `ontap-nas`和 `ontap-nas-economy`后端。启用安全 SMB 后，您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

- 输入 `ontap-nas-economy`不支持卷。
- 仅支持只读克隆 `ontap-nas-economy`卷。
- 如果启用了安全 SMB， Trident将忽略后端提到的 SMB 共享。
- 更新 PVC 注解、存储类注解和后端字段不会更新 SMB 共享 ACL。
- 克隆 PVC 注释中指定的 SMB 共享 ACL 将优先于源 PVC 中的 ACL。
- 启用安全 SMB 时，请确保提供有效的 AD 用户。无效用户将不会被添加到访问控制列表 (ACL) 中。
- 如果在后端、存储类和 PVC 中为同一个 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。
- 支持安全 SMB `ontap-nas`仅适用于托管卷导入，不适用于非托管卷导入。

步骤

1. 在 TridentBackendConfig 中指定 adAdminUser，如下例所示：

```
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. 在存储类中添加注解。

   添加 `trident.netapp.io/smbShareAdUser`对存储类进行注解，以启用安全可靠的 SMB 连接。用户为注释指定的值 `trident.netapp.io/smbShareAdUser`应该与指定的用户名相同 `smbcreds`秘密。您可以从以下选项中选择一项 `smbShareAdUserPermission: full_control , change , 或者 read。默认权限是 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

学习如何在Trident系统中创建和使用ONTAP NAS 驱动程序。本节提供后端配置示例以及将后端映射到 StorageClasses 的详细信息。

后端配置选项

请参阅下表了解后端配置选项：

参数	描述	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`ontap-nas`， `ontap-nas-economy` ， 或者 `ontap-nas-flexgroup`
`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域名 (FQDN)。如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如：`[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]` 。为了实现MetroCluster 的无缝切换，请参阅MetroCluster示例。	"10.0.0.1", "[2001:1234:abcd::fefe]"

参数	描述	默认
dataLIF	LIF协议的IP地址。 NetApp建议指定 dataLIF。如果未提供， Trident将从 SVM 获取 dataLIF。您可以指定一个完全限定域名 (FQDN) 用于 NFS 挂载操作，从而创建轮询 DNS 以在多个 dataLIF 之间进行负载均衡。初始设置后可以更改。参考。如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如： [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555] 。 *Metrocluster 除外。*查看MetroCluster示例。	指定地址或从 SVM 派生，如果未指定（不推荐）
svm	要使用的存储虚拟机 *Metrocluster 除外。*查看MetroCluster示例。	如果是 SVM 则推导而来 `managementLIF`已指定
autoExportPolicy	启用自动导出策略创建和更新 [布尔值]。使用 `autoExportPolicy`和 `autoExportCIDRs`可选功能包括： Trident可以自动管理出口策略。	false
autoExportCIDRs	用于过滤 Kubernetes 节点 IP 的 CIDR 列表 `autoExportPolicy`已启用。使用 `autoExportPolicy`和 `autoExportCIDRs`可选功能包括： Trident可以自动管理出口策略。	["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	用于连接到集群/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 的身份"。	
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 的身份"。	
storagePrefix	在 SVM 中配置新卷时使用的前缀。设置后无法更新 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，尽管它会包含在卷名称中。	“三叉戟”

参数	描述	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。 当 SVM 中的聚合数据更新时， Trident会自动轮询 SVM 进行更新，而无需重启Trident控制器。当您在Trident中配置特定聚合以配置卷时，如果该聚合被重命名或移出 SVM，则在轮询 SVM 聚合时， Trident中的后端将变为失败状态。您必须将聚合更改为 SVM 上存在的聚合，或者将其完全删除，才能使后端恢复联机。	""
`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**。	（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。 当 SVM 中的聚合列表更新时， Trident会通过轮询 SVM 自动更新该列表，而无需重新启动Trident控制器。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， Trident中的后端将进入失败状态。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，才能使后端恢复联机。	""
`limitVolumeSize`	如果请求的卷大小大于此值，则配置失败。此外，它还限制了 qtree 管理的卷的最大大小，以及 `qtreesPerFlexvol`此选项允许自定义每个FlexVol volume的最大 qtree 数量。	（默认情况下不强制执行）
`debugTraceFlags`	故障排除时要使用的调试标志。例如，{"api":false, "method":true} 请勿使用 `debugTraceFlags`除非您正在进行故障排除并且需要详细的日志转储。	无效的
`nasType`	配置 NFS 或 SMB 卷的创建。选项有 `nfs`， `smb`或空值。设置为 null 则默认使用 NFS 卷。	`nfs`
`nfsMountOptions`	以逗号分隔的 NFS 挂载选项列表。 Kubernetes 持久卷的挂载选项通常在存储类中指定，但如果存储类中没有指定挂载选项， Trident将回退到使用存储后端配置文件中指定的挂载选项。如果在存储类或配置文件中未指定任何挂载选项， Trident将不会在关联的持久卷上设置任何挂载选项。	""

参数	描述	默认
qtreesPerFlexvol	每个FlexVol 的最大 Qtree 数量必须在 [50, 300] 范围内	“200”
smbShare	您可以指定以下一项：使用 Microsoft 管理控制台或ONTAP CLI 创建的 SMB 共享的名称；允许Trident 创建 SMB 共享的名称；或者您可以将参数留空以防止对卷的公共共享访问。对于本地部署的ONTAP，此参数是可选的。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`Trident使用ONTAP REST API 与后端通信；当设置为 `false`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登录角色必须具有访问权限。 `ontapi`应用。预定义项满足了这一点。 `vsadmin`和 `cluster-admin`角色。从Trident 24.06 版本和ONTAP 9.15.1 或更高版本开始， `useREST`设置为 `true`默认值；更改 `useREST`到 `false`使用 ONTAPI (ZAPI) 调用。	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 时可请求的最大FlexVol大小。	（默认情况下不强制执行）
denyNewVolumePools	限制 `ontap-nas-economy`后端创建新的FlexVol卷来包含它们的 Qtree。只有预先存在的 Flexvol 才能用于配置新的 PV。	
adAdminUser	具有对 SMB 共享完全访问权限的 Active Directory 管理员用户或用户组。使用此参数可为 SMB 共享提供管理员权限，并赋予其完全控制权限。	

卷配置的后端配置选项

您可以使用以下选项控制默认配置。 `defaults`配置部分。例如，请参见下面的配置示例。

参数	描述	默认
spaceAllocation	Q树的空间分配	“真的”
spaceReserve	空间预留模式；“无”（细）或“大量”（粗）	“没有任何”
snapshotPolicy	要使用的快照策略	“没有任何”
qosPolicy	要为创建的卷分配的 QoS 策略组。每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 之一	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 之一。ontap-nas-economy 不支持此功能。	""
snapshotReserve	快照预留的卷百分比	如果为“0”， `snapshotPolicy`为“无”，否则为“

参数	描述	默认
splitOnClone	创建时将克隆体从其母体中分离出来	"false"
encryption	在新卷上启用NetApp卷加密 (NVE)；默认设置为 false。要使用此选项，必须在集群上获得 NVE 许可并启用 NVE。如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。更多信息，请参阅："Trident如何与 NVE 和 NAE 协同工作" 。	"false"
tieringPolicy	分层策略使用“无”	
unixPermissions	新卷模式	NFS 卷的端口号为“777”；SMB 卷的端口号为空（不适用）。
snapshotDir	控制对以下方面的访问 `.snapshot`目录	NFSv4 为“true”，NFSv3 为“false”。
exportPolicy	出口政策的使用	“默认”
securityStyle	新卷的安全样式。 NFS 支持 `mixed`和 `unix`安全措施。中小企业支持 `mixed`和 `ntfs`安全措施。	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。	""

将 QoS 策略组与Trident结合使用需要ONTAP 9.8 或更高版本。您应该使用非共享的 QoS 策略组，并确保该策略组单独应用于每个成员。共享的 QoS 策略组强制规定所有工作负载的总吞吐量上限。

卷配置示例

以下是一个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"
```

为了 ontap-nas`和 `ontap-nas-flexgroups`Trident现在使用新的计算方法，以确保FlexVol 的尺寸与 snapshotReserve 百分比和 PVC 正确匹配。当用户请求 PVC 时， Trident会使用新的计算方法创建具有更多空间的原始FlexVol 。此计算方法可确保用户在 PVC 中获得其请求的可写空间，而不是少于其请求的空间。在 v21.07 之前，当用户请求 PVC（例如 5 GiB）时，如果快照预留百分比为 50%，则只能获得 2.5 GiB 的可写空间。这是因为用户请求的是整个卷。 `snapshotReserve`是其中的百分比。在Trident 21.07 中，用户请求的是可写空间，而Trident定义了该空间。 `snapshotReserve`将数值表示为占总体积的百分比。这不适用于 `ontap-nas-economy。请参阅以下示例了解其工作原理：

计算方法如下：

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

对于快照预留 = 50% 且 PVC 请求 = 5 GiB 的情况，总卷大小为 5/.5 = 10 GiB，可用大小为 5 GiB，这正是用户在 PVC 请求中请求的大小这 `volume show`该命令应显示与此示例类似的结果：

升级Trident时，先前安装的现有后端将按上述方式配置卷。对于升级前创建的卷，您应该调整其大小才能观察到更改。例如，一个 2 GiB 的 PVC `snapshotReserve=50`之前的结果是一个提供 1 GiB 可写空间的卷。例如，将卷大小调整为 3 GiB，将在 6 GiB 的卷上为应用程序提供 3 GiB 的可写空间。

最小配置示例

以下示例展示了基本配置，其中大多数参数都保留默认值。这是定义后端最简单的方法。

> 如果您在NetApp ONTAP上使用Amazon FSx和Trident，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，以避免在切换和切换回后端后手动更新后端定义。"SVM复制和恢复" 。

为了实现无缝切换和切换回，请指定 SVM `managementLIF`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`（如果使用受信任的 CA，则为可选） `backend.json`分别取客户端证书、私钥和受信任 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_
```

自动出口政策示例

本示例向您展示如何指示Trident使用动态导出策略来自动创建和管理导出策略。这对于以下情况也适用：`ontap-nas-economy`和 `ontap-nas-flexgroup`司机。

```
---
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

这个例子表明 `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

这 `smbShare`使用 SMB 卷的 FSx for ONTAP需要此参数。

```
---
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_
```

使用 **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

在下方所示的示例后端定义文件中，所有存储池都设置了特定的默认值，例如： `spaceReserve`没有，`spaceAllocation`为假，并且 `encryption`错误。虚拟池在存储部分中定义。

Trident在“备注”字段中设置配置标签。 FlexVol上已设置评论 `ontap-nas`或`FlexGroup `ontap-nas-flexgroup`。 Trident在配置时会将虚拟池上的所有标签复制到存储卷。为了方便起见，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，一些存储池会设置自己的参数。 `spaceReserve`， `spaceAllocation`， 和 `encryption`有些值会覆盖默认值，有些池会覆盖默认值。

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"
```

将后端映射到存储类

以下 StorageClass 定义指的是[具有虚拟池的后端示例]。使用 `parameters.selector`字段中，每个 StorageClass 都会指定哪些虚拟池可用于托管卷。卷将具有所选虚拟池中定义的各个方面。

- 这 `protection-gold`StorageClass 将映射到第一个和第二个虚拟池。 `ontap-nas-flexgroup`后端。只有这些泳池提供黄金级保护。

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

- 这 `protection-not-gold`StorageClass 将映射到第三个和第四个虚拟池。 `ontap-nas-flexgroup`后端。除了黄金之外，只有这些金池提供其他级别的保护。

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

- 这 `app-mysqldb`StorageClass 将映射到第四个虚拟池。 `ontap-nas`后端。这是唯一一个为mysqldb类型应用程序提供存储池配置的存储池。

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

- 该 `protection-silver-creditpoints-20k`StorageClass 将映射到第三个虚拟池。 `ontap-nas-flexgroup`后端。这是唯一提供银级保护和 20000 积分的彩池。

```
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"
```

- 这 `creditpoints-5k`StorageClass 将映射到第三个虚拟池。 `ontap-nas`后端和第二个虚拟池 `ontap-nas-economy`后端。这是唯一提供 5000 积分的彩池产品。

```
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，并确保满足存储需求。

更新 `dataLIF`初始配置后

初始配置完成后，您可以通过运行以下命令来更改 dataLIF，从而为新的后端 JSON 文件提供更新后的 dataLIF。

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

如果 PVC 连接到一个或多个舱体，则必须将所有相应的舱体放下，然后再将它们重新升起，以便新的 dataLIF 生效。

安全的中小企业示例

使用 **ontap-nas** 驱动程序进行后端配置

```
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** 驱动程序进行后端配置

```
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
```

后端配置及存储池

```
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
```

使用 **ontap-nas** 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: 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
```

请确保添加 `annotations`实现安全的SMB。如果没有注释，无论后端或 PVC 中设置了什么配置，安全 SMB 都无法工作。

使用 **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
```

包含单个 **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
```

包含多个 **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

"Amazon FSx for NetApp ONTAP"是一项完全托管的 AWS 服务，使客户能够启动和运行由NetApp ONTAP存储操作系统支持的文件系统。 FSx for ONTAP使您能够利用您熟悉的NetApp功能、性能和管理能力，同时享受在 AWS 上存储数据的简单性、敏捷性、安全性和可扩展性。 FSx for ONTAP支持ONTAP文件系统功能和管理 API。

您可以将Amazon FSx for NetApp ONTAP文件系统与Trident集成，以确保在 Amazon Elastic Kubernetes Service (EKS) 中运行的 Kubernetes 集群可以配置由ONTAP支持的块和文件持久卷。

文件系统是Amazon FSx中的主要资源，类似于本地的ONTAP集群。在每个 SVM 中，您可以创建一个或多个卷，这些卷是用于存储文件系统中的文件和文件夹的数据容器。 Amazon FSx for NetApp ONTAP将作为云端托管文件系统提供。新的文件系统类型称为 * NetApp ONTAP*。

通过将Trident与Amazon FSx for NetApp ONTAP结合使用，您可以确保在 Amazon Elastic Kubernetes Service (EKS) 中运行的 Kubernetes 集群可以配置由ONTAP支持的块和文件持久卷。

要求

此外"Trident的要求"要将 FSx for ONTAP与Trident集成，您需要：

- 现有的 Amazon EKS 集群或自管理 Kubernetes 集群 `kubectl`已安装。
- 集群工作节点可访问的现有Amazon FSx for NetApp ONTAP文件系统和存储虚拟机 (SVM)。
- 已准备好的工作节点"NFS 或 iSCSI"。

> 请务必按照 Amazon Linux 和 Ubuntu 所需的节点准备步骤进行操作。 "亚马逊机器图像"（AMI）取决于您的 EKS AMI 类型。

注意事项

- SMB 卷：
  - 使用以下方式支持 SMB 卷 `ontap-nas`仅限司机。
  - Trident EKS 插件不支持 SMB 卷。
  - Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。参考 "准备配置SMB卷" 了解详情。
- 在Trident 24.02 之前，在Amazon FSx文件系统上创建的、启用了自动备份的卷无法被Trident删除。为防止在Trident 24.02 或更高版本中出现此问题，请指定 `fsxFilesystemID`，AWS `apiRegion` ，AWS `apikey`以及 AWS `secretKey`在 AWS FSx for ONTAP的后端配置文件中。

> 如果您要为Trident指定 IAM 角色，则可以省略指定以下内容： `apiRegion` ， `apiKey` ， 和 `secretKey`明确地将字段传递给Trident 。更多信息，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。这个例子展示了 `multipath.conf`包含所需Trident设置但排除 EBS 磁盘进行多路径配置的文件：

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

身份验证

Trident提供两种身份验证方式。

- 基于凭证（推荐）：将凭证安全地存储在 AWS Secrets Manager 中。您可以使用 `fsxadmin`文件系统的用户或 `vsadmin`用户已配置到您的 SVM。

> Trident预计将以……的形式运营 `vsadmin`SVM 用户，或者使用具有相同角色但名称不同的用户。 Amazon FSx for NetApp ONTAP具有 `fsxadmin`该用户是ONTAP的有限替代品 `admin`集群用户。我们强烈建议使用 `vsadmin`用Trident。

- 基于证书： Trident将使用安装在 SVM 上的证书与 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说明：

- "ONTAP NAS 认证"
- "ONTAP SAN 身份验证"

已测试的亚马逊机器映像 **(AMI)**

EKS 集群支持各种操作系统，但 AWS 针对容器和 EKS 优化了某些 Amazon Machine Image (AMI)。以下 AMI 已使用NetApp Trident 25.02 进行测试。

急性心肌梗死	NAS	NAS经济	iSCSI	iSCSI经济型
AL2023_x86_64_STANDARD	是	是	是	是
AL2_x86_64	是	是	是的*	是的*
BOTTLEROCKET_x86_64	是的**	是	不适用	不适用
AL2023_ARM_64_STANDARD	是	是	是	是
AL2_ARM_64	是	是	是的*	是的*
BOTTLEROCKET_ARM_64	是的**	是	不适用	不适用

- * 如果不重启节点，则无法删除 PV
- ** 不适用于Trident版本 25.02 的 NFSv3。

> 如果您所需的 AMI 未在此处列出，并不意味着它不受支持；这仅仅意味着它尚未经过测试。此列表可作为 AMI 已知可用系统的指南。

使用以下工具进行测试：

- EKS版本：1.32
- 安装方法：Helm 25.06 和 AWS 附加组件 25.06
- 对于 NAS，NFSv3 和 NFSv4.1 都进行了测试。
- SAN 仅测试了 iSCSI，未测试 NVMe-oF。

已执行测试：

- 创建：存储类、PVC、Pod
- 删除：pod、pvc（常规、qtree/lun – 经济型、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

您可以配置 Kubernetes pod 以通过 AWS IAM 角色进行身份验证来访问 AWS 资源，而不是提供显式的 AWS 凭证。

要使用 AWS IAM 角色进行身份验证，您必须拥有一个使用 EKS 部署的 Kubernetes 集群。

创建 AWS Secrets Manager 密钥

由于Trident将向 FSx 虚拟服务器发出 API 来为您管理存储，因此它需要凭据才能执行此操作。传递这些凭证的安全方法是通过 AWS Secrets Manager 密钥。因此，如果您还没有 AWS Secrets Manager 密钥，则需要创建一个包含 vsadmin 帐户凭证的密钥。

此示例创建一个 AWS Secrets Manager 密钥来存储Trident CSI 凭证：

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

创建 IAM 策略

Trident也需要 AWS 权限才能正常运行。因此，您需要创建一个策略，赋予Trident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"
}
```

创建 **Pod** 身份或 **IAM** 角色以关联服务帐户 **(IRSA)**

您可以配置 Kubernetes 服务账户，使其承担 AWS Identity and Access Management (IAM) 角色（使用 EKS Pod Identity 或 IAM 角色进行服务账户关联）(IRSA)。任何配置为使用该服务账户的 Pod 都可以访问该角色有权访问的任何 AWS 服务。

Pod 身份

Amazon EKS Pod 身份关联允许您管理应用程序的凭证，类似于 Amazon EC2 实例配置文件向 Amazon EC2 实例提供凭证的方式。

在 **EKS** 集群上安装 **Pod Identity**：

您可以通过 AWS 控制台创建 Pod 标识，也可以使用以下 AWS CLI 命令：

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

更多信息请参阅"设置 Amazon EKS Pod 身份代理"。

创建 **trust-relationship.json** 文件：

创建 trust-relationship.json 文件，使 EKS 服务主体能够承担 Pod 身份的此角色。然后使用以下信任策略创建角色：

```
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：

将上一步创建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** 身份关联：

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"
        }
      }
    }
  ]
}
```

更新以下值 `trust-relationship.json`文件：

- **<account_id>** - 您的 AWS 账户 ID
- **<oidc_provider>** - 您的 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** 策略关联：

角色创建完成后，使用以下命令将策略（在上一步中创建的策略）附加到该角色：

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

请核实 **OICD** 提供商是否已关联：

请确认您的 OIDC 提供程序已与您的集群关联。您可以使用以下命令进行验证：

```
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 关联到您的集群：

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

如果您使用的是 **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：

- 舵
- EKS 附加组件

如果要使用快照功能，请安装 CSI 快照控制器插件。请参阅"为CSI卷启用快照功能"了解更多信息。

通过 **Helm** 安装**Trident**。

Pod 身份

1. 添加Trident Helm 仓库：

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

2. 请按照以下示例安装Trident：

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

您可以使用 `helm list`查看安装详细信息的命令，例如名称、命名空间、图表、状态、应用程序版本和修订号。

```
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
```

服务账户协会（**IRSA**）

1. 添加Trident Helm 仓库：

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

2. 设置*云提供商*和*云身份*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`查看安装详细信息的命令，例如名称、命名空间、图表、状态、应用程序版本和修订号。

```
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-
100.2506.0    25.06.0
```

如果您计划使用 iSCSI，请确保您的客户端计算机上已启用 iSCSI。如果您使用的是 AL2023 工作节点操作系统，可以通过在 Helm 安装中添加节点准备参数来自动安装 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** 插件安装**Trident**

Trident EKS 插件包含最新的安全补丁和错误修复，并经过 AWS 验证，可与 Amazon EKS 配合使用。 EKS 插件使您能够持续确保您的 Amazon EKS 集群安全稳定，并减少安装、配置和更新插件所需的工作量。

前提条件

在为 AWS EKS 配置Trident插件之前，请确保您已具备以下条件：

- 带有附加订阅的 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
- 现有的Amazon FSx for NetApp ONTAP文件系统

启用适用于 **AWS** 的**Trident**插件

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“集群”。
3. 选择要为其配置NetApp Trident CSI 插件的集群名称。
4. 选择“附加组件”，然后选择“获取更多附加组件”。
5. 请按照以下步骤选择插件：
   a. 向下滚动到“AWS Marketplace 附加组件”部分，然后在搜索框中输入“Trident”。
   b. 选中“Trident by NetApp”框右上角的复选框。
   c. 选择“下一步”。
6. 在“配置所选插件”设置页面上，执行以下操作：

   如果您使用 **Pod Identity** 关联，请跳过这些步骤。

   a. 请选择您要使用的*版本*。
   b. 如果您使用IRSA身份验证，请确保设置可选配置设置中提供的配置值：
      - 请选择您要使用的*版本*。
      - 按照*附加组件配置方案*进行操作，并将*配置值*部分中的*configurationValues*参数设置为您在上一步创建的角色 ARN（值应采用以下格式）：

```
{

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

}
```

+
如果选择“覆盖”作为冲突解决方法，则现有插件的一个或多个设置可能会被 Amazon EKS 插件设置覆盖。如果您不启用此选项，并且与您现有的设置存在冲突，则操作将失败。您可以使用生成的错误信息来排查冲突问题。在选择此选项之前，请确保 Amazon EKS 插件不会管理您需要自行管理的设置。

7. 选择“下一步”。
8. 在“审核和添加”页面上，选择“创建”。

   插件安装完成后，您将看到已安装的插件。

AWS CLI

1.创建 **`add-on.json`**文件：

对于 **Pod** 标识，请使用以下格式：

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

对于**IRSA**认证，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"
  }
}
```

代替 `<role ARN>`使用上一步创建的角色的 ARN。

2.安装**Trident EKS** 插件。

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

eksctl

以下示例命令安装Trident EKS 插件：

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

更新**Trident EKS** 插件

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“集群”。
3. 选择要为其更新NetApp Trident CSI 插件的集群名称。
4. 选择“附加组件”选项卡。
5. 选择“ NetApp Trident ”，然后选择“编辑”。
6. 在“配置NetApp Trident ”页面上，执行以下操作：
   a. 请选择您要使用的*版本*。
   b. 展开“可选配置设置”，并根据需要进行修改。
   c. 选择“保存更改”。

AWS CLI

以下示例更新 EKS 插件：

```
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

- 请检查您的 FSxN Trident CSI 插件的当前版本。代替 `my-cluster`使用您的集群名称。

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

示例输出：

```
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'"}
```

- 将插件更新到上一步输出中“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
```

如果你移除 `--force`如果选项和任何 Amazon EKS 插件设置与您现有的设置冲突，则更新 Amazon EKS 插件将失败；您将收到一条错误消息，以帮助您解决冲突。在指定此选项之前，请确保 Amazon EKS 插件不会管理您需要管理的设置，因为此选项会覆盖这些设置。有关此设置的其他选项的更多信息，请参阅"插件"。有关 Amazon EKS Kubernetes 字段管理的更多信息，请参阅"Kubernetes 字段管理"。

卸载/移除**Trident EKS** 插件

您可以通过两种方式移除 Amazon EKS 插件：

- 保留集群上的附加软件 – 此选项移除 Amazon EKS 对所有设置的管理。它还取消了 Amazon EKS 通知您更新以及在您启动更新后自动更新 Amazon EKS 插件的功能。但是，它可以保留集群上的附加软件。此选项使插件成为自管理安装，而不是 Amazon EKS 插件。选择此选项，插件不会出现停机时间。保留 `--preserve`命令中的选项用于保留插件。
- 从集群中完全移除附加软件 – NetApp建议，仅当集群中没有任何资源依赖于 Amazon EKS 附加组件时，才从集群中移除该附加组件。移除 `--preserve`选项 `delete`移除插件的命令。

如果插件关联了 IAM 帐户，则不会删除该 IAM 帐户。

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“集群”。
3. 选择要从中移除NetApp Trident CSI 插件的集群名称。
4. 选择“附加组件”选项卡，然后选择“ NetApp Trident ”。
5. 选择*删除*。
6. 在“移除 netapp_trident-operator 确认”对话框中，执行以下操作：
   a. 如果您希望 Amazon EKS 停止管理插件的设置，请选择“在集群上保留”。如果您希望在集群上保留附加软件，以便您可以自行管理附加软件的所有设置，请执行此操作。
   b. 输入 **netapp_trident-operator**。
   c. 选择*删除*。

AWS CLI

代替 `my-cluster`输入集群名称，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 插件：

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

ONTAP SAN 和 NAS 驱动程序集成

要创建存储后端，您需要创建 JSON 或 YAML 格式的配置文件。该文件需要指定您想要的存储类型（NAS 或 SAN）、文件系统、要从中获取存储的 SVM 以及如何对其进行身份验证。以下示例展示了如何定义基于 NAS 的存储，以及如何使用 AWS Secret 来存储要使用的 SVM 的凭证：

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"
    }
  }
}
```

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 Trident 后端配置 (TBC) 是否已成功创建：

```
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
```

FSx 适用于ONTAP驱动程序的详细信息

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

- `ontap-san`每个已配置的 PV 都是其自身Amazon FSx for NetApp ONTAP卷中的一个 LUN。推荐用于块存储。
- `ontap-nas`每个已配置的 PV 都是一个完整的Amazon FSx for NetApp ONTAP卷。推荐用于NFS和SMB。
- `ontap-san-economy`每个已配置的 PV 都是一个 LUN，每个Amazon FSx for NetApp ONTAP卷可配置 LUN 的数量。
- `ontap-nas-economy`每个已配置的 PV 都是一个 qtree，每个Amazon FSx for NetApp ONTAP卷可以配置 qtree 的数量。
- `ontap-nas-flexgroup`每个已配置的 PV 都是一个完整的Amazon FSx for NetApp ONTAP FlexGroup卷。

有关司机详情，请参阅"NAS驱动程序"和"SAN驱动程序"。

配置文件创建完成后，运行以下命令将其创建在 EKS 中：

```
kubectl create -f configuration_file
```

要验证状态，请运行以下命令：

```
kubectl get tbc -n trident
```

```
NAME                      BACKEND NAME            BACKEND UUID
PHASE   STATUS
backend-fsx-ontap-nas   backend-fsx-ontap-nas   7a551921-997c-4c37-a1d1-
f2f4c87fa629   Bound   Success
```

后端高级配置和示例

请参阅下表了解后端配置选项：

参数	描述	示例
`version`		始终为 1
`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 地址可以指定完全限定域名 (FQDN)。如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。如果您提供 `fsxFilesystemID`在 `aws`字段中，您无需提供 `managementLIF`因为Trident可以检索 SVM `managementLIF`来自 AWS 的信息。因此，您必须提供 SVM 下用户的凭据（例如：vsadmin），并且该用户必须拥有以下权限： `vsadmin`角色。	"10.0.0.1", "[2001:1234:abcd::fefe]"

参数	描述	示例
dataLIF	LIF协议的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。您可以指定一个完全限定域名 (FQDN) 用于 NFS 挂载操作，从而创建轮询 DNS 以在多个 dataLIF 之间进行负载均衡。初始设置后可以更改。参考。 * ONTAP SAN 驱动程序*：不要为 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 iSCI LIF。如果显式定义了dataLIF，则会生成警告。如果Trident安装时使用了 IPv6 标志，则可以设置为使用 IPv6 地址。 IPv6 地址必须用方括号定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。	
autoExportPolicy	启用自动导出策略创建和更新 [布尔值]。使用 `autoExportPolicy`和 `autoExportCIDRs`可选功能包括：Trident可以自动管理出口策略。	false
autoExportCIDRs	用于过滤 Kubernetes 节点 IP 的 CIDR 列表 `autoExportPolicy`已启用。使用 `autoExportPolicy`和 `autoExportCIDRs`可选功能包括：Trident可以自动管理出口策略。	"["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	连接到集群或 SVM 的用户名。用于基于凭证的身份验证。例如，vsadmin。	
password	连接集群或SVM的密码。用于基于凭证的身份验证。	
svm	使用的存储虚拟机	如果指定了 SVM 管理 LIF，则派生出该 LIF。
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

参数	描述	示例
`limitAggregateUsage`	*请勿指定用于Amazon FSx for NetApp ONTAP。*提供的 `fsxadmin`和 `vsadmin`不包含检索汇总使用情况和使用Trident限制它所需的权限。	请勿使用。
`limitVolumeSize`	如果请求的卷大小大于此值，则配置失败。此外，它还限制了其管理的 qtree 和 LUN 卷的最大大小，以及 `qtreesPerFlexvol`此选项允许自定义每个FlexVol volume的最大 qtree 数量。	（默认情况下不强制执行）
`lunsPerFlexvol`	每个 Flexvol 卷的最大 LUN 数必须在 [50, 200] 范围内。仅限SAN。	“100”
`debugTraceFlags`	故障排除时要使用的调试标志。例如，{"api":false, "method":true} 请勿使用 `debugTraceFlags`除非您正在进行故障排除并且需要详细的日志转储。	无效的
`nfsMountOptions`	以逗号分隔的 NFS 挂载选项列表。Kubernetes 持久卷的挂载选项通常在存储类中指定，但如果存储类中没有指定挂载选项， Trident将回退到使用存储后端配置文件中指定的挂载选项。如果在存储类或配置文件中未指定任何挂载选项， Trident将不会在关联的持久卷上设置任何挂载选项。	""
`nasType`	配置 NFS 或 SMB 卷的创建。选项有 `nfs`， `smb`或空值。 *必须设置为 `smb`适用于 SMB 卷。*设置为 null 则默认使用 NFS 卷。	`nfs`
`qtreesPerFlexvol`	每个FlexVol volume的最大 Qtree 数量必须在 [50, 300] 范围内	`"200"`
`smbShare`	您可以指定以下名称之一：使用 Microsoft 管理控制台或ONTAP CLI 创建的 SMB 共享的名称，或者允许Trident创建 SMB 共享的名称。此参数是Amazon FSx for ONTAP后端所必需的。	`smb-share`
`useREST`	使用ONTAP REST API 的布尔参数。设置为 `true`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登录角色必须具有访问权限。 `ontap`应用。预定义项满足了这一点。 `vsadmin`和 `cluster-admin`角色。	`false`

参数	描述	示例
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 凭证。 - name：包含 SVM 凭证的密钥的 Amazon 资源名称 (ARN)。 - type`设置为 `awsarn。请参阅"创建 AWS Secrets Manager 密钥"了解更多信息。	

卷配置的后端配置选项

您可以使用以下选项控制默认配置。 `defaults`配置部分。例如，请参见下面的配置示例。

参数	描述	默认
spaceAllocation	LUN 的空间分配	true
spaceReserve	空间预留模式；“无”（细）或“大量”（粗）	none
snapshotPolicy	要使用的快照策略	none
qosPolicy	要为创建的卷分配的 QoS 策略组。每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 之一。将 QoS 策略组与Trident结合使用需要ONTAP 9.8 或更高版本。您应该使用非共享的 QoS 策略组，并确保该策略组单独应用于每个成员。共享的 QoS 策略组强制规定所有工作负载的总吞吐量上限。	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 之一。 ontap-nas-economy 不支持此功能。	""
snapshotReserve	为快照“0”预留的卷百分比	如果 snapshotPolicy`是 `none，else “”
splitOnClone	创建时将克隆体从其母体中分离出来	false

参数	描述	默认
`encryption`	在新卷上启用NetApp卷加密 (NVE)；默认设置为 `false`。要使用此选项，必须在集群上获得 NVE 许可并启用 NVE。如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。更多信息，请参阅："Trident如何与 NVE 和 NAE 协同工作"。	`false`
`luksEncryption`	启用LUKS加密。参考"使用 Linux 统一密钥设置 (LUKS)"。仅限 SAN。	""
`tieringPolicy`	分层策略的使用 `none`	
`unixPermissions`	新卷模式。 **SMB**卷请留空。	""
`securityStyle`	新卷的安全样式。 NFS 支持 `mixed`和 `unix`安全措施。中小企业支持 `mixed`和 `ntfs`安全措施。	NFS 默认值为 `unix`。 SMB 默认值为 `ntfs`。

准备配置**SMB**卷

您可以使用以下方式配置 SMB 卷： `ontap-nas`司机。在你完成之前ONTAP SAN 和 NAS 驱动程序集成请完成以下步骤。

开始之前

在使用以下方式配置 SMB 卷之前： `ontap-nas`驾驶员，您必须具备以下条件。

- 一个 Kubernetes 集群，包含一个 Linux 控制器节点和至少一个运行 Windows Server 2019 的 Windows 工作节点。 Trident仅支持挂载到运行在 Windows 节点上的 pod 的 SMB 卷。
- 至少有一个包含您的 Active Directory 凭据的Trident密钥。生成秘密 `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代理"或者"GitHub：适用于 Windows 的 CSI 代理"适用于在 Windows 上运行的 Kubernetes 节点。

步骤

1. 创建SMB共享。您可以通过以下两种方式之一创建 SMB 管理共享："Microsoft 管理控制台"共享文件夹管理单元或使用ONTAP CLI。使用ONTAP CLI 创建 SMB 共享：
   a. 如有必要，请创建共享的目录路径结构。

      这 `vserver cifs share create`该命令检查在创建共享时 -path 选项中指定的路径。如果指定的路径不存在，则命令执行失败。

   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. 确认共享已创建：

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

请参阅"创建 SMB 共享"了解详细信息。

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

参数	描述	示例
`smbShare`	您可以指定以下名称之一：使用 Microsoft 管理控制台或ONTAP CLI 创建的 SMB 共享的名称，或者允许Trident创建 SMB 共享的名称。此参数是Amazon FSx for ONTAP后端所必需的。	`smb-share`
`nasType`	*必须设置为 `smb`.*如果为空，则默认为空。 `nfs` 。	`smb`
`securityStyle`	新卷的安全样式。 必须设置为 **`ntfs`**或者 **`mixed`**适用于 **SMB** 卷。	`ntfs`或者 `mixed`适用于 SMB 卷
`unixPermissions`	新卷模式。 对于 **SMB** 卷，此项必须留空。	""

配置存储等级和**PVC**

配置 Kubernetes StorageClass 对象并创建存储类，以指示Trident如何配置卷。创建一个使用已配置的 Kubernetes StorageClass 的 PersistentVolumeClaim (PVC) 来请求访问 PV。然后您可以将光伏组件安装到支架上。

创建存储类

配置 **Kubernetes StorageClass** 对象

这 "Kubernetes StorageClass 对象"该对象将Trident标识为该类使用的配置器，并指示Trident如何配置卷。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"
```

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"
```

要在 AWS Bottlerocket 上配置 NFSv3 卷，请添加所需的 `mountOptions`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
```

请参阅"Kubernetes 和Trident对象"有关存储类如何与……交互的详细信息 `PersistentVolumeClaim`以及控制Trident如何分配容量的参数。

创建存储类

步骤

1. 这是一个 Kubernetes 对象，所以请使用 `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上的存储池。

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

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

制作PVC管

一个 "*PersistentVolumeClaim*"（PVC）是对集群上持久卷的访问请求。

PVC 可以配置为请求存储特定尺寸或访问模式。使用关联的 StorageClass，集群管理员可以控制的不仅仅是 PersistentVolume 的大小和访问模式，还可以控制性能或服务级别。

制作好 PVC 后，就可以将容积安装到舱体中。

样品清单

PersistentVolumeClaim 样本清单

这些示例展示了PVC的基本配置选项。

带RWX接口的PVC

此示例展示了一个具有 RWX 访问权限的基本 PVC，它与一个名为 StorageClass 的 StorageClass 相关联。 `basic-csi` 。

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

使用 iSCSI 示例的 PVC

此示例展示了一个与名为 StorageClass 的存储类关联的、具有 RWO 访问权限的 iSCSI 基本 PVC。 `protection-gold` 。

```
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. 核实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"有关存储类如何与……交互的详细信息 `PersistentVolumeClaim`以及控制Trident如何分配容量的参数。

Trident属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	类型	价值观	提供	要求	由……支持
媒体[1]	string	机械硬盘、混合硬盘、固态硬盘	Pool 包含此类媒体；混合型媒体是指两者兼具。	指定的媒体类型	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san
供应类型	string	薄的，厚的	池支持这种配置方法	指定的配置方法	厚：全部 ontap ；薄：全部 ontap 和 solidfire-san
后端类型	string	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、gcp-cvs 、azure-netapp-files、ontap-san-economy	池属于这种类型的后端	指定的后端	所有司机
snapshots	布尔值	真，假	存储池支持带快照的卷	已启用快照的卷	ontap-nas 、ontap-san 、solidfire-san 、gcp-cvs
个克隆	布尔值	真，假	存储池支持卷克隆	已启用克隆的卷	ontap-nas 、ontap-san 、solidfire-san 、gcp-cvs

属性	类型	价值观	提供	要求	由.....支持
加密	布尔值	真，假	存储池支持加密卷	已启用加密的卷	ontap-nas、ontap-nas-economy、ontap-nas-flexgroups、ontap-san
IOPS	整数	正整数	Pool 能够保证在此范围内的 IOPS	容量保证了这些IOPS	solidfire-san

[1]： ONTAP Select系统不支持此系统

部署示例应用程序

创建存储等级和 PVC 后，即可将光伏组件安装到舱体上。本节列出了将 PV 连接到 pod 的示例命令和配置。

步骤

1. 将音量旋钮安装在一个音控器中。

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

以下示例展示了将PVC连接到舱体的基本配置：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
```

您可以使用以下方式监控进度 `kubectl get pod --watch`。

2. 确认卷已挂载到 `/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
```

现在您可以删除该 Pod 了。 Pod 应用将不复存在，但卷将保留。

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

在 **EKS** 集群上配置**Trident EKS** 插件

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 插件包含最新的安全补丁和错误修复，并经过 AWS 验证，可与 Amazon EKS 配合使用。 EKS 插件使您能够持续确保您的 Amazon EKS 集群安全稳定，并减少安装、配置和更新插件所需的工作量。

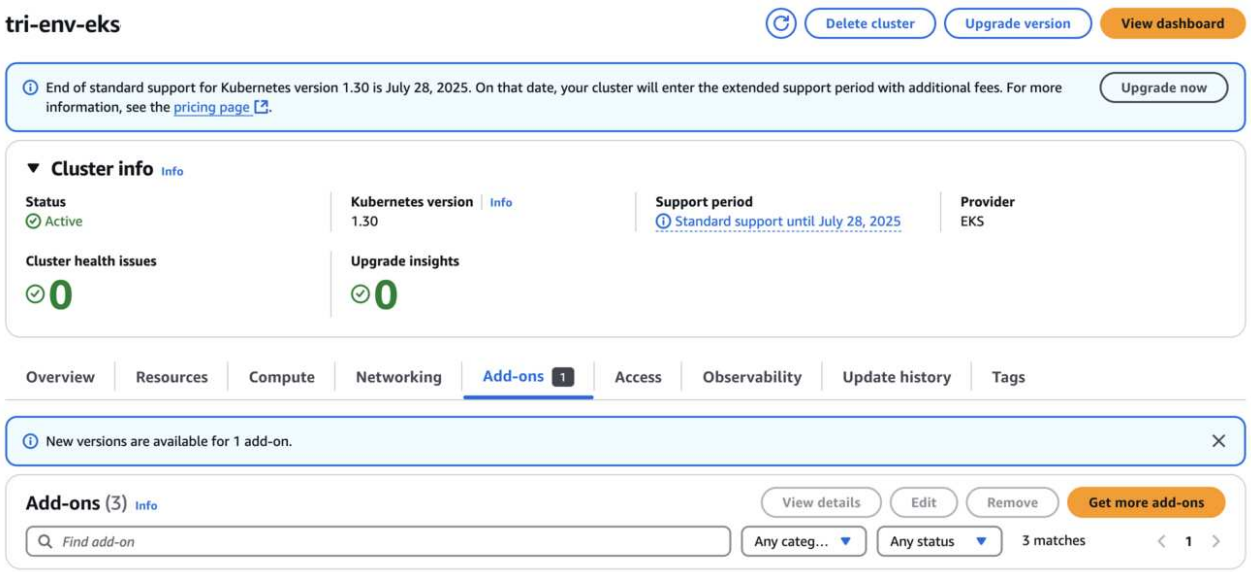
前提条件

在为 AWS EKS 配置Trident插件之前，请确保您已具备以下条件：

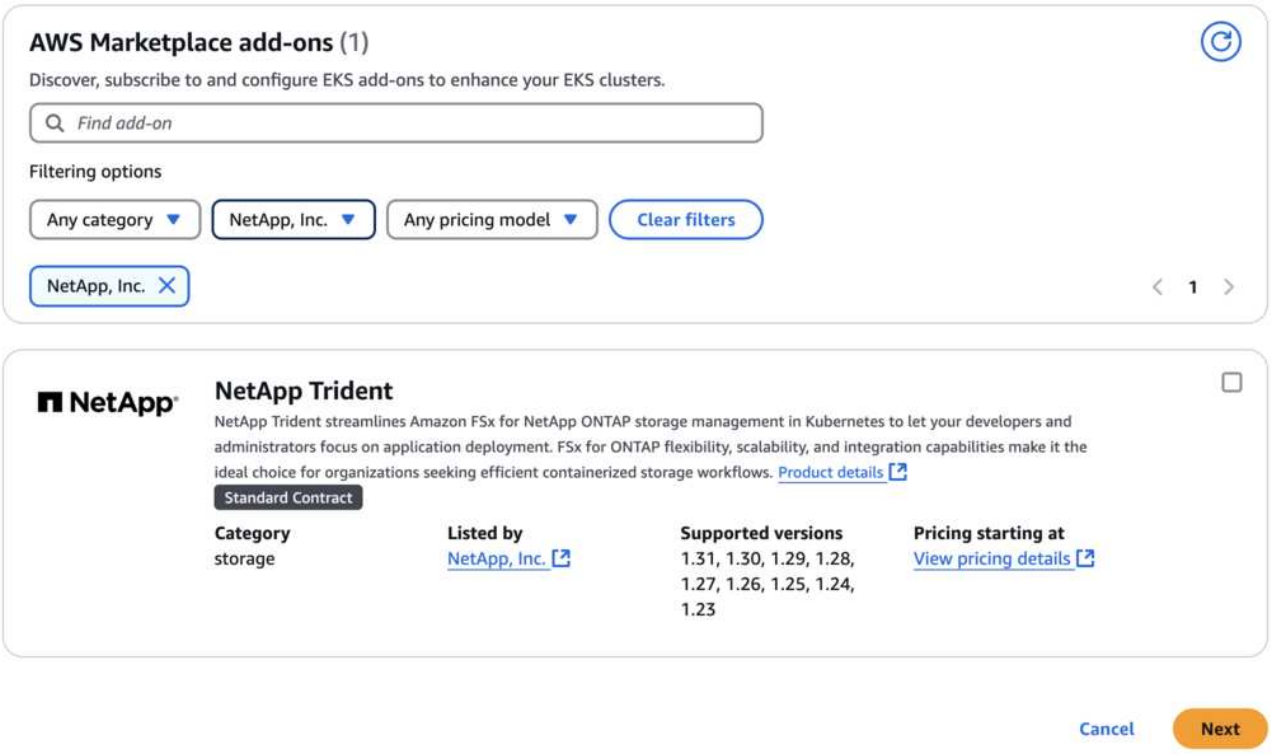
- 具有使用插件权限的 Amazon EKS 集群账户。参考"Amazon EKS 附加组件"。
- AWS 对 AWS Marketplace 的权限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI 类型：Amazon Linux 2 (AL2_x86_64) 或 Amazon Linux 2 Arm (AL2_ARM_64)
- 节点类型：AMD 或ARM
- 现有的Amazon FSx for NetApp ONTAP文件系统

步骤

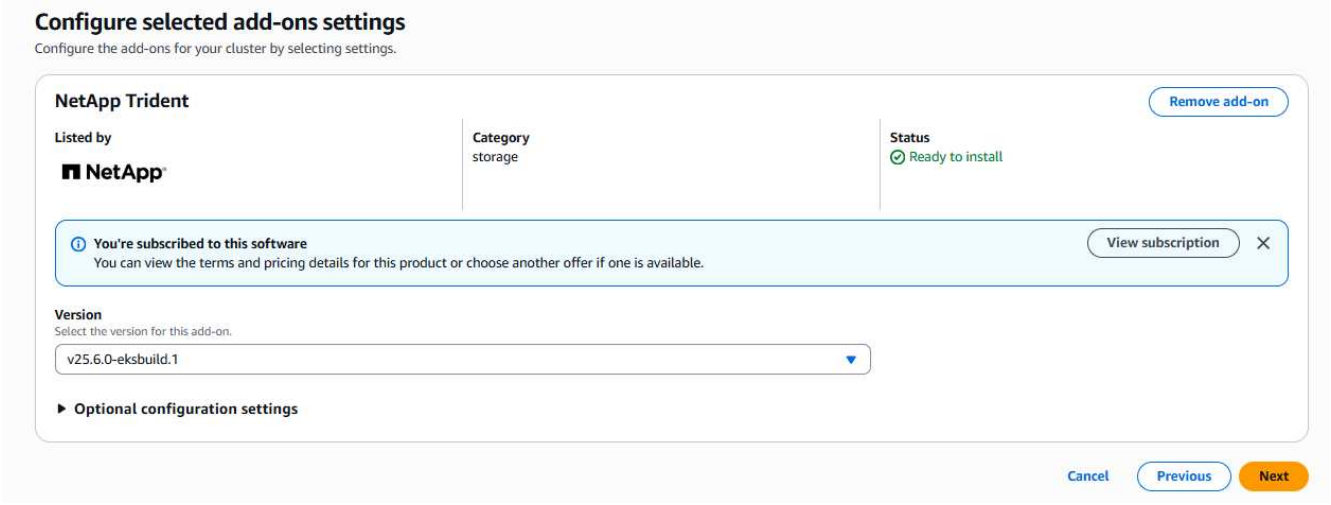
1. 请务必创建 IAM 角色和 AWS 密钥，以使 EKS pod 能够访问 AWS 资源。有关说明，请参阅"创建 IAM 角色和 AWS Secret"。
2. 在 EKS Kubernetes 集群上，导航到“附加组件”选项卡。



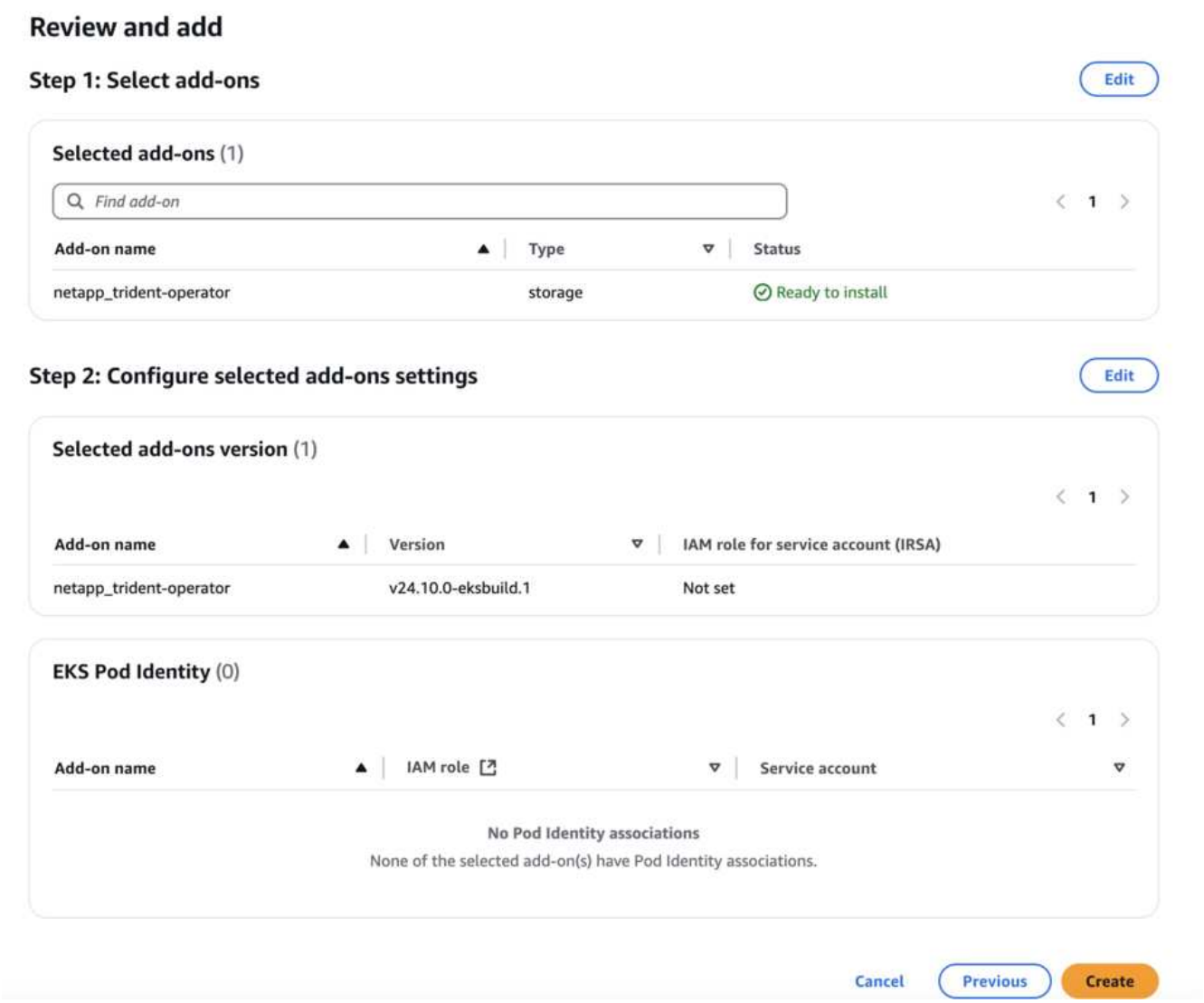
3. 前往 **AWS Marketplace** 插件，然后选择 *storage* 类别。



4. 找到 * NetApp Trident*，选中Trident插件的复选框，然后单击 下一步。
5. 选择所需的插件版本。

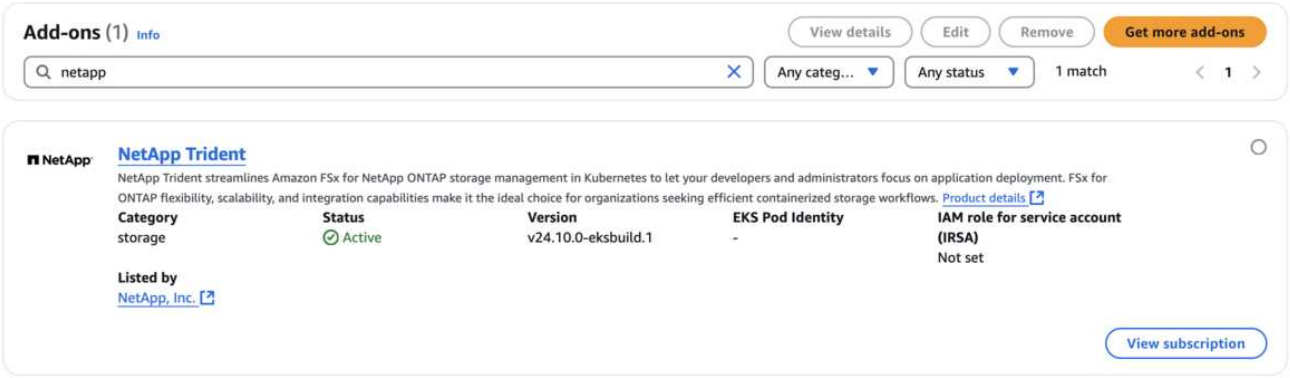


6. 配置所需的附加组件设置。



7. 如果您使用的是 IRSA（服务帐户的 IAM 角色），请参阅其他配置步骤。"此处" 。
8. 选择“创建”。

9. 确认插件状态为_Active_。



10. 运行以下命令以验证Trident是否已正确安装在集群上：

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

11. 继续进行设置并配置存储后端。有关信息，请参阅"配置存储后端"。

使用 **CLI** 安装/卸载**Trident EKS** 插件

使用 **CLI** 安装**NetApp Trident EKS** 插件：

以下示例命令安装Trident EKS 插件：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.0-eksbuild.1 （另有专用版本）
```

使用 **CLI** 卸载**NetApp Trident EKS** 插件：

以下命令卸载Trident EKS 插件：

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

使用 **kubectl** 创建后端

后端定义了Trident与存储系统之间的关系。它告诉Trident如何与该存储系统通信，以及Trident应该如何从中配置卷。 Trident安装完成后，下一步是创建后端。这 `TridentBackendConfig`自定义资源定义 (CRD) 使您能够通过 Kubernetes 界面直接创建和管理Trident后端。您可以使用 `kubectl`或者适用于您的 Kubernetes 发行版的等效 CLI 工具。

`TridentBackendConfig`

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

在创建之后 `TridentBackendConfig`对于该对象，会发生以下情况：

- Trident会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。
- 这 `TridentBackendConfig`与……有着独特的联系 `TridentBackend`这是由Trident生产的。

每个 `TridentBackendConfig`与……保持一对一映射关系 `TridentBackend`前者是提供给用户设计和配置后端的界面；后者是Trident表示实际后端对象的方式。

`TridentBackend`CR 由Trident自动创建。你*不应该*修改它们。如果您想对后端进行更新，请通过修改以下文件来实现： `TridentBackendConfig`目的。

以下示例展示了格式： `TridentBackendConfig` CR：

```
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
```

您还可以查看以下示例： "trident-installer"包含所需存储平台/服务的示例配置的目录。

这 `spec`接受后端特定的配置参数。在这个例子中，后端使用了 `ontap-san`存储驱动程序，并使用此处表格中列出的配置参数。有关所需存储驱动程序的配置选项列表，请参阅"存储驱动程序的后端配置信息"。

这 `spec`本节还包括 `credentials`和 `deletionPolicy`这些字段是新引入的 `TridentBackendConfig`CR：

- `credentials`此参数为必填字段，包含用于向存储系统/服务进行身份验证的凭据。这设置为用户创建的 Kubernetes Secret。凭据不能以明文形式传递，否则将导致错误。
- `deletionPolicy`此字段定义了当……时应该发生的情况 `TridentBackendConfig`被删除。它可以取以下两个值之一：
  - `delete`这会导致两者被删除。 `TridentBackendConfig` CR及其相关后端。这是默认值。
  - `retain`当 `TridentBackendConfig`CR 被删除后，后端定义仍然存在，并且可以通过以下方式进行管理： `tridentctl` 。将删除策略设置为 `retain`允许用户降级到早期版本（21.04 之前），并保留已创建的后端。该字段的值可以在之后更新 `TridentBackendConfig`已创建。

后端名称是通过以下方式设置的： `spec.backendName` 。如果未指定，则后端名称设置为……的名称 `TridentBackendConfig`对象（`metadata.name`）。建议使用显式方式设置后端名称。 `spec.backendName` 。

用以下方式创建的后端 `tridentctl`没有关联的 `TridentBackendConfig`目的。您可以选择使用以下方式管理此类后端 `kubectl`通过创建一个 `TridentBackendConfig`CR。必须注意指定完全相同的配置参数（例如： `spec.backendName`， `spec.storagePrefix`， `spec.storageDriverName`， 等等）。 Trident将自动绑定新创建的`TridentBackendConfig`利用现有的后端。

步骤概述

使用以下方式创建新的后端 `kubectl`你应该这样做：

1. 创建一个 "Kubernetes Secret"该密钥包含Trident与存储集群/服务通信所需的凭据。
2. 创建一个 `TridentBackendConfig`目的。这包含有关存储集群/服务的具体信息，并引用上一步中创建的密钥。

创建后端后，您可以使用以下方式观察其状态 `kubectl get tbc <tbc-name> -n <trident-namespace>`并收集更多细节信息。

步骤 **1**：创建 **Kubernetes Secret**

创建一个包含后端访问凭据的密钥。这是每个存储服务/平台独有的。以下是一个例子：

```
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密钥中必须包含的字段：

存储平台密钥字段描述	秘密	字段描述
Azure NetApp Files	客户端ID	应用注册中的客户端 ID
Cloud Volumes Service for GCP	私钥 ID	私钥的ID。具有 CVS 管理员角色的 GCP 服务帐户的部分 API 密钥
Cloud Volumes Service for GCP	私钥	私钥。具有 CVS 管理员角色的 GCP 服务帐户的部分 API 密钥

存储平台密钥字段描述	秘密	字段描述
元素（NetApp HCI/ SolidFire）	端点	针对SolidFire集群的 MVIP，包含租户凭证
ONTAP	用户名	用于连接到集群/SVM 的用户名。用于基于凭证的身份验证
ONTAP	password	连接到集群/SVM 的密码。用于基于凭证的身份验证
ONTAP	客户端私钥	客户端私钥的 Base64 编码值。用于基于证书的身份验证
ONTAP	chap用户名	入站用户名。如果 useCHAP=true，则为必填项。为了 `ontap-san`和 `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP 发起者密钥。如果 useCHAP=true，则此项为必填项。为了 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true，则此项为必填项。为了 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP 目标发起者密钥。如果 useCHAP=true，则此项为必填项。为了 `ontap-san`和 `ontap-san-economy`

此步骤中创建的秘密将在以下位置引用： `spec.credentials`领域的 `TridentBackendConfig`在下一步创建的对象。

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

您现在可以开始创建您的 `TridentBackendConfig`CR。在这个例子中，后端使用了 `ontap-san`驱动程序是通过使用以下方式创建的： `TridentBackendConfig`下图所示物体：

```
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
```

步骤 3：验证状态 `TridentBackendConfig`CR

现在你已经创建了 `TridentBackendConfig`CR，您可以核实状态。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
```

后端已成功创建并绑定到 `TridentBackendConfig`CR。

相位可以取以下值之一：

- `Bound`： 这 `TridentBackendConfig`CR 与后端关联，该后端包含 `configRef`设置为 `TridentBackendConfig`CR 的 uid。
- `Unbound`：用以下方式表示 ""。这 `TridentBackendConfig`该对象未绑定到后端。所有新创建 `TridentBackendConfig`CR 默认处于此阶段。阶段变化后，它无法再恢复到未绑定状态。
- `Deleting`： 这 `TridentBackendConfig`CR的 `deletionPolicy`已设置为删除。当 `TridentBackendConfig`CR 被删除后，状态变为正在删除。
  - 如果后端不存在持久卷声明 (PVC)，则删除 `TridentBackendConfig`这将导致Trident删除后端以及 `TridentBackendConfig`CR。
  - 如果后端存在一个或多个 PVC，则进入删除状态。这 `TridentBackendConfig`CR随后也进入删除阶段。后端和 `TridentBackendConfig`只有在所有 PVC 都被删除后才会删除。
- `Lost`与后端相关的 `TridentBackendConfig`CR 被意外或故意删除，并且 `TridentBackendConfig`CR 仍然保留着对已删除后端的引用。这 `TridentBackendConfig`无论如何，CR 仍然可以被删除。 `deletionPolicy`价值。
- `Unknown`Trident无法确定与以下系统关联的后端的状态或是否存在： `TridentBackendConfig` CR。例如，如果 API 服务器没有响应，或者如果 `tridentbackends.trident.netapp.io`CRD 缺失。这可能需要干预。

至此，后端已成功创建！此外，还可以处理以下几种操作："后端更新和后端删除"。

（可选）步骤 **4**：了解更多详情

您可以运行以下命令来获取有关后端的更多信息：

```
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 YAML/JSON 转储文件。`TridentBackendConfig`。

```
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 `backendName`以及 `backendUUID`后端是根据以下情况创建的：`TridentBackendConfig CR。这 lastOperationStatus`该字段表示上次操作的状态`TridentBackendConfig`CR（变更请求）可以由用户触发（例如，用户更改了某些内容）。 `spec）或由Trident触发（例如，在Trident重启期间）。结果要么是成功，要么是失败。 phase`代表了以下关系的状态：`TridentBackendConfig CR 和后端。在上面的例子中， `phase`具有 Bound 值，这意味着`TridentBackendConfig`CR与后端相关。

您可以运行 `kubectl -n trident describe tbc <tbc-cr-name>`获取事件日志详细信息的命令。

> 您无法更新或删除包含关联后端的后端。 TridentBackendConfig`使用对象`tridentctl。要了解在以下两者之间切换所涉及的步骤： tridentctl`和`TridentBackendConfig，"参见此处"。

管理后端

使用 **kubectl** 执行后端管理

了解如何使用以下方式执行后端管理操作 kubectl。

删除后端

通过删除一个 TridentBackendConfig`您指示Trident删除/保留后端（基于 `deletionPolicy）。要删除后端，请确保 deletionPolicy`已设置为删除。仅删除 `TridentBackendConfig`确保 `deletionPolicy`设置为保留。这样可以确保后端仍然存在，并且可以通过以下方式进行管理：`tridentctl 。

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 用户负责清理密钥。删除机密信息时务必谨慎。只有当后端不再使用密钥时，才应该删除密钥。

查看现有后端

运行以下命令：

```
kubectl get tbc -n trident
```

你也可以运行 tridentctl get backend -n trident`或者 `tridentctl get backend -o yaml -n trident`获取所有现有后端的列表。此列表还将包括使用以下方式创建的后端： `tridentctl 。

更新后端

更新后端的原因可能有很多：

- 存储系统的凭证已更改。要更新凭据，需要更新 Kubernetes Secret，该 Secret 用于：`TridentBackendConfig`对象必须更新。 Trident会自动使用提供的最新凭据更新后端。运行以下命令更新 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）。
  - 您可以更新 `TridentBackendConfig`使用以下命令直接通过 Kubernetes 访问对象：

    ```
    kubectl apply -f <updated-backend-file.yaml>
    ```

  - 或者，您可以对现有内容进行更改。 `TridentBackendConfig`使用以下命令进行回车：

```
kubectl edit tbc <tbc-name> -n trident
```

- 如果后端更新失败，后端将继续保持其最后一次已知的配置。您可以通过运行以下命令查看日志以确定原因。 `kubectl get tbc <tbc-name> -o yaml -n trident`或者 `kubectl describe tbc <tbc-name> -n trident`。
- 在您发现并纠正配置文件中的问题后，您可以重新运行更新命令。

使用 **tridentctl** 执行后端管理

了解如何使用以下方式执行后端管理操作 `tridentctl`。

创建后端

创建之后"后端配置文件"运行以下命令：

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

如果后端创建失败，则说明后端配置存在问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在您发现并纠正配置文件中的问题后，您只需运行以下命令即可。 `create`再次发出命令。

删除后端

要从Trident中删除后端，请执行以下操作：

1. 获取后端名称：

```
tridentctl get backend -n trident
```

2. 删除后端：

```
tridentctl delete backend <backend-name> -n trident
```

如果Trident已从此后端配置了仍然存在的卷和快照，则删除后端将阻止从该后端配置新卷。后端将继续处于“删除”状态。

查看现有后端

要查看Trident已知的后端，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，或者您尝试了无效的更新。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在您发现并纠正配置文件中的问题后，您只需运行以下命令即可。 `update`再次发出命令。

确定使用后端存储的存储类

这是一个您可以使用 JSON 回答的问题示例： `tridentctl`后端对象的输出。这使用 `jq`您需要安装该实用程序。

```
tridentctl get backend -o json | jq '[.items[] | {backend: .name,
storageClasses: [.storage[].storageClasses]|unique}]'
```

这也适用于使用以下方式创建的后端： `TridentBackendConfig` 。

在后端管理选项之间切换

了解Trident中管理后端的不同方法。

后端管理选项

随着 `TridentBackendConfig`现在，管理员有两种独特的后端管理方式。这就引出了以下问题：

- 可以使用以下方式创建后端 `tridentctl`**以……进行管理** `TridentBackendConfig`?
- 可以使用以下方式创建后端 `TridentBackendConfig`**可通过以下方式进行管理** `tridentctl`?

管理 tridentctl`使用后端 `TridentBackendConfig

本节介绍管理使用以下方式创建的后端所需的步骤： `tridentctl`直接通过 Kubernetes 接口创建 `TridentBackendConfig`物体。

这适用于以下情况：

- 预先存在的后端，没有 `TridentBackendConfig`因为它们是用……创造的 `tridentctl。
- 使用以下方式创建的新后端 `tridentctl`而其他 `TridentBackendConfig`物体是存在的。

在这两种情况下，后端都将继续存在， Trident将调度卷并对其进行操作。管理员此时有两种选择：

- 继续使用 `tridentctl`管理使用它创建的后端。
- 使用以下方式创建的绑定后端 `tridentctl`到一个新的 `TridentBackendConfig`目的。这样做意味着后端将使用以下方式进行管理： `kubectl`而不是 `tridentctl。

使用以下方式管理预先存在的后端 `kubectl`您需要创建一个 `TridentBackendConfig`它与现有后端绑定。以下是其工作原理概述：

1. 创建 Kubernetes Secret。该密钥包含Trident与存储集群/服务通信所需的凭据。
2. 创建一个 `TridentBackendConfig`目的。这包含有关存储集群/服务的具体信息，并引用上一步中创建的密钥。必须注意指定完全相同的配置参数（例如： `spec.backendName`， `spec.storagePrefix`， `spec.storageDriverName`， 等等）。 `spec.backendName`必须设置为现有后端的名称。

步骤 0：确定后端

创建一个 `TridentBackendConfig`如果要绑定到现有后端，则需要获取后端配置。在这个例子中，我们假设使用以下 JSON 定义创建了一个后端：

```
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，如下例所示：

```
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` 将自动绑定到预先存在的 `ontap-nas-backend（如本例所示）。请确保满足以下要求：

- 后端名称在以下位置定义： `spec.backendName` 。
- 配置参数与原后端相同。
- 虚拟池（如果存在）必须保持与原始后端相同的顺序。
- 凭证通过 Kubernetes Secret 提供，而不是以明文形式提供。

在这种情况下， `TridentBackendConfig`将会像这样：

```
cat backend-tbc-ontap-nas.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
```

步骤 3：验证状态 `TridentBackendConfig`CR

之后 `TridentBackendConfig`已经创建，它的阶段必须是 `Bound`。它还应该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 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

后端现在将完全使用以下方式进行管理： `tbc-ontap-nas-backend` `TridentBackendConfig`目的。

管理 `TridentBackendConfig`使用后端 `tridentctl`

`tridentctl`可用于列出使用以下方式创建的后端： `TridentBackendConfig` 。此外，管理员还可以选择通过以下方式完全管理此类后端： `tridentctl`通过删除 `TridentBackendConfig`并确保 `spec.deletionPolicy`设置为 `retain`。

步骤 0：确定后端

例如，假设我们使用以下方式创建了以下后端 `TridentBackendConfig`：

```
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 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

从输出结果可以看出： `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`

让我们来看看它的价值 `deletionPolicy`。需要将其设置为 `retain`。这确保了当 `TridentBackendConfig`CR 被删除后，后端定义仍然存在，并且可以通过以下方式进行管理： `tridentctl`。

```
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
```

除非另有说明，否则请勿进行下一步。 `deletionPolicy`设置为 `retain`。

步骤二：删除 `TridentBackendConfig`CR

最后一步是删除 `TridentBackendConfig`CR。确认后 `deletionPolicy`设置为 `retain`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 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

删除后 `TridentBackendConfig`Trident只是移除该对象，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，以指示Trident如何配置卷。

配置 **Kubernetes StorageClass** 对象

这 "Kubernetes StorageClass 对象"确定Trident是该类使用的配置器，并指示Trident如何配置卷。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"有关存储类如何与……交互的详细信息 `PersistentVolumeClaim`以及控制Trident如何分配容量的参数。

创建存储类

创建 StorageClass 对象后，即可创建存储类。[存储类示例]提供一些您可以使用或修改的基本示例。

步骤

1. 这是一个 Kubernetes 对象，所以请使用 `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上的存储池。

```
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提供 "针对特定后端的简单存储类定义"。

或者，您可以编辑 `sample-input/storage-class-csi.yaml.templ`安装程序附带的文件并替换 `*BACKEND_TYPE*` 使用存储驱动程序名称。

```
./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、识别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。如果用户未在持久卷声明 (PVC) 中指定持久卷，则将使用此存储类来配置持久卷。

- 通过设置注解来定义默认存储类 `storageclass.kubernetes.io/is-default-class`在存储类定义中设置为 true。根据规范，任何其他值或缺少注释均被解释为错误。
- 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"}}}'
```

- 同样，您可以使用以下命令移除默认的存储类注解：

```
kubectl patch storageclass <storage-class-name> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

Trident安装程序包中也有一些包含此注释的示例。

集群中一次只能存在一个默认存储类。从技术上讲，Kubernetes 并不阻止你拥有多个默认存储类，但它的行为就好像根本没有默认存储类一样。

确定存储类的后端

这是一个您可以使用 JSON 回答的问题示例： `tridentctl` Trident后端对象的输出。这使用 `jq`您可能需要先安装该实用程序。

```
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>`应该替换成你的存储类。

通过此存储类别创建的任何持久卷都将保持不变， Trident将继续管理它们。

Trident强制执行空白 `fsType`因为它创造了大量的销量。对于 iSCSI 后端，建议强制执行 `parameters.fsType`在存储类中。您应该删除现有的 StorageClasses 并重新创建它们。 `parameters.fsType`指定的。

配置和管理卷

提供一定量

创建一个使用已配置的 Kubernetes StorageClass 的 PersistentVolumeClaim (PVC) 来请求访问 PV。然后您可以将光伏组件安装到支架上。

概述

一个 "*PersistentVolumeClaim*"（PVC）是对集群上持久卷的访问请求。

PVC 可以配置为请求存储特定尺寸或访问模式。使用关联的 StorageClass，集群管理员可以控制的不仅仅是 PersistentVolume 的大小和访问模式，还可以控制性能或服务级别。

制作好PVC管后，就可以将管体安装到管座中。

制作**PVC**管

步骤

1. 创建 PVC。

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

2. 核实PVC状态。

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS AGE
pvc-storage Bound    pv-name     1Gi        RWO                    5m
```

1. 将音量旋钮安装在一个音控器中。

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

> 您可以使用以下方式监控进度 `kubectl get pod --watch`。

2. 确认卷已挂载到 `/my/mount/path`。

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

3. 现在您可以删除该 Pod 了。 Pod 应用将不复存在，但卷将保留。

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

样品清单

PersistentVolumeClaim 样本清单

这些示例展示了PVC的基本配置选项。

PVC管材，带RWO通道

此示例展示了一个具有 RWO 访问权限的基本 PVC，它与一个名为 StorageClass 的 StorageClass 相关联。 `basic-csi` 。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

PVC 和 NVMe/TCP

此示例展示了一个与名为 StorageClass 的 StorageClass 关联的、具有 RWO 访问权限的 NVMe/TCP 基本 PVC。 `protection-gold` 。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

Pod清单示例

这些示例展示了将 PVC 连接到舱体的基本配置。

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"有关存储类如何与……交互的详细信息 `PersistentVolumeClaim`以及控制Trident如何分配容量的参数。

扩大规模

Trident为 Kubernetes 用户提供了在创建卷后扩展卷的能力。查找有关扩展 iSCSI、NFS、SMB、NVMe/TCP 和 FC 卷所需的配置的信息。

扩展 iSCSI 卷

您可以使用 CSI 配置程序扩展 iSCSI 持久卷 (PV)。

> iSCSI 卷扩展受以下方式支持： `ontap-san` ， `ontap-san-economy` ， `solidfire-san`驱动程序需要 Kubernetes 1.16 及更高版本。

步骤 1：配置 StorageClass 以支持卷扩展

编辑 StorageClass 定义以进行设置 `allowVolumeExpansion`田野 `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，对其进行编辑以包含以下内容： `allowVolumeExpansion`范围。

步骤 2：使用您创建的 StorageClass 创建 PVC。

编辑PVC定义并更新 `spec.resources.requests.storage`为了反映新的所需尺寸，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创建一个持久销量 (PV) 并将其与此持久销量声明 (PVC) 关联起来。

```
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** 的舱体

将光伏组件连接到舱体上，以便调整其尺寸。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。
- 尝试调整未连接的 PV 的大小时， Trident会在存储后端扩展卷。 PVC 与 pod 绑定后， Trident会重新扫描设备并调整文件系统的大小。 Kubernetes 会在扩展操作成功完成后更新 PVC 大小。

在这个例子中，创建了一个使用以下功能的 pod： `san-pvc` 。

```
 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**：展开 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并更新。 `spec.resources.requests.storage`至 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的体积来验证扩容是否正确：

```
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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

扩大 **FC** 体积

您可以使用 CSI 配置程序扩展 FC 持久卷 (PV)。

FC体积扩张得到了以下方面的支持： `ontap-san`驱动程序需要 Kubernetes 1.16 及更高版本。

步骤 **1**：配置 **StorageClass** 以支持卷扩展

编辑 StorageClass 定义以进行设置 `allowVolumeExpansion`田野 `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，对其进行编辑以包含以下内容： `allowVolumeExpansion`范围。

步骤 2：使用您创建的 **StorageClass** 创建 **PVC**。

编辑PVC定义并更新 `spec.resources.requests.storage`为了反映新的所需尺寸，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创建一个持久销量 (PV) 并将其与此持久销量声明 (PVC) 关联起来。

```
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** 的舱体

将光伏组件连接到舱体上，以便调整其尺寸。调整燃料电池光伏组件容量时有两种情况：

- 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。
- 尝试调整未连接的 PV 的大小时， Trident会在存储后端扩展卷。 PVC 与 pod 绑定后， Trident会重新扫描设备并调整文件系统的大小。 Kubernetes 会在扩展操作成功完成后更新 PVC 大小。

在这个例子中，创建了一个使用以下功能的 pod： `san-pvc` 。

```
 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**：展开 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并更新。 `spec.resources.requests.storage`至 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的体积来验证扩容是否正确：

```
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支持为已配置的 NFS PV 扩展容量。 `ontap-nas` ， `ontap-nas-economy` ， `ontap-nas-flexgroup` ， `gcp-cvs` ， 和 `azure-netapp-files`后端。

步骤 1：配置 StorageClass 以支持卷扩展

要调整 NFS PV 的大小，管理员首先需要配置存储类以允许卷扩展，方法是设置以下参数：`allowVolumeExpansion`田野 `true`：

```
cat storageclass-ontapnas.yaml
```

```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，则可以通过使用以下命令直接编辑现有存储类： `kubectl edit storageclass`允许体积膨胀。

步骤 2：使用您创建的 **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应该为该 PVC 创建一个 20 MiB 的 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：展开 **PV**

要将新建的 20 MiB PV 调整为 1 GiB，请编辑 PVC 并进行设置 `spec.resources.requests.storage`到 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体积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    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

进口量

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以执行以下操作：

- 将应用程序容器化并重用其现有数据集
- 为临时应用程序使用数据集的克隆版本
- 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

导入卷之前，请考虑以下事项。

- 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，应该先断开镜像关系。

- 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，然后执行导入操作。

这对于块卷来说尤其重要，因为 Kubernetes 不知道之前的连接，很容易将活动卷附加到 pod 上。这可能导致数据损坏。

- 尽管 `StorageClass`必须在 PVC 上指定， Trident在导入过程中不使用此参数。在创建卷时，会使用存储类根据存储特性从可用存储池中进行选择。由于卷已存在，因此导入时无需选择存储池。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。
- 现有容积尺寸在PVC中确定和设定。卷被存储驱动程序导入后，PV 会创建，并带有指向 PVC 的 ClaimRef。
  - 回收策略初始设置为 `retain`在PV中。 Kubernetes 成功绑定 PVC 和 PV 后，回收策略会更新为与存储类的回收策略相匹配。
  - 如果存储类的回收策略是 `delete`当 PV 被删除时，存储卷也会被删除。
- 默认情况下， Trident管理 PVC，并在后端重命名FlexVol volume和 LUN。你可以通过 `--no-manage`导入非托管卷的标志。如果你使用 `--no-manage`在对象的生命周期内， Trident不会对 PVC 或 PV 执行任何额外的操作。删除 PV 时，存储卷不会被删除，其他操作（如卷克隆和卷调整大小）也会被忽略。

如果您想使用 Kubernetes 来管理容器化工作负载，但又想在 Kubernetes 之外管理存储卷的生命周期，则此选项非常有用。

- 在 PVC 和 PV 中添加注释，其作用有两个：一是指示卷已导入，二是指示 PVC 和 PV 是否已管理。此注释不应修改或删除。

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。

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`。

   以下是一个最低规格示例：

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: my_claim
     namespace: my_namespace
   spec:
     accessModes:
       - ReadWriteOnce
     storageClassName: my_storage_class
   ```

   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`用于指定包含卷的Trident后端名称以及唯一标识存储上卷的名称的命令（例如：ONTAP FlexVol、Element Volume、 Cloud Volumes Service路径）。这 `-f`需要提供参数来指定PVC文件的路径。

   ```
   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

请查看以下卷导入示例，了解支持的驱动程序。

ONTAP NAS 和ONTAP NAS FlexGroup

Trident支持使用以下方式导入卷： `ontap-nas`和 `ontap-nas-flexgroup`司机。

- Trident不支持使用 `ontap-nas-economy` 司机。
- 这 `ontap-nas`和 `ontap-nas-flexgroup`驱动程序不允许重复的卷名称。

每卷都是用以下方式创建的 `ontap-nas`driver 是ONTAP集群上的FlexVol volume。使用以下方式导入FlexVol卷 `ontap-nas`驱动程序的工作原理相同。已存在于ONTAP集群上的FlexVol卷可以作为卷导入。 `ontap-nas` PVC。同样， FlexGroup卷也可以导入为 `ontap-nas-flexgroup`PVC。

使用 **tridentctl** 的 **ONTAP NAS** 示例

下面展示了托管卷和非托管卷导入的示例。

托管卷

以下示例导入一个名为“ managed_volume`在名为 `ontap_nas：

```
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    |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

未托管卷

使用时 `--no-manage`理由是， Trident不会重命名卷。

以下示例导入 `unmanaged_volume`在 `ontap_nas`后端：

```
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>
```

未托管卷

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支持使用卷导入 `ontap-san` （iSCSI、NVMe/TCP 和 FC）和 `ontap-san-economy` 司机。

Trident可以导入包含单个 LUN 的ONTAP SAN FlexVol卷。这与 `ontap-san` 驱动程序，它为每个 PVC 创建一个FlexVol volume，并在FlexVol volume内创建一个 LUN。Trident导入FlexVol volume并将其与 PVC 定义关联。Trident可以导入 `ontap-san-economy` 包含多个 LUN 的卷。

ONTAP SAN 示例

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 volume重命名为 `pvc-<uuid>`FlexVol volume中的 LUN 格式 `lun0。

以下示例导入了 `ontap-san-managed`FlexVol volume存在于 `ontap_san_default`后端：

```
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    |
+-----------------------------------------+--------+--------------
+----------+-----------------------------------------+--------+---------+
```

未托管卷

以下示例导入 `unmanaged_example_volume`在 `ontap_san`后端：

```
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   |
+-----------------------------------------+---------+--------------
+----------+-----------------------------------------+--------+---------+
```

如果将 LUN 映射到与 Kubernetes 节点 IQN 共享同一 IQN 的 igroup，如下例所示，则会收到以下错误： `LUN already mapped to initiator(s) in this group`。您需要移除启动器或取消映射 LUN 才能导入卷。

```
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
```

Element

Trident支持使用NetApp Element软件和NetApp HCI卷导入 `solidfire-san`司机。

> Element驱动程序支持重复的卷名称。但是，如果卷名称重复， Trident会返回错误。作为一种变通方法，克隆卷，提供一个唯一的卷名称，然后导入克隆的卷。

以下示例导入一个 `element-managed`后端容量 `element_default。

```
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    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud Platform

Trident支持使用以下方式导入卷： `gcp-cvs`司机。

> 要将由NetApp Cloud Volumes Service支持的卷导入 Google Cloud Platform，请通过卷路径识别该卷。卷路径是卷导出路径中位于以下位置之后的部分： `:/`。例如，如果导出路径是 `10.0.0.1:/adroit-jolly-swift`体积路径为 `adroit-jolly-swift。

Google Cloud Platform 示例

以下示例导入一个 `gcp-cvs`后端容量 `gcpcvs_YEppr`具有体积路径 `adroit-jolly-swift。

```
tridentctl import volume gcpcvs_YEppr adroit-jolly-swift -f <path-to-pvc-
file> -n trident

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | gcp-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident支持使用以下方式导入卷： `azure-netapp-files`司机。

> 要导入Azure NetApp Files卷，请通过卷路径识别该卷。卷路径是卷导出路径中位于以下位置之后的部分： `:/` 。例如，如果挂载路径是 `10.0.0.2:/importvol1`体积路径为 `importvol1`。

以下示例导入一个 `azure-netapp-files`后端容量 `azurenetappfiles_40517`体积路径 `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支持使用以下方式导入卷： `google-cloud-netapp-volumes`司机。

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
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

以下示例导入一个 `google-cloud-netapp-volumes`当两个体积存在于同一区域时，体积为：

```
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。这可以帮助您识别卷并将其轻松映射到各自的 Kubernetes 资源（PVC）。您还可以在后端级别定义模板，以创建自定义卷名称和自定义标签；您创建、导入或克隆的任何卷都将遵循这些模板。

开始之前

支持自定义卷名称和标签：

1. 卷创建、导入和克隆操作。
2. 对于 ontap-nas-economy 驱动程序，只有 Qtree 卷的名称符合名称模板。
3. 对于 ontap-san-economy 驱动程序，只有 LUN 名称符合名称模板。

限制

1. 可自定义卷名称仅与ONTAP本地驱动程序兼容。
2. 可自定义的卷名称不适用于现有卷。

可自定义卷名称的关键行为

1. 如果由于名称模板中的语法无效而导致失败，则后端创建将失败。但是，如果模板应用失败，则卷将按照现有的命名约定命名。
2. 当使用后端配置中的名称模板命名卷时，存储前缀不适用。可以直接在模板中添加任何所需的前缀值。

后端配置示例，包含名称模板和标签

可以在根级别和/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}}"
  }
}
```

池级示例

```
{
  "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 }}"
      }
    }
  ]
}
```

名称模板示例

示例 **1**：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

示例 **2**：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

需要考虑的几点

1. 对于卷导入，只有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，卷名称遵循后端定义根级别定义的名称模板。
3. Trident不支持将切片运算符与存储前缀一起使用。
4. 如果模板无法生成唯一的卷名称， Trident将添加一些随机字符来创建唯一的卷名称。
5. 如果 NAS 经济型卷的自定义名称长度超过 64 个字符， Trident将按照现有的命名约定命名这些卷。对于所有其他ONTAP驱动程序，如果卷名称超过名称限制，则卷创建过程将失败。

跨命名空间共享 **NFS** 卷

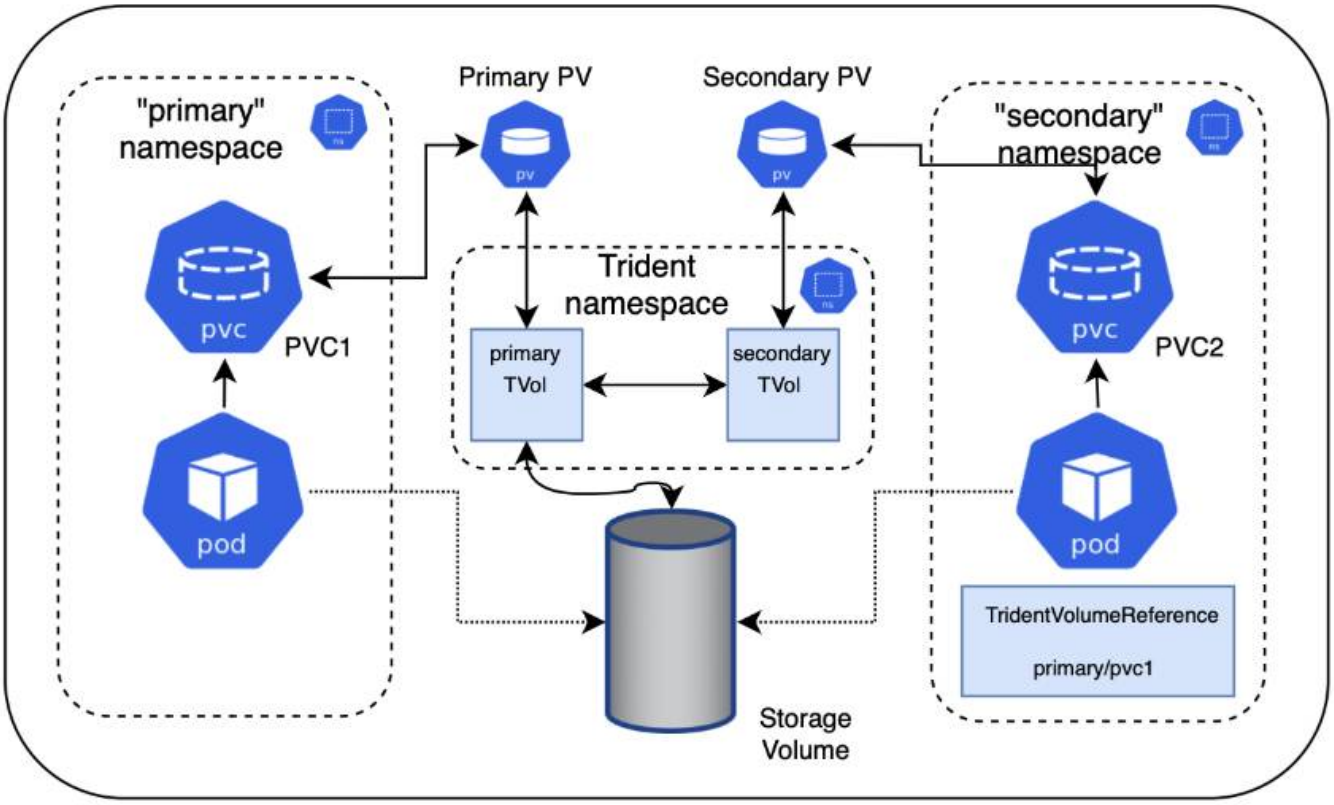
使用Trident，您可以在主命名空间中创建卷，并在一个或多个辅助命名空间中共享该卷。

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。这种 Kubernetes 原生解决方案具有以下优点：

- 多级访问控制以确保安全性
- 适用于所有Trident NFS 卷驱动程序
- 不依赖 tridentctl 或任何其他非原生 Kubernetes 功能

此图展示了跨两个 Kubernetes 命名空间的 NFS 卷共享。



快速启动

只需几个步骤即可设置NFS卷共享。

1 配置源**PVC**以共享体积

源命名空间所有者授予访问源 PVC 中数据的权限。

2 授予在目标命名空间中创建 **CR** 的权限

集群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，以使用源 PVC 中的数据源。

配置源命名空间和目标命名空间

为确保安全，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行动。用户角色在每个步骤中都会被指定。

步骤

1. 源命名空间所有者： 创建PVC(`pvc1` ）在源命名空间中，授予与目标命名空间共享的权限(`namespace2`) 使用 `shareToNamespace`注解。

```
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。

- 您可以使用逗号分隔的列表将 PVC 共享给多个命名空间。例如，`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4` 。
- 您可以使用以下方式共享到所有命名空间 `*`。例如，`trident.netapp.io/shareToNamespace: *`
- 您可以更新PVC以包含 `shareToNamespace`随时可以添加注释。

2. *集群管理员：*确保已建立适当的 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一个指向源命名空间的 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： 创建PVC(`pvc2` ）在目标命名空间(`namespace2`) 使用 `shareFromPVC`注释以指定来源 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
```

目标PVC管的尺寸必须小于或等于源PVC管的尺寸。

结果

Trident读取 `shareFromPVC`在目标 PVC 上添加注释，并将目标 PV 创建为没有自身存储资源的从属卷，该卷指向源 PV 并共享源 PV 存储资源。目的地 PVC 和 PV 看起来已正常绑定。

删除共享卷

您可以删除跨多个命名空间共享的卷。 Trident将移除对源命名空间中该卷的访问权限，并保留对共享该卷的其他命名空间的访问权限。当所有引用该卷的命名空间都被删除后， Trident会删除该卷。

使用 **`tridentctl get`**查询子卷

使用[`tridentctl`实用程序，您可以运行 `get`获取子卷的命令。更多信息，请参阅链接：../trident-reference/tridentctl.html[`tridentctl`命令和选项]。

```
Usage:
  tridentctl get [option]
```

标志：

- `-h, --help`：有关卷的帮助。
- `--parentOfSubordinate string`：将查询限制在子源卷上。
- `--subordinateOf string`：将查询限制在卷的下级。

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方法来防止覆盖共享卷数据。
- 您无法通过移除源PVC来撤销对源PVC的访问权限。 `shareToNamespace`或者 `shareFromNamespace`注

释或删除 `TridentVolumeReference`CR。要撤销访问权限，必须删除从属 PVC。

- 从属卷无法进行快照、克隆和镜像操作。

了解更多信息

要了解有关跨命名空间卷访问的更多信息：

- 访问"在命名空间之间共享卷：迎接跨命名空间卷访问"。
- 观看演示"NetAppTV"。

跨命名空间克隆卷

使用Trident，您可以利用同一 Kubernetes 集群中不同命名空间内的现有卷或卷快照创建新卷。

前提条件

克隆卷之前，请确保源后端和目标后端是同一类型，并且具有相同的存储类别。

跨命名空间克隆仅支持以下情况： `ontap-san`和 `ontap-nas`存储驱动程序。不支持只读克隆。

快速启动

只需几个步骤即可设置卷克隆。

1 配置源 **PVC** 以克隆卷

源命名空间所有者授予访问源 PVC 中数据的权限。

2 授予在目标命名空间中创建 **CR** 的权限

集群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 以克隆源命名空间中的 PVC。

配置源命名空间和目标命名空间

为确保安全，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。用户角色在每个步骤中都会被指定。

步骤

1. 源命名空间所有者： 创建PVC(`pvc1`）在源命名空间中(`namespace1`)授予与目标命名空间共享的权限(`namespace2`)使用 `cloneToNamespace`注解。

```
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。

- 您可以使用逗号分隔的列表将 PVC 共享给多个命名空间。例如，`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4` 。
- 您可以使用以下方式共享到所有命名空间 `*`。例如，`trident.netapp.io/cloneToNamespace: *`
- 您可以更新PVC以包含 `cloneToNamespace`随时可以添加注释。

2. 集群管理员： 确保已配置正确的基于角色的访问控制 (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一个指向源命名空间的 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： 创建PVC(`pvc2` ）在目标命名空间(`namespace2`) 使用 `cloneFromPVC`或者 `cloneFromSnapshot`， 和 `cloneFromNamespace`用于指定源PVC的注释。

```
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，用于复制数据以实现灾难恢复。 您可以使用名为Trident镜像关系 (TMR) 的命名空间自定义资源定义 (CRD) 来执行以下操作：

- 在体积（PVC）之间建立镜像关系
- 移除卷之间的镜像关系
- 打破镜像关系
- 在灾难情况下（故障转移）提升备用卷的可用性
- 在计划内故障转移或迁移期间，实现应用程序在集群间的无损迁移。

复制的前提条件

开始之前，请确保满足以下先决条件：

ONTAP集群

- * Trident *：使用ONTAP作为后端的源 Kubernetes 集群和目标 Kubernetes 集群上必须存在Trident版本 22.10 或更高版本。
- 许可证：使用数据保护包的ONTAP SnapMirror异步许可证必须在源 ONTAP 集群和目标ONTAP集群上启用。请参阅 "ONTAP中的SnapMirror许可概述"了解更多信息。

  从ONTAP 9.10.1 开始，所有许可证均以NetApp许可证文件 (NLF) 的形式交付，这是一个可以启用多种功能的单个文件。请参阅"ONTAP One 附带的许可证"了解更多信息。

仅支持SnapMirror异步保护。

对等

- 集群和 **SVM**： ONTAP存储后端必须相互连接。请参阅 "集群和SVM对等连接概述"了解更多信息。

  确保两个ONTAP集群之间复制关系中使用的 SVM 名称是唯一的。

- * Trident和 SVM*：对等远程 SVM 必须可供目标集群上的Trident使用。

支持的驱动程序

NetApp Trident支持使用NetApp SnapMirror技术进行卷复制，该技术使用以下驱动程序支持的存储类：**`ontap-nas`**：**NFS** `ontap-san`：iSCSI **`ontap-san`**：**FC** `ontap-san`：NVMe/TCP（最低要求ONTAP版本 9.15.1）

ASA r2 系统不支持使用SnapMirror进行卷复制。有关ASA r2 系统的信息，请参阅"了解ASA r2 存储系统"。

制作镜面**PVC**

按照这些步骤，并使用 CRD 示例，在主卷和辅助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. 使用以下方式创建一个 StorageClass 对象 `trident.netapp.io/replication: true`范围。

      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. 使用本地信息创建镜像关系变更请求。

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 的状态字段。

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. 在辅助 Kubernetes 集群上执行以下步骤：

   a. 创建一个 StorageClass，并设置 trident.netapp.io/replication: true 参数。

   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镜像关系 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将使用配置的关系策略名称（或ONTAP的默认值）创建SnapMirror关系并对其进行初始化。

c. 创建一个 PVC，使用先前创建的 StorageClass 作为辅助存储类（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，如果该关系不存在，则创建卷失败。如果存在此关系，Trident将确保将新的FlexVol volume放置在与 MirrorRelationship 中定义的远程 SVM 对等的 SVM 上。

卷复制状态

Trident镜像关系 (TMR) 是一种 CRD，它表示 PVC 之间复制关系的一端。目标 TMR 具有一个状态，该状态告诉Trident期望的状态是什么。目的地 TMR 具有以下状态：

- 已建立：本地 PVC 是镜像关系的目标量，这是一个新的关系。
- 已推广：本地 PVC 可读写且可安装，目前没有镜像关系。
- 重新建立：本地 PVC 是镜像关系的目标量，并且之前也处于该镜像关系中。
  - 如果目标卷曾经与源卷存在关联，则必须使用重新建立的状态，因为它会覆盖目标卷的内容。
  - 如果卷之前未与源建立关系，则重新建立状态将失败。

在计划外故障切换期间促进辅助**PVC**的运行

在辅助 Kubernetes 集群上执行以下步骤：

- 将 TridentMirrorRelationship 的 *spec.state* 字段更新为 `promoted`。

在计划故障切换期间推广备用**PVC**

在计划故障转移（迁移）期间，执行以下步骤以提升辅助 PVC：

步骤

1. 在主 Kubernetes 集群上，创建 PVC 的快照，并等待快照创建完成。

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*，并将 *spec.promotedSnapshotHandle* 更新为快照的内部名称。
4. 在辅助 Kubernetes 集群上，确认 TridentMirrorRelationship 的状态（status.state 字段）是否已提升。

故障转移后恢复镜像关系

在恢复镜像关系之前，选择你想作为新主要方的那一方。

步骤

1. 在辅助 Kubernetes 集群上，确保 TridentMirrorRelationship 上的 *spec.remoteVolumeHandle* 字段的值已更新。
2. 在辅助 Kubernetes 集群上，将 TridentMirrorRelationship 的 *spec.mirror* 字段更新为 `reestablished`。

附加手术

Trident支持对主卷和辅助卷执行以下操作：

将主**PVC**复制到新的辅助**PVC**

请确保您已备有主PVC管和备用PVC管。

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. 在主（源）集群上为要建立的新辅助（目标）PVC 创建一个新的 TridentMirrorRelationship CRD。

调整镜像、主或次级**PVC**的尺寸

PVC 可以像往常一样调整大小，如果数据量超过当前大小， ONTAP将自动扩展任何目标 flevxols。

从 **PVC** 中移除复制

要移除复制，请对当前辅助卷执行以下操作之一：

- 删除辅助 PVC 上的镜像关系。这会破坏复制关系。
- 或者，将 spec.state 字段更新为 *promoted*。

删除一个**PVC**（之前已镜像）

Trident会检查是否存在复制的 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完成删除之前转换为 *promoted* 状态。如果选择删除的 TMR 已处于 *promoted* 状态，则不存在镜像关系，TMR 将被删除， Trident会将本地 PVC 提升为 *ReadWrite*。此删除操作会释放ONTAP中本地卷的SnapMirror元数据。如果将来要将此卷用于镜像关系，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在线时，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`或者 `state: reestablished`用于更新关系的字段。将目标卷提升为常规读写卷时，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与ONTAP集群直接连接。请参考以下 TridentActionMirrorUpdate 的示例格式：

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 TridentActionMirrorUpdate CRD 的状态。它可以取值 成功、进行中_或_失败。

使用 **CSI** 拓扑

Trident可以利用以下方式选择性地创建卷并将其附加到 Kubernetes 集群中的节点： "CSI 拓扑功能" 。

概述

使用 CSI 拓扑功能，可以根据区域和可用区域将对卷的访问限制到节点子集。如今，云服务提供商允许 Kubernetes 管理员创建基于区域的节点。节点可以位于一个区域内的不同可用区，也可以跨多个区域。为了方便在多区域架构中为工作负载配置卷， Trident使用了 CSI 拓扑。

了解更多关于 CSI 拓扑功能的信息 "此处"。

Kubernetes 提供了两种独特的卷绑定模式：

- 和 `VolumeBindingMode`设置为 `Immediate`Trident创建卷时没有任何拓扑感知。卷绑定和动态配置在创建

PVC 时处理。这是默认设置。 `VolumeBindingMode`适用于不强制执行拓扑约束的集群。持久卷的创建不依赖于请求 pod 的调度要求。

- 和 `VolumeBindingMode`设置为 `WaitForFirstConsumer`PVC 的持久卷的创建和绑定将被延迟，直到使用该 PVC 的 pod 被调度和创建。这样，就可以创建卷来满足拓扑要求所强制执行的调度约束。

> 这 `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。

你需要什么

要使用 CSI 拓扑结构，您需要以下组件：

- 运行中的 Kubernetes 集群"支持的 Kubernetes 版本"

```
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"}
```

- 集群中的节点应带有标签，以体现拓扑感知能力。(`topology.kubernetes.io/region`和`topology.kubernetes.io/zone`）。在安装Trident之前，集群中的节点上*应该存在这些标签*，以便Trident能够感知拓扑结构。

```
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**：创建拓扑感知后端

Trident存储后端可以设计为根据可用区选择性地配置卷。每个后端都可以携带一个可选组件 `supportedTopologies`表示所支持的区域和地区列表的块。对于使用此类后端的 StorageClasses，只有在受支持的区域/区域中调度的应用程序请求时才会创建卷。

以下是一个后端定义示例：

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"
    }
  ]
}
```

`supportedTopologies`用于提供每个后端区域和分区的列表。这些区域和分区代表了StorageClass 中可以提供的允许值的列表。对于包含后端提供的区域和可用区子集的存储类，Trident会在后端创建一个卷。

你可以定义 `supportedTopologies`每个存储池也是如此。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
```

在这个例子中， `region`和 `zone`标签代表存储池的位置。 `topology.kubernetes.io/region`和 `topology.kubernetes.io/zone`决定存储池可以从何处使用。

步骤 **2**：定义拓扑感知的存储类。

根据提供给集群中节点的拓扑标签，可以定义 StorageClasses 来包含拓扑信息。这将决定哪些存储池可作为 PVC 请求的候选对象，以及哪些节点子集可以使用Trident提供的卷。

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
```

在上述 StorageClass 定义中， `volumeBindingMode`设置为 `WaitForFirstConsumer`。使用此 StorageClass 请求的 PVC 只有在 pod 中被引用后才会执行。和， `allowedTopologies`提供要使用的区域和范围。这 `netapp-san-us-east1`StorageClass 在存储上创建 PVC。 `san-backend-us-east1`后端定义如上所述。

步骤 3：制作并使用 PVC

创建 StorageClass 并将其映射到后端后，现在可以创建 PVC。

请参阅示例 `spec`以下：

```
---
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 将产生以下结果：

```
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
```

为了让Trident形成体积并将其与 PVC 结合，请使用 PVC 管。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
```

此 podSpec 指示 Kubernetes 将 pod 调度到存在于以下位置的节点上： `us-east1`在该区域中，选择任何存在的节点。 `us-east1-a`或者 `us-east1-b`区域。

请查看以下输出：

```
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
```

更新后端以包含 `supportedTopologies`

可以更新现有的后端，使其包含以下列表： `supportedTopologies`使用 `tridentctl backend update`。这不会影响已经分配的容量，只会用于后续的PVC。

查找更多信息

- "管理容器资源"
- "节点选择器"
- "亲和力和反亲和力"
- "污点和容忍度"

使用快照

Kubernetes 持久卷 (PV) 的卷快照可以创建卷的某个时间点副本。您可以创建使用Trident创建的卷的快照，导入在Trident之外创建的快照，从现有快照创建新卷，以及从快照恢复卷数据。

概述

卷快照受支持 `ontap-nas`， `ontap-nas-flexgroup` ， `ontap-san` ， `ontap-san-economy` ， `solidfire-san` ， `gcp-cvs` ， `azure-netapp-files` ， 和 `google-cloud-netapp-volumes`司机。

开始之前

要使用快照，您必须拥有外部快照控制器和自定义资源定义 (CRD)。这是 Kubernetes 编排器（例如：Kubeadm、GKE、OpenShift）的职责。

如果您的 Kubernetes 发行版不包含快照控制器和 CRD，请参阅[部署卷快照控制器]。

> 如果在 GKE 环境中创建按需卷快照，则不要创建快照控制器。 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`更多信息，请参阅……"卷快照类" 。
   - 这 `driver`指向Trident CSI 驱动程序。
   - `deletionPolicy`可以 `Delete`或者 `Retain`。设置为 `Retain`即使发生以下情况，存储集群上的底层物理快照仍会被保留： `VolumeSnapshot`对象已被删除。

   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`快照名称设置为 `pvc1-snap`。 VolumeSnapshot 类似于 PVC，并且与某个 PVC 相关联。 `VolumeSnapshotContent`代表实际快照的对象。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 你可以识别出 `VolumeSnapshotContent`对象为 `pvc1-snap`通过描述来获取卷快照。这 `Snapshot Content Name`标识提供此快照的 VolumeSnapshotContent 对象。这 `Ready To Use`该参数表明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`使用名为 VolumeSnapshot 的 VolumeSnapshot 创建 PVC `<pvc-name>`作为数据来源。 PVC管制作完成后，可以将其连接到舱体上，并像其他PVC管一样使用。

> PVC 将在与源卷相同的后端创建。参考"知识库：无法在备用后端从Trident PVC 快照创建 PVC。"。

以下示例使用以下方法创建 PVC： `pvc1-snap`作为数据源。

```
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 预配置快照流程"使集群管理员能够创建 `VolumeSnapshotContent`在Trident之外创建的对象和导入快照。

开始之前

Trident肯定创建或导入了快照的父卷。

步骤

1. 集群管理员： 创建一个 `VolumeSnapshotContent`引用后端快照的对象。这将启动Trident中的快照工作流程。
   - 指定后端快照的名称 `annotations`作为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 指定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`在 `snapshotHandle`这是外部快照程序提供给Trident的唯一信息。 `ListSnapshots`称呼。

     > 这 `<volumeSnapshotContentName>`由于 CR 命名限制，有时无法与后端快照名称完全匹配。

     示例

     以下示例创建了一个 `VolumeSnapshotContent`引用后端快照的对象 `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 `VolumeSnapshot`引用 CR `VolumeSnapshotContent`目的。这是对使用权限的请求 `VolumeSnapshot`在给定的命名空间中。

   示例

   以下示例创建了一个 `VolumeSnapshot`CR命名 `import-snap`指的是 `VolumeSnapshotContent` 命名 `import-snap-content。

```
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（无需操作）： 外部快照程序识别新创建的内容 `VolumeSnapshotContent`并运行 `ListSnapshots`称呼。`Trident`创造了 `TridentSnapshot。

   - 外部快照程序设置 `VolumeSnapshotContent`到 `readyToUse`以及 `VolumeSnapshot`到 `true。
   - Trident回归 `readyToUse=true。

4. 任何用户： 创建一个 `PersistentVolumeClaim`参考新的 `VolumeSnapshot，其中 `spec.dataSource`（或者 `spec.dataSourceRef`名称是 `VolumeSnapshot`姓名。

   示例

以下示例创建一个引用 PVC 的 `VolumeSnapshot`命名 `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，快照目录是隐藏的，以确保使用以下方式配置的卷的最大兼容性： `ontap-nas`和 `ontap-nas-economy`司机。启用 `.snapshot`直接从快照恢复数据的目录。

使用卷快照恢复ONTAP CLI 将卷恢复到先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

恢复快照副本时，现有卷配置将被覆盖。创建快照副本后对卷数据所做的更改将会丢失。

从快照进行原地体积恢复

Trident利用快照提供快速、原位体积恢复功能 `TridentActionSnapshotRestore`（TASR）CR。此 CR 作为一项强制性 Kubernetes 操作，在操作完成后不会持久保存。

Trident支持快照恢复 `ontap-san`， `ontap-san-economy` ， `ontap-nas` ， `ontap-nas-flexgroup` ， `azure-netapp-files` ， `gcp-cvs` ， `google-cloud-netapp-volumes` ， 和 `solidfire-san`司机。

开始之前

您必须拥有已绑定的PVC和可用的卷快照。

- 确认PVC状态是否已绑定。

```
kubectl get pvc
```

- 确认卷快照已准备就绪。

```
kubectl get vs
```

步骤

1. 创建 TASR CR。此示例为PVC创建CR。 `pvc1`和卷快照 `pvc1-snapshot`。

   TASR CR 必须位于 PVC 和 VS 存在的命名空间中。

```
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 从快照恢复。此示例从快照恢复 `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。您可以验证快照恢复状态：

```
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: ""
```

- 大多数情况下， Trident不会在操作失败后自动重试。您需要再次执行此操作。
- 没有管理员权限的 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。

删除包含关联快照的 **PV**

删除具有关联快照的持久卷时，相应的Trident卷将更新为“正在删除”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 发行版不包含快照控制器和 CRD，您可以按如下方式部署它们。

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
```

如有必要，打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`添加到您的命名空间。

相关链接

- "卷快照"
- "卷快照类"

使用卷组快照

NetApp Trident提供了创建多个卷（一组卷快照）快照的功能，可用于创建持久卷 (PV) 的 Kubernetes 卷组快照。此卷组快照表示在同一时间点从多个卷中获取的副本。

VolumeGroupSnapshot 是 Kubernetes 中的一项测试版功能，其 API 也处于测试版阶段。VolumeGroupSnapshot 所需的最低 Kubernetes 版本为 1.32。

创建卷组快照

卷组快照受支持 `ontap-san`该驱动程序仅适用于 iSCSI 协议，尚不支持光纤通道 (FCP) 或 NVMe/TCP。开始之

前

- 请确保您的 Kubernetes 版本为 K8s 1.32 或更高版本。
- 要使用快照，您必须拥有外部快照控制器和自定义资源定义 (CRD)。这是 Kubernetes 编排器（例如：Kubeadm、GKE、OpenShift）的职责。

  如果您的 Kubernetes 发行版不包含外部快照控制器和 CRD，请参阅[部署卷快照控制器]。

  > 如果在 GKE 环境中创建按需卷组快照，则不要创建快照控制器。 GKE 使用内置的隐藏快照控制器。

- 在快照控制器 YAML 中，设置 `CSIVolumeGroupSnapshot`将功能门设置为“true”，以确保启用卷组快照。
- 在创建卷组快照之前，请先创建所需的卷组快照类。
- 确保所有 PVC/卷都在同一 SVM 上，以便能够创建 VolumeGroupSnapshot。

步骤

- 在创建 VolumeGroupSnapshot 之前，请先创建 VolumeGroupSnapshotClass。更多信息，请参阅"卷组快照类"。

```
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
```

- 使用现有的存储类别创建带有所需标签的 PVC，或者将这些标签添加到现有的 PVC。

  以下示例使用以下方法创建 PVC： `pvc1-group-snap`作为数据源和标签 `consistentGroupSnapshot: groupA`。根据您的需求定义标签的键和值。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- 创建具有相同标签的卷组快照(`consistentGroupSnapshot: groupA`）在PVC中规定。

  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

您可以使用作为卷组快照一部分创建的各个快照来恢复各个持久卷。您无法将卷组快照作为一个整体恢复。

使用卷快照恢复ONTAP CLI 将卷恢复到先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

恢复快照副本时，现有卷配置将被覆盖。创建快照副本后对卷数据所做的更改将会丢失。

从快照进行原地体积恢复

Trident利用快照提供快速、原位体积恢复功能 `TridentActionSnapshotRestore`（TASR）CR。此 CR 作为一项强制性 Kubernetes 操作，在操作完成后不会持久保存。

有关详细信息，请参阅 "从快照进行原地体积恢复"。

删除包含关联组快照的 **PV**

删除组卷快照时：

- 您可以删除整个 VolumeGroupSnapshots，而不是删除组中的单个快照。
- 如果在持久卷存在快照的情况下删除该持久卷， Trident会将该卷移至“正在删除”状态，因为必须先删除快照才能安全地删除该卷。
- 如果使用分组快照创建了克隆，然后要删除该组，则会开始在克隆时进行拆分操作，并且在拆分完成之前无法删除该组。

部署卷快照控制器

如果您的 Kubernetes 发行版不包含快照控制器和 CRD，您可以按如下方式部署它们。

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
```

如有必要，打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`添加到您的命名空间。

相关链接

- "卷组快照类"
- "卷快照"

管理和监控Trident

升级版Trident

升级版Trident

从 24.02 版本开始， Trident遵循四个月的发布节奏，每年发布三个主要版本。每个新版本都以之前的版本为基础，并提供新功能、性能增强、错误修复和改进。我们建议您至少每年升级一次，以充分利用Trident的新功能。

升级前的注意事项

升级到最新版Trident时，请注意以下事项：

- 在给定的 Kubernetes 集群的所有命名空间中，应该只安装一个Trident实例。
- Trident 23.07 及更高版本需要 v1 卷快照，不再支持 alpha 或 beta 快照。
- 如果您在 Google Cloud 中创建了Cloud Volumes Service，"CVS 服务类型"您必须更新后端配置才能使用 `standardsw`或者 `zoneredundantstandardsw`从Trident 23.01 升级时的服务级别。未能更新 `serviceLevel` 后端故障可能导致卷出现故障。参考 "CVS 服务类型示例" 了解详情。
- 升级时，请务必提供以下信息 `parameter.fsType`在 `StorageClasses`由Trident使用。您可以删除并重新创建 `StorageClasses`在不影响原有流量的情况下。
  - 这是强制执行的要求 "安全上下文"适用于 SAN 卷。
  - https://github.com/NetApp/trident/tree/master/trident-installer/sample-input [示例输入^] 目录包含示例，例如https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templ[`storage-class-basic.yaml.templ` ^] 和链接：`storage-class-bronze-default.yaml` 。
  - 更多信息，请参阅"已知问题"。

步骤 1：选择版本

Trident 的版本遵循基于日期的命名规则。 `YY.MM`命名规则中，“YY”是年份的最后两位数字，“MM”是月份。Dot 版本遵循 `YY.MM.X`按照惯例，其中“X”是补丁级别。您需要根据当前版本选择要升级到的版本。

- 您可以直接升级到与已安装版本相差不超过四个版本窗口内的任何目标版本。例如，您可以直接从 24.06（或任何 24.06 的小版本）升级到 25.06。
- 如果您要从四版本窗口之外的版本升级，请执行多步骤升级。请按照升级说明进行操作。"早期版本"您正在从当前版本升级到符合四版本发布窗口的最新版本。例如，如果您当前运行的是 23.07 版本，并且想要升级到 25.06 版本：
  a. 首次升级从 23.07 升级到 24.06。
  b. 然后从 24.06 升级到 25.06。

> 在 OpenShift 容器平台上使用Trident操作符进行升级时，应升级到Trident 21.01.1 或更高版本。随 21.01.0 版本发布的Trident操作符存在一个已知问题，该问题已在 21.01.1 版本中修复。更多详情请参阅…… "GitHub 上的问题详情" 。

步骤二：确定原始安装方法

要确定您最初安装Trident时使用的版本：

1. 使用 `kubectl get pods -n trident`检查豆荚。
   - 如果没有操作员舱，Trident是通过以下方式安装的： `tridentctl` 。
   - 如果存在 operator pod，则Trident是通过Trident operator 手动安装的，或者使用 Helm 安装的。
2. 如果有操作员舱，请使用 `kubectl describe torc`确定Trident是否使用 Helm 安装。
   - 如果存在 Helm 标签，则表示Trident是使用 Helm 安装的。
   - 如果没有 Helm 标签，则表示Trident是使用Trident操作员手动安装的。

步骤 **3**：选择升级方式

通常情况下，您应该使用与初始安装相同的方法进行升级，但是您也可以……"安装方法之间的转换" 。升级Trident有两种选择。

- "使用Trident操作员进行升级"

我们建议您查看"了解操作员升级工作流程"在与运营商升级之前。

*

通过运营商进行升级

了解操作员升级工作流程

在使用Trident操作员升级Trident之前，您应该了解升级期间发生的后台进程。这包括对Trident控制器、控制器 Pod 和节点 Pod 以及节点 DaemonSet 的更改，以启用滚动更新。

Trident操作员升级处理

众多之一"使用Trident运算符的好处"安装和升级Trident 的过程是自动处理Trident和 Kubernetes 对象，而不会中断现有的已挂载卷。这样一来， Trident就能支持零停机时间的升级，或者说，"滚动更新" 。具体来说， Trident操作符与 Kubernetes 集群通信以：

- 删除并重新创建Trident Controller 部署和节点 DaemonSet。
- 将Trident Controller Pod 和Trident Node Pod 替换为新版本。
  - 如果某个节点没有更新，并不妨碍其他节点的更新。
  - 只有运行了Trident Node Pod 的节点才能挂载卷。

有关 Kubernetes 集群上Trident架构的更多信息，请参阅："Trident架构" 。

操作员升级工作流程

当您使用Trident操作符发起升级时：

1. Trident运算符：
   a. 检测当前安装的Trident版本（版本 *n*）。
   b. 更新所有 Kubernetes 对象，包括 CRD、RBAC 和Trident SVC。
   c. 删除版本为 *n* 的Trident Controller 部署。
   d. 创建版本 *n+1* 的Trident Controller 部署。
2. **Kubernetes** 为 *n+1* 创建Trident Controller Pod。
3. Trident运算符：
   a. 删除 *n* 的Trident节点守护进程集。操作员不会等待节点 Pod 终止。
   b. 为 *n+1* 创建Trident节点守护进程集。
4. **Kubernetes** 在未运行Trident Node Pod *n* 的节点上创建Trident Node Pod。这样可以确保一个节点上永远不会有超过一个Trident Node Pod，无论版本如何。

使用**Trident Operator** 或 **Helm** 升级**Trident**安装

您可以使用Trident Operator 手动升级Trident ，也可以使用 Helm 进行升级。您可以从一个Trident Operator 安装升级到另一个Trident Operator 安装，或者从一个 Trident Operator 安装升级到另一个 Trident Operator 安装。 `tridentctl`安装到Trident操作员版本。审查"选择升级方式"在升级Trident操作程序之前。

升级手动安装

您可以从集群范围的Trident操作员安装升级到另一个集群范围的Trident操作员安装。所有Trident版本都使用集群范围的操作符。

> 要从使用命名空间作用域运算符安装的Trident （版本 20.07 至 20.10）进行升级，请使用以下升级说明："您已安装的版本"Trident。

关于此任务

Trident提供了一个捆绑文件，您可以使用该文件安装操作员并为您的 Kubernetes 版本创建关联对象。

- 对于运行 Kubernetes 1.24 的集群，请使用"bundle_pre_1_25.yaml"。
- 对于运行 Kubernetes 1.25 或更高版本的集群，请使用"bundle_post_1_25.yaml"。

开始之前

请确保您使用的是正在运行的 Kubernetes 集群。"受支持的 Kubernetes 版本" 。

步骤

1. 请验证您的Trident版本：

```
./tridentctl -n trident version
```

2. 更新 `operator.yaml`， `tridentorchestrator_cr.yaml` ， 和 `post_1_25_bundle.yaml`使用要升级到的版本（例如 25.06）的注册表和映像路径，以及正确的密钥。

3. 删除用于安装当前Trident实例的Trident操作员。例如，如果您从 25.02 升级，请运行以下命令：

```
kubectl delete -f 25.02.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

4. 如果您使用自定义方式对初始安装进行了设置 `TridentOrchestrator`属性，您可以编辑 `TridentOrchestrator`用于修改安装参数的对象。这可能包括为离线模式指定镜像Trident和 CSI 映像注册表、启用调试日志或指定映像拉取密钥所做的更改。
5. 使用适用于您环境的正确 bundle.yaml 文件安装Trident ，其中 *<bundle.yaml>* 是 `bundle_pre_1_25.yaml`或者 `bundle_post_1_25.yaml`根据您的 Kubernetes 版本。例如，如果您正在安装Trident 25.06.0，请运行以下命令：

```
kubectl create -f 25.06.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

6. 编辑三叉戟项圈以包含图像 25.06.0。

升级 **Helm** 安装

您可以升级Trident Helm 安装。

> 当将已安装Trident 的Kubernetes 集群从 1.24 版本升级到 1.25 或更高版本时，必须更新 values.yaml 文件进行设置。 `excludePodSecurityPolicy`到 `true`或添加 `--set excludePodSecurityPolicy=true`到 `helm upgrade`升级集群前必须先执行此命令。

如果您已将 Kubernetes 集群从 1.24 升级到 1.25，但未升级Trident helm，则 helm 升级将失败。要成功升级 Helm，请先执行以下步骤：

1. 从以下位置安装 helm-mapkubeapis 插件 https://github.com/helm/helm-mapkubeapis。
2. 在Trident安装所在的命名空间中，对Trident版本执行一次试运行。这里列出了需要清理的资源。

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. 使用 Helm 执行完整运行以进行清理。

```
helm mapkubeapis trident  --namespace trident
```

步骤

1. 如果你"使用 Helm 安装了Trident 。"你可以使用 `helm upgrade trident netapp-trident/trident-operator --version 100.2506.0`一步即可升级。如果您没有添加 Helm 仓库或无法使用它进行升级：
   a. 从此处下载最新版Trident"GitHub 上的 *Assets* 部分" 。
   b. 使用 `helm upgrade`其中命令 `trident-operator-25.06.0.tgz`反映您想要升级到的版本。

```
helm upgrade <name> trident-operator-25.06.0.tgz
```

如果在初始安装期间设置了自定义选项（例如，为Trident和 CSI 映像指定私有镜像注册表），请附加以下内容： `helm upgrade`命令使用 `--set`确保将这些选项包含在升级命令中，否则这些值将重置为默认值。

2. 跑步 `helm list`确认图表和应用程序版本均已升级。跑步 `tridentctl logs`查看所有调试信息。

从 `tridentctl`安装到Trident操作员

您可以从以下位置升级到最新版本的Trident操作符： `tridentctl`安装。现有的后端和PVC将自动可用。

在切换安装方法之前，请先查看"安装方法之间的转换"。

步骤

1. 下载最新版的Trident 。

```
# Download the release required [25.06.0]
mkdir 25.06.0
cd 25.06.0
wget
https://github.com/NetApp/trident/releases/download/v25.06.0/trident-
installer-25.06.0.tar.gz
tar -xf trident-installer-25.06.0.tar.gz
cd trident-installer
```

2. 创建 `tridentorchestrator`来自清单文件的 CRD。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. 在同一命名空间中部署集群范围的操作符。

```
kubectl create -f deploy/<bundle-name.yaml>

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

4. 创建一个 `TridentOrchestrator`安装Trident的 CR。

```
cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                 READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc         6/6     Running   0          1m
trident-csi-xrst8                    2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv    1/1     Running   0          5m41s
```

5. 确认Trident已升级到预期版本。

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v25.06.0
```

使用 **tridentctl** 进行升级

您可以使用以下方法轻松升级现有的Trident安装 `tridentctl`。

关于此任务

卸载并重新安装Trident相当于升级。卸载Trident时， Trident部署使用的持久卷声明 (PVC) 和持久卷 (PV) 不会被删除。在Trident离线期间，已配置的 PV 仍将可用； Trident恢复在线后，将为在此期间创建的任何 PVC 配置卷。

开始之前

审查"选择升级方式"升级前使用 `tridentctl`。

步骤

1. 运行卸载命令 `tridentctl`移除与Trident关联的所有资源，但保留 CRD 和相关对象。

```
./tridentctl uninstall -n <namespace>
```

2. 重新安装Trident。参考"使用 tridentctl 安装Trident"。

> 请勿中断升级过程。确保安装程序运行完成。

使用 **tridentctl** 管理Trident

这 "Trident安装包"包括 `tridentctl`用于提供对Trident 的简单访问的命令行实用程序。拥有足够权限的 Kubernetes 用户可以使用它来安装Trident或管理包含Trident pod 的命名空间。

命令和全局标志

你可以运行 `tridentctl help`获取可用命令列表 `tridentctl`或附加 `--help`向任何命令添加标志，即可获取该特定命令的选项和标志列表。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl`该实用程序支持以下命令和全局标志。

命令

create

向Trident添加资源。

delete

从Trident中移除一个或多个资源。

get

从Trident获取一项或多项资源。

help

关于任何命令的帮助。

images

打印一份Trident所需容器图像的表格。

import

将现有资源导入Trident。

install

安装Trident。

logs

打印Trident的日志。

send

从Trident发送资源。

uninstall

卸载Trident。

update

在Trident中修改资源。

update backend state

暂时停止后端运行。

upgrade

在Trident中升级资源。

version

打印Trident版本。

全局标志

-d，--debug

调试输出。

-h，--help

帮助 `tridentctl`。

-k，--kubeconfig string

指定 `KUBECONFIG`本地运行命令或从一个 Kubernetes 集群向另一个 Kubernetes 集群运行命令的路径。

> 或者，您可以导出 `KUBECONFIG`变量指向特定的 Kubernetes 集群并提出问题 `tridentctl`向该集群发出命令。

-n，--namespace string

Trident部署的命名空间。

-o，--output string

输出格式。 json|yaml|name|wide|ps（默认）之一。

-s，--server string

Trident REST接口的地址/端口。

> Trident REST 接口可以配置为仅监听和提供服务于 127.0.0.1（对于 IPv4）或 [::1]（对于 IPv6）。

命令选项和标志

create

使用 `create`向Trident添加资源的命令。

```
tridentctl create [option]
```

选项

`backend`为Trident添加后端。

删除

使用 `delete`从Trident中移除一个或多个资源的命令。

```
tridentctl delete [option]
```

选项

`backend`从Trident中删除一个或多个存储后端。
`snapshot`从Trident中删除一个或多个卷快照。

`storageclass`从Trident中删除一个或多个存储类。
`volume`从Trident中删除一个或多个存储卷。

得到

使用 `get`从Trident获取一个或多个资源的命令。

```
tridentctl get [option]
```

选项

`backend`从Trident获取一个或多个存储后端。
`snapshot`从Trident获取一个或多个快照。
`storageclass`从Trident获取一个或多个存储类。
`volume`从Trident获取一卷或多卷。

旗帜

`-h，--help`：有关卷的帮助。
`--parentOfSubordinate string`：将查询限制在子源卷上。
`--subordinateOf string`：将查询限制在卷的下级。

图片

使用 `images`用于打印Trident所需容器图像表格的标志。

```
tridentctl images [flags]
```

旗帜

`-h，--help`图片帮助。
`-v，--k8s-version string` Kubernetes 集群的语义版本。

进口量

使用 `import volume`将现有卷导入Trident 的命令。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

别名

`volume，v`

旗帜

`-f，--filename string`：YAML 或 JSON PVC 文件的路径。
`-h，--help`：音量帮助。
`--no-manage`仅创建 PV/PVC。不要想当然地认为存在销量生命周期管理。

安装

使用 `install`安装Trident 的标志。

```
tridentctl install [flags]
```

旗帜

--autosupport-image string：自动支持遥测的容器镜像（默认为“netapp/trident autosupport:<当前版本>”）。
--autosupport-proxy string`用于发送 Autosupport 遥测数据的代理地址/端口。
`--enable-node-prep`尝试在节点上安装所需的软件包。
`--generate-custom-yaml`无需安装任何软件即可生成 YAML 文件。
`-h， --help`安装帮助。
`--http-request-timeout： 覆盖Trident控制器 REST API 的 HTTP 请求超时时间（默认 1 分 30 秒）。
--image-registry string： 内部镜像仓库的地址/端口。
--k8s-timeout duration： 所有 Kubernetes 操作的超时时间（默认 3 分钟）。
--kubelet-dir string： kubelet 内部状态的主机位置（默认为“/var/lib/kubelet”）。
--log-format string Trident日志格式（文本、JSON）（默认“文本”）。
--node-prep： 使Trident能够准备 Kubernetes 集群的节点，以使用指定的数据存储协议来管理卷。 现在， **iscsi**`是唯一支持的值。从 **OpenShift 4.19** 开始，此功能支持的最低**Trident**版本为 **25.06.1.**
`**--pv string**： Trident使用的旧版 PV 的名称，确保它不存在（默认值为“trident”）。
--pvc string： Trident使用的旧版 PVC 的名称，确保它不存在（默认值为“trident”）。
--silence-autosupport： 不要自动向NetApp发送自动支持包（默认为 true）。
--silent`安装过程中禁用大部分输出。
`--trident-image string`要安装的Trident镜像。
`--k8s-api-qps Kubernetes API 请求的每秒查询数 (QPS) 限制（默认为 100；可选）。
--use-custom-yaml`使用安装目录中已存在的任何 YAML 文件。
`--use-ipv6： Trident 通信使用 IPv6。

logs

使用 `logs`用于打印Trident日志的标志。

```
tridentctl logs [flags]
```

旗帜

-a， --archive`除非另有规定，否则请创建包含所有日志的支持存档。
`-h， --help： 日志帮助。
-l， --log string`要显示的Trident日志。 trident|auto|trident-operator|all（默认值“auto”）之一。
`--node string： 要从中收集节点 pod 日志的 Kubernetes 节点名称。
-p， --previous： 获取先前容器实例的日志（如果存在）。
`--sidecars`获取边车容器的日志。

发送

使用 `send`从Trident发送资源的命令。

```
tridentctl send [option]
```

选项

`autosupport`将 Autosupport 归档文件发送给NetApp。

卸载

使用 `uninstall`卸载Trident 的标志。

```
tridentctl uninstall [flags]
```

旗帜

-h, --help：卸载帮助。
`--silent`卸载过程中禁用大部分输出。

更新

使用 `update`在Trident中修改资源的命令。

```
tridentctl update [option]
```

选项

backend：在Trident中更新后端。

更新后端状态

使用 `update backend state`暂停或恢复后端操作的命令。

```
tridentctl update backend state <backend-name> [flag]
```

需要考虑的几点

- 如果后端是使用 TridentBackendConfig (tbc) 创建的，则无法使用以下方式更新后端： `backend.json`文件。
- 如果 `userState`该值已在待办事项列表中设置，无法使用以下方式修改： `tridentctl update backend state <backend-name> --user-state suspended/normal`命令。
- 重新获得设置能力 `userState`通过 tridentctl，在通过 tbc 设置之后， `userState`必须从待办事项列表中移除该字段。这可以通过以下方式完成： `kubectl edit tbc`命令。之后 `userState`字段已移除，您可以使用 `tridentctl update backend state`更改命令 `userState`后端。
- 使用 `tridentctl update backend state`改变 `userState`。您还可以更新 `userState`使用 `TridentBackendConfig`或者 `backend.json`文件；这将触发后端完全重新初始化，可能会很耗时。

  旗帜

  -h， --help ： 后端状态帮助。
  --user-state`设置为 `suspended`暂停后端操作。设置为 `normal`恢复后端操作。设置为 `suspended：

- `AddVolume`和 `Import Volume`暂停。
- CloneVolume， ResizeVolume ， PublishVolume ， UnPublishVolume ， CreateSnapshot ， GetSnapshot ， RestoreSnapshot ， DeleteSnapshot ， RemoveVolume ， GetVolumeExternal ， `ReconcileNodeAccess`仍然可用。

您还可以使用以下方式更新后端状态 userState`后端配置文件中的字段 `TridentBackendConfig`或者 `backend.json。更多信息，请参阅"后端管理选项"和"使用 kubectl 执行后端管理"。

例子：

JSON

请按照以下步骤进行更新 `userState`使用 `backend.json`文件：

1. 编辑 `backend.json`要包含的文件 `userState`字段值设置为“已暂停”。
2. 使用以下方式更新后端 `tridentctl update backend`命令和更新后的路径 `backend.json`文件。

   例子： `tridentctl update backend -f /<path to backend JSON file>/backend.json -n trident`

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "<redacted>",
  "svm": "nas-svm",
  "backendName": "customBackend",
  "username": "<redacted>",
  "password": "<redacted>",
  "userState": "suspended"
}
```

YAML

应用待办事项后，您可以使用以下方法对其进行编辑： `kubectl edit <tbc-name> -n <namespace>`命令。以下示例使用以下方式将后端状态更新为挂起： `userState: suspended`选项：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
  userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

version

使用 `version`标志用于打印版本 `tridentctl`以及正在运行的Trident服务。

```
tridentctl version [flags]
```

旗帜

`--client`：仅限客户端版本（无需服务器）。
`-h, --help`：版本帮助。

插件支持

Tridentctl 支持类似于 kubectl 的插件。如果插件二进制文件名遵循“tridentctl-<plugin>”格式，并且该二进制文件位于 PATH 环境变量列出的文件夹中，则 Tridentctl 会检测到该插件。所有检测到的插件都列在 tridentctl 帮助的插件部分中。您还可以通过在环境变量 TRIDENTCTL_PLUGIN_PATH 中指定插件文件夹来限制搜索范围（例如： `TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/` ）。如果使用该变量，tridenctl 将仅在指定的文件夹中搜索。

监视Trident

Trident提供了一组 Prometheus 指标端点，您可以使用这些端点来监控Trident 的性能。

概述

Trident提供的指标使您能够执行以下操作：

- 密切关注 Trident 的健康状况和配置。您可以检查操作是否成功，以及是否能按预期与后端通信。
- 检查后端使用情况信息，了解后端配置了多少卷、消耗了多少空间等等。
- 维护可用后端已配置卷量的映射表。
- 赛道表现。您可以查看Trident与后端通信和执行操作所需的时间。

> 默认情况下，Trident 的指标会暴露在目标端口上。 `8001`在 `/metrics`端点。安装Trident时，这些指标*默认启用*。

你需要什么

- 已安装Trident的 Kubernetes 集群。
- 普罗米修斯实例。这可能是一个 "容器化的 Prometheus 部署"或者您可以选择以某种方式运行 Prometheus "本机应用程序" 。

步骤 1：定义 Prometheus 目标

您应该定义一个 Prometheus 目标来收集指标并获取有关Trident管理的后端、它创建的卷等的信息。这 "博客"解释了如何将 Prometheus 和 Grafana 与Trident结合使用来检索指标。该博客解释了如何在 Kubernetes 集群中以 Operator 的形式运行 Prometheus，以及如何创建 ServiceMonitor 来获取Trident指标。

步骤 2：创建 Prometheus 服务监视器

要使用Trident指标，您应该创建一个 Prometheus ServiceMonitor 来监视 `trident-csi`服务和监听 `metrics`港口。一个示例 ServiceMonitor 如下所示：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - port: metrics
      interval: 15s
```

此 ServiceMonitor 定义检索由以下方式返回的指标： `trident-csi`服务，特别是寻找 `metrics`服务的端点。因此，Prometheus 现在已配置为能够理解 Trident 的指标。

除了Trident直接提供的指标外，kubelet 还公开了许多其他指标。 `kubelet_volume_*`通过其自身的指标端点获取指标。 Kubelet 可以提供有关已附加卷、Pod 以及它处理的其他内部操作的信息。参考 "此处"。

步骤 3：使用 PromQL 查询Trident指标

PromQL 非常适合创建返回时间序列或表格数据的表达式。

以下是一些您可以使用的 PromQL 查询：

获取Trident健康信息

- Trident服务器 HTTP 2XX 响应的百分比

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Trident REST** 响应状态码百分比

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Trident**执行操作的平均持续时间（毫秒）

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

获取**Trident**使用信息

- 平均体积大小

```
trident_volume_allocated_bytes/trident_volume_count
```

- 各后端分配的总存储空间

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

获取个人用量

只有在同时收集 kubelet 指标的情况下才会启用此功能。

- 各卷册的空间利用率

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

了解**Trident AutoSupport**遥测技术

默认情况下， Trident每天向NetApp发送 Prometheus 指标和基本后端信息。

- 要阻止Trident向NetApp发送 Prometheus 指标和基本后端信息，请传递以下参数： `--silence-autosupport` 在Trident安装过程中发出警报。
- Trident还可以按需通过以下方式将容器日志发送给NetApp支持： `tridentctl send autosupport` 。您需要触发Trident上传其日志。提交日志之前，您应该接受 NetApp 的条款。https://www.netapp.com/company/legal/privacy-policy/["隐私政策"^] 。
- 除非另有说明， Trident会获取过去 24 小时的日志。
- 您可以使用以下方式指定日志保留时间范围： `--since`旗帜。例如： `tridentctl send autosupport --since=1h` 。这些信息通过以下方式收集和发送： `trident-autosupport`与Trident一起安装的容器。您可以从以下位置获取容器镜像： "Trident AutoSupport" 。
- Trident AutoSupport不会收集或传输个人身份信息 (PII) 或个人信息。它附带一个 "最终用户许可协议"这并不适用于Trident容器镜像本身。您可以了解更多关于NetApp对数据安全和信任的承诺。 "此处" 。

Trident发送的有效载荷示例如下所示：

```
---
items:
  - backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
    protocol: file
    config:
      version: 1
      storageDriverName: ontap-nas
      debug: false
      debugTraceFlags: null
      disableDelete: false
      serialNumbers:
        - nwkvzfanek_SN
      limitVolumeSize: ""
    state: online
    online: true
```

- AutoSupport消息会发送到 NetApp 的AutoSupport端点。如果您使用私有镜像仓库来存储容器镜像，则可以使用 `--image-registry`旗帜。
- 您还可以通过生成安装 YAML 文件来配置代理 URL。这可以通过使用 `tridentctl install --generate-custom-yaml`创建 YAML 文件并添加 `--proxy-url`论证 `trident-autosupport`容器 `trident-deployment.yaml`。

禁用Trident指标

要禁用指标报告，您应该生成自定义 YAML 文件（使用以下方式）： `--generate-custom-yaml`标记）并编辑它们以删除 `--metrics`阻止对 `trident-main`容器。

卸载Trident

卸载Trident时，应使用与安装Trident时相同的方法。

关于此任务

- 如果升级后出现错误、依赖项问题或升级失败/不完整，需要修复，则应卸载Trident并按照相应的说明重新安装早期版本。"version" 。这是降级到早期版本的唯一推荐方法。
- 为了方便升级和重新安装，卸载Trident不会删除Trident创建的 CRD 或相关对象。如果您需要彻底删除Trident及其所有数据，请参阅以下内容："彻底移除Trident和CRDs" 。

开始之前

如果您要停用 Kubernetes 集群，则必须先删除所有使用Trident创建的卷的应用程序，然后再卸载。这样可以确保在 Kubernetes 节点上删除 PVC 之前，PVC 会被取消发布。

确定原始安装方法

卸载Trident时，应使用与安装 Trident 时相同的方法。卸载前，请确认您最初安装Trident时使用的版本。

1. 使用 `kubectl get pods -n trident`检查豆荚。
   - 如果没有操作员舱，Trident是通过以下方式安装的： `tridentctl` 。
   - 如果存在 operator pod，则Trident是通过Trident operator 手动安装的，或者使用 Helm 安装的。
2. 如果有操作员舱，请使用 `kubectl describe tproc trident`确定Trident是否使用 Helm 安装。
   - 如果存在 Helm 标签，则表示Trident是使用 Helm 安装的。
   - 如果没有 Helm 标签，则表示Trident是使用Trident操作员手动安装的。

卸载Trident操作员安装程序

您可以手动卸载 Trident Operator 安装，也可以使用 Helm 卸载。

手动卸载

如果您使用操作员安装了Trident ，则可以通过执行以下操作之一来卸载它：

1. 编辑 **`TridentOrchestrator`CR** 并设置卸载标志：

   ```
   kubectl patch torc <trident-orchestrator-name> --type=merge -p
   '{"spec":{"uninstall":true}}'
   ```

   当 `uninstall`标志位已设置为 `true`Trident操作符会卸载Trident，但不会移除 TridentOrchestrator 本身。如果您想再次安装Trident，则应该清理 TridentOrchestrator 并创建一个新的。

2. 删除 **`TridentOrchestrator`**：通过移除 `TridentOrchestrator`如果 CR 用于部署Trident，则指示操作员卸载Trident。操作员处理移除操作 `TridentOrchestrator`然后继续移除Trident部署和守护进程集，并删除安装过程中创建的Trident pod。

   ```
   kubectl delete -f deploy/<bundle.yaml> -n <namespace>
   ```

卸载 **Helm** 安装

如果您使用 Helm 安装了Trident ，则可以使用以下命令卸载它： `helm uninstall` 。

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

卸载 `tridentctl`安装

使用 `uninstall`命令 `tridentctl`移除与Trident关联的所有资源，但保留 CRD 及其相关对象：

```
./tridentctl uninstall -n <namespace>
```

Trident for Docker

部署先决条件

在部署Trident之前，您必须在主机上安装和配置必要的协议先决条件。

核实要求

- 确认您的部署满足所有要求"要求"。
- 请确认您已安装受支持的 Docker 版本。如果您的 Docker 版本已过时， "安装或更新它" 。

```
docker --version
```

- 请确认您的主机上已安装并配置协议所需的先决条件。

NFS 工具

使用适用于您操作系统的命令安装 NFS 工具。

RHEL 8+

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

Ubuntu

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

安装 NFS 工具后重启工作节点，以防止将卷附加到容器时发生故障。

iSCSI 工具

使用适用于您操作系统的命令安装 iSCSI 工具。

RHEL 8+

1. 安装以下系统软件包：

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-
mapper-multipath
```

2. 请检查 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. 将扫描方式设置为手动：

```
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
```

确保 `etc/multipath.conf`包含 `find_multipaths no`在下面 `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. 请检查 open-iscsi 版本是否为 2.0.874-5ubuntu2.10 或更高版本（适用于 bionic）或 2.0.874-7.1ubuntu6.1 或更高版本（适用于 focal）：

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

3. 将扫描方式设置为手动：

```
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
```

确保 `etc/multipath.conf`包含 `find_multipaths no`在下面 `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
```

NVMe 工具

使用适用于您操作系统的命令安装 NVMe 工具。

- NVMe 需要 RHEL 9 或更高版本。
- 如果您的 Kubernetes 节点的内核版本太旧，或者您的内核版本没有 NVMe 软件包，则您可能需要将节点的内核版本更新为包含 NVMe 软件包的版本。

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
```

FC工具

使用适用于您操作系统的命令安装 FC 工具。

- 当使用运行 RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 且带有 FC PV 的工作节点时，请指定 `discard`StorageClass 中的 mountOption 用于执行内联空间回收。参考 "红帽文档"。

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
```

确保 `etc/multipath.conf`包含 `find_multipaths no`在下面 `defaults。

3. 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
```

确保 `etc/multipath.conf`包含 `find_multipaths no`在下面 `defaults。

3. 确保 `multipath-tools`已启用并正在运行：

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

部署Trident

Trident for Docker 为NetApp存储平台提供与 Docker 生态系统的直接集成。它支持从存储平台到 Docker 主机的存储资源的配置和管理，并提供了一个框架，以便将来添加其他平台。

同一主机上可以同时运行多个Trident实例。这样就可以同时连接到多个存储系统和存储类型，并且可以自定义 Docker 卷使用的存储。

你需要什么

查看"部署的先决条件"。确保满足先决条件后，即可部署Trident。

Docker 管理的插件方法（版本 1.13/17.03 及更高版本）

开始之前

如果您之前在 Docker 1.13/17.03 之前的版本中使用过Trident 的传统守护进程方法，请确保在使用托管插件方法之前停止Trident进程并重新启动 Docker 守护进程。

1. 停止所有正在运行的实例：

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. 重启 Docker。

```
systemctl restart docker
```

3. 请确保您已安装 Docker Engine 17.03（新版本 1.13）或更高版本。

```
docker --version
```

如果您的版本已过时， "安装或更新您的安装程序" 。

步骤

1. 创建配置文件并按如下方式指定选项：
   - `config`默认文件名是 `config.json`不过，您可以通过指定名称来使用您选择的任何名称。 `config`文件名选项。配置文件必须位于 `/etc/netappdvp`主机系统上的目录。
   - `log-level`：指定日志级别(`debug`， `info` ， `warn` ， `error` ， `fatal` ）。默认值为 `info`。
   - `debug`：指定是否启用调试日志记录。默认值为 false。如果为真，则覆盖日志级别。
     i. 创建配置文件存放位置：

```
sudo mkdir -p /etc/netappdvp
```

ii. 创建配置文件：

```
cat << EOF > /etc/netappdvp/config.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. 使用托管插件系统启动Trident 。代替 `<version>`使用您正在使用的插件版本（xxx.xx.x）。

```
docker plugin install --grant-all-permissions --alias netapp
netapp/trident-plugin:<version> config=myConfigFile.json
```

3. 开始使用Trident从已配置的系统中获取存储空间。

a. 创建一个名为“firstVolume”的卷：

```
docker volume create -d netapp --name firstVolume
```

b. 容器启动时创建默认卷：

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

c. 移除音量“firstVolume”：

```
docker volume rm firstVolume
```

传统方法（版本 **1.12** 或更早版本）

开始之前

1. 请确保您使用的是 Docker 版本 1.10 或更高版本。

```
docker --version
```

如果您的版本过旧，请更新您的安装。

```
curl -fsSL https://get.docker.com/ | sh
```

或者， "请按照您的分发说明进行操作" 。

2. 请确保您的系统已配置 NFS 和/或 iSCSI。

步骤

1. 安装并配置NetApp Docker 卷插件：

a. 下载并解压应用程序：

```
wget
https://github.com/NetApp/trident/releases/download/v25.06.0/trident-
installer-25.06.0.tar.gz
tar zxf trident-installer-25.06.0.tar.gz
```

b. 移动到回收站路径中的某个位置：

```
sudo mv trident-installer/extras/bin/trident /usr/local/bin/
sudo chown root:root /usr/local/bin/trident
sudo chmod 755 /usr/local/bin/trident
```

c. 创建配置文件存放位置：

```
sudo mkdir -p /etc/netappdvp
```

d. 创建配置文件：

```
cat << EOF > /etc/netappdvp/ontap-nas.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
EOF
```

2. 放置好二进制文件并创建配置文件后，使用所需的配置文件启动Trident守护进程。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

除非另有指定，卷驱动程序的默认名称为“netapp”。

守护进程启动后，您可以使用 Docker CLI 界面创建和管理卷。

3. 创建卷：

```
docker volume create -d netapp --name trident_1
```

4. 启动容器时配置 Docker 卷：

```
docker run --rm -it --volume-driver netapp --volume trident_2:/my_vol
alpine ash
```

5. 删除 Docker 卷：

```
docker volume rm trident_1
```

```
docker volume rm trident_2
```

系统启动时启动Trident

systemd 系统的示例单元文件可在以下位置找到： `contrib/trident.service.example`在 Git 仓库中。要在 RHEL 系统中使用该文件，请执行以下操作：

1. 将文件复制到正确位置。

   如果运行多个实例，则应为单元文件使用唯一的名称。

   ```
   cp contrib/trident.service.example
   /usr/lib/systemd/system/trident.service
   ```

2. 编辑该文件，将描述（第 2 行）更改为与驱动程序名称匹配，并将配置文件路径（第 9 行）更改为反映您的环境。
3. 重新加载 systemd 以使其吸收更改：

   ```
   systemctl daemon-reload
   ```

4. 启用该服务。

   这个名称会根据你给文件命名的方式而有所不同。 `/usr/lib/systemd/system`目录。

   ```
   systemctl enable trident
   ```

5. 启动服务。

   ```
   systemctl start trident
   ```

6. 查看状态。

   ```
   systemctl status trident
   ```

每次修改单元文件后，请运行以下命令： `systemctl daemon-reload`命令使其能够感知这些变化。

升级或卸载Trident

您可以安全地升级Trident for Docker，而不会对正在使用的卷产生任何影响。升级过程中会有一段短暂的时间， `docker volume`针对该插件的命令将不会成功，应用程序将无法挂载卷，直到该插件再次运行。大多数情况下，这只需要几秒钟。

升级

按照以下步骤升级Trident for Docker。

步骤

1. 列出现有卷册：

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

2. 禁用插件：

```
docker plugin disable -f netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. 升级插件：

```
docker plugin upgrade --skip-remote-check --grant-all-permissions
netapp:latest netapp/trident-plugin:21.07
```

> Trident 18.01 版本取代了 nDVP。你应该直接从以下位置升级： `netapp/ndvp-plugin`图片到`netapp/trident-plugin`图像。

4. 启用插件：

```
docker plugin enable netapp:latest
```

5. 请确认插件已启用：

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. 确认卷可见：

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

如果您从旧版本的Trident （20.10 之前）升级到Trident 20.10 或更高版本，则可能会遇到错误。更多信息，请参阅"已知问题"。如果遇到此错误，您应该先禁用该插件，然后删除该插件，最后通过传递额外的配置参数来安装所需的Trident版本： `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

卸载

按照以下步骤卸载Trident for Docker。

步骤

1. 删除插件创建的所有卷。
2. 禁用插件：

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. 移除插件：

```
docker plugin rm netapp:latest
```

使用卷

您可以使用标准方法轻松创建、克隆和删除卷 `docker volume`需要时，请使用指定Trident驱动程序名称的命令。

创建卷

- 使用默认名称创建带有驱动程序的卷：

```
docker volume create -d netapp --name firstVolume
```

- 创建包含特定Trident实例的卷：

```
docker volume create -d ntap_bronze --name bronzeVolume
```

如果您没有指定任何内容"options"使用的是驱动程序的默认设置。

- 覆盖默认卷大小。请参阅以下示例，使用驱动程序创建 20 GiB 的卷：

```
docker volume create -d netapp --name my_vol --opt size=20G
```

卷大小以字符串形式表示，其中包含一个整数值，单位可选（例如：10G、20GB、3TiB）。如果没有指定单位，则默认值为 G。大小单位可以表示为 2 的幂（B、KiB、MiB、GiB、TiB）或 10 的幂（B、KB、MB、GB、TB）。简写单位使用 2 的幂（G = GiB，T = TiB，...）。

移除音量

- 删除该卷的方式与其他 Docker 卷一样：

```
docker volume rm firstVolume
```

使用时 `solidfire-san`上述示例驱动程序删除并清除卷。

按照以下步骤升级Trident for Docker。

克隆卷

使用时 `ontap-nas`， `ontap-san`， `solidfire-san`， 和 `gcp-cvs storage drivers`Trident可以克隆卷。使用时 `ontap-nas-flexgroup`或者 `ontap-nas-economy`驱动程序不支持克隆。从现有卷创建新卷将创建一个新的快照。

- 检查卷以枚举快照：

```
docker volume inspect <volume_name>
```

- 从现有卷创建新卷。这将导致创建一个新的快照：

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume>
```

- 从现有卷上的快照创建新卷。这不会创建新的快照：

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume> -o fromSnapshot=<source_snap_name>
```

示例

```
docker volume inspect firstVolume

[
  {
    "Driver": "ontap-nas",
    "Labels": null,
    "Mountpoint": "/var/lib/docker-volumes/ontap-
nas/netappdvp_firstVolume",
    "Name": "firstVolume",
    "Options": {},
    "Scope": "global",
    "Status": {
      "Snapshots": [
        {
          "Created": "2017-02-10T19:05:00Z",
          "Name": "hourly.2017-02-10_1505"
        }
      ]
    }
  }
]

docker volume create -d ontap-nas --name clonedVolume -o from=firstVolume
clonedVolume

docker volume rm clonedVolume
docker volume create -d ontap-nas --name volFromSnap -o from=firstVolume
-o fromSnapshot=hourly.2017-02-10_1505
volFromSnap

docker volume rm volFromSnap
```

访问外部创建的卷

只有当外部创建的块设备（或其克隆）没有分区且其文件系统受Trident支持时（例如：一个），容器才能使用Trident访问这些块设备（或其克隆）。 `ext4`格式化 `/dev/sdc1`无法通过Trident访问）。

驱动程序特定音量选项

每个存储驱动程序都有一组不同的选项，您可以在创建卷时指定这些选项以自定义结果。请参阅下方适用于您配置的存储系统的选项。

在创建卷操作期间使用这些选项非常简单。提供选项和值 `-o`CLI 操作期间的运算符。这些值会覆盖 JSON 配置文件中的任何等效值。

ONTAP容量选项

NFS、iSCSI 和 FC 的卷创建选项包括以下几种：

选项	描述
`size`	卷大小默认为 1 GiB。
`spaceReserve`	容量配置方式可以是精简配置或厚配置，默认为精简配置。有效值为 `none`（精简配置）和 `volume`（厚配置）。
`snapshotPolicy`	这将把快照策略设置为所需值。默认值为 `none`这意味着不会自动为该卷创建快照。除非存储管理员修改，否则所有ONTAP系统上都存在一个名为“default”的策略，该策略会创建并保留 6 个每小时快照、2 个每日快照和 2 个每周快照。可以通过浏览来恢复快照中保存的数据。 `.snapshot`卷中任意目录中的目录。
`snapshotReserve`	这将把快照预留量设置为所需的百分比。默认值为空，这意味着如果您选择了 snapshotPolicy， ONTAP将选择 snapshotReserve（通常为 5%）；如果 snapshotPolicy 为 none，则 ONTAP 将选择 snapshotReserve（通常为 0%）。您可以在配置文件中为所有ONTAP后端设置默认 snapshotReserve 值，并且可以将其用作除 ontap-nas-economy 之外的所有ONTAP后端的卷创建选项。
`splitOnClone`	克隆卷时，这将导致ONTAP立即将克隆卷与其父卷分离。默认值为 `false`。有些克隆卷的用例最好在创建后立即将克隆卷与其父卷分离，因为不太可能有机会提高存储效率。例如，克隆一个空数据库可以节省大量时间，但节省的存储空间却很少，因此最好立即拆分克隆。
`encryption`	在新卷上启用NetApp卷加密 (NVE)；默认设置为 `false`。要使用此选项，必须在集群上获得 NVE 许可并启用 NVE。 如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。 更多信息，请参阅："Trident如何与 NVE 和 NAE 协同工作" 。
`tieringPolicy`	设置卷要使用的分层策略。这决定了当数据变为不活动状态（冷数据）时，是否将其迁移到云层。

以下附加选项仅适用于 NFS：

选项	描述
unixPermissions	这控制卷本身的权限集。默认情况下，权限将设置为 ``---rwxr-xr-x`或者用数字表示法表示为 0755，并且 `root`将成为所有者。文本格式或数字格式均可。
snapshotDir	将此设置为 `true`将使 `.snapshot`客户端访问该卷时可见的目录。默认值为 `false`这意味着可见性 `.snapshot`目录默认处于禁用状态。某些镜像，例如官方的 MySQL 镜像，在以下情况下无法按预期工作：`.snapshot`目录可见。
exportPolicy	设置该卷要使用的导出策略。默认值为 default。
securityStyle	设置用于访问卷的安全样式。默认值为 unix。有效值为 unix`和 `mixed。

以下附加选项仅适用于 iSCSI：

选项	描述
fileSystemType	设置用于格式化 iSCSI 卷的文件系统。默认值为 ext4。有效值为 ext3， ext4 ， 和 xfs。
spaceAllocation	将其设置为 `false`将关闭 LUN 的空间分配功能。默认值为 `true`这意味着当卷空间不足且卷中的 LUN 无法接受写入时， ONTAP会通知主机。此选项还允许ONTAP在主机删除数据时自动回收空间。

示例

请看以下示例：

- 创建一个 10 GiB 卷：

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- 创建一个带有快照的 100 GiB 卷：

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- 创建一个启用了 setUID 位的卷：

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

最小卷大小为 20 MiB。

如果未指定快照保留，且快照策略为 `none`Trident使用 0% 的快照储备。

- 创建一个没有快照策略和快照保留的卷：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- 创建一个没有快照策略且自定义快照保留比例为 10% 的卷：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- 创建一个具有快照策略和 10% 自定义快照保留空间的卷：

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- 创建具有快照策略的卷，并接受 ONTAP 的默认快照保留（通常为 5%）：

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element 软件音量选项

Element 软件选项会显示与卷相关的大小和服务质量 (QoS) 策略。创建卷时，使用以下方式指定与其关联的 QoS 策略： `-o type=service_level`命名法。

使用 Element 驱动程序定义 QoS 服务级别的第一步是创建至少一个类型，并在配置文件中指定与名称关联的最小、最大和突发 IOPS。

Element软件的其他卷创建选项包括以下几种：

选项	描述
`size`	卷的大小，默认为 1 GiB 或配置条目…“defaults”：{“size”：“5G”}。
`blocksize`	可以使用 512 或 4096，默认值为 512 或配置项 DefaultBlockSize。

示例

请参阅以下包含 QoS 定义的示例配置文件：

```
{
  "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
      }
    }
  ]
}
```

在上述配置中，我们有三个策略定义：青铜、白银和黄金。这些名称是随意起的。

- 创建 10 GiB 黄金卷：

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 创建 100 GiB 青铜卷：

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

收集日志

您可以收集日志以帮助进行故障排除。收集日志的方法取决于您运行 Docker 插件的方式。

收集日志以进行故障排除

步骤

1. 如果您使用推荐的托管插件方法运行Trident （即，使用 `docker plugin`命令），可以这样理解：

```
docker plugin ls
```

```
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin    false
journalctl -u docker | grep 4fb97d2b956b
```

标准日志级别应该足以诊断大多数问题。如果这还不够，您可以启用调试日志记录。

2. 要启用调试日志记录，请安装启用调试日志记录功能的插件：

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

或者，在插件已安装的情况下启用调试日志记录：

```
docker plugin disable <plugin>
```

```
docker plugin set <plugin> debug=true
```

```
docker plugin enable <plugin>
```

3. 如果您在主机上运行该二进制文件，则日志位于主机的目录中。 `/var/log/netappdvp`目录。要启用调试日志记录，请指定 `-debug`当你运行插件时。

一般故障排除技巧

- 新用户遇到的最常见问题是配置错误，导致插件无法初始化。发生这种情况时，当您尝试安装或启用插件时，可能会看到类似这样的消息：

```
Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
connect: no such file or directory
```

这意味着插件启动失败。幸运的是，该插件内置了全面的日志记录功能，应该可以帮助您诊断可能遇到的大多数问题。

- 如果将光伏系统安装到集装箱上遇到问题，请确保： `rpcbind`已安装并正在运行。使用主机操作系统所需的软件包管理器并检查是否 `rpcbind`正在运行。您可以通过运行以下命令来检查 rpcbind 服务的状态： `systemctl status rpcbind`或其等效物。

管理多个Trident实例

当您需要同时拥有多个存储配置时，需要多个Trident实例。实现多个实例的关键在于使用不同的名称来命名它们。 `--alias`使用容器化插件的选项，或者 `--volume-driver`在主机上实例化Trident时的选择。

Docker 管理插件（版本 1.13/17.03 或更高版本）的步骤

1. 启动第一个实例时，指定别名和配置文件。

```
docker plugin install --grant-all-permissions --alias silver
netapp/trident-plugin:21.07 config=silver.json
```

2. 启动第二个实例，指定不同的别名和配置文件。

```
docker plugin install --grant-all-permissions --alias gold
netapp/trident-plugin:21.07 config=gold.json
```

3. 创建卷时，将别名指定为驱动程序名称。

   例如，黄金交易量：

```
docker volume create -d gold --name ntapGold
```

   例如，对于白银交易量而言：

```
docker volume create -d silver --name ntapSilver
```

传统方法（版本 1.12 或更早版本）的步骤

1. 使用自定义驱动程序 ID 通过 NFS 配置启动插件：

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. 使用自定义驱动程序 ID 和 iSCSI 配置启动插件：

```
sudo trident --volume-driver=netapp-san --config=/path/to/config
-iscsi.json
```

3. 为每个驱动程序实例配置 Docker 卷：

   例如，对于 NFS：

```
docker volume create -d netapp-nas --name my_nfs_vol
```

   例如，对于 iSCSI：

```
docker volume create -d netapp-san --name my_iscsi_vol
```

存储配置选项

查看适用于您的Trident配置的配置选项。

全局配置选项

这些配置选项适用于所有Trident配置，无论使用何种存储平台。

选项	描述	示例
`version`	配置文件版本号	`1`
`storageDriverName`	存储驱动程序名称	`ontap-nas`， `ontap-san` ，`ontap-nas-economy` ，`ontap-nas-flexgroup` ，`solidfire-san`
`storagePrefix`	卷名称的可选前缀。默认：`netappdvp_` 。	`staging_`
`limitVolumeSize`	对容量大小有可选限制。默认值：""（不强制执行）	`10g`

请勿使用 `storagePrefix`（包括默认值）适用于 Element 后端。默认情况下， `solidfire-san` 驱动程序将忽略此设置，并且不使用前缀。 NetApp建议使用特定的租户 ID 进行 Docker 卷映射，或者使用属性数据，该属性数据填充了 Docker 版本、驱动程序信息和来自 Docker 的原始名称（如果可能使用了任何名称修改）。

可以使用默认选项，避免在创建的每个卷上都指定这些选项。这 `size`此选项适用于所有控制器类型。有关如何设置默认卷大小的示例，请参阅ONTAP配置部分。

选项	描述	示例
`size`	新建卷的可选默认大小。默认： `1G`	`10G`

ONTAP 配置

除了上述全局配置值之外，使用ONTAP时，还可以使用以下顶级选项。

选项	描述	示例
`managementLIF`	ONTAP管理 LIF 的 IP 地址。您可以指定一个完全限定域名（FQDN）。	`10.0.0.1`
`dataLIF`	LIF协议的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。您可以指定一个完全限定域名 (FQDN) 用于 NFS 挂载操作，从而创建轮询 DNS 以在多个 dataLIF 之间进行负载均衡。 • ONTAP SAN 驱动程序*：请勿指定 iSCSI 或 FC。Trident的使用"ONTAP选择性 LUN 地图"发现建立多路径会话所需的 iSCSI 或 FC LIF。如果出现以下情况，则会生成警告： `dataLIF`已明确定义。	`10.0.0.2`
`svm`	要使用的存储虚拟机（如果管理 LIF 是集群 LIF，则此项为必填项）	`svm_nfs`
`username`	连接到存储设备的用户名	`vsadmin`
`password`	连接存储设备的密码	`secret`

选项	描述	示例
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。分配给 SVM 的所有聚合都用于配置FlexGroup卷。	aggr1
limitAggregateUsage	可选，如果使用率超过此百分比，则配置失败	75%
nfsMountOptions	对 NFS 挂载选项进行精细控制；默认值为“-o nfsvers=3”。 仅限以下情况： **`ontap-nas`**和 **`ontap-nas-economy`**司机。 "请在此处查看 NFS 主机配置信息" 。	-o nfsvers=4
igroupName	Trident创建和管理每个节点 igroups`作为 `netappdvp。 此值不能更改或省略。 仅限以下情况： **`ontap-san`**司机。	netappdvp
limitVolumeSize	最大可请求容量。	300g
qtreesPerFlexvol	每个FlexVol 的最大 qtree 数量必须在 [50, 300] 范围内，默认值为 200。 对于 **`ontap-nas-economy`**驱动程序，此选项允许自定义每个**FlexVol**的最大 qtree 数量。	300
sanType	*支持 `ontap-san`仅限司机。*用于选择 `iscsi`对于 iSCSI， `nvme`适用于 NVMe/TCP 或 `fcp`用于光纤通道 (FC) 上的 SCSI。	`iscsi`如果为空
limitVolumePoolSize	*支持 `ontap-san-economy`和 `ontap-san-economy`仅限司机。*限制ONTAP ontap-nas-economy 和 ontap-SAN-economy 驱动程序中的FlexVol大小。	300g

系统提供了默认选项，避免在创建的每个卷上都进行指定：

选项	描述	示例
spaceReserve	空间预订模式； none （精简配置）或 volume（厚的）	none
snapshotPolicy	要使用的快照策略，默认值为 none	none
snapshotReserve	快照预留百分比，默认值为空字符串，表示接受ONTAP默认值。	10
splitOnClone	创建时将克隆体与其父级分离，默认为 false	false
encryption	在新卷上启用NetApp卷加密 (NVE)；默认启用 false。要使用此选项，必须在集群上获得 NVE 许可并启用 NVE。 如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。 更多信息，请参阅："Trident如何与 NVE 和 NAE 协同工作" 。	true
unixPermissions	NAS 选项，用于已配置的 NFS 卷，默认值为 777	777
snapshotDir	NAS选项用于访问 `.snapshot`目录。	NFSv4 为“true”，NFSv3 为“false”。
exportPolicy	NFS导出策略要使用的NAS选项，默认值为 default	default
securityStyle	NAS 选项，用于访问已配置的 NFS 卷。 NFS 支持 mixed`和 `unix`安全措施。默认值为 `unix。	unix
fileSystemType	SAN选项用于选择文件系统类型，默认为 ext4	xfs
tieringPolicy	要使用的分层策略，默认值为 none。	none

缩放选项

这 `ontap-nas`和 `ontap-san`驱动程序为每个 Docker 卷创建一个ONTAP FlexVol 。 ONTAP每个集群节点最多支持 1000 个 FlexVol，集群最多可支持 12,000 个FlexVol卷。如果您的 Docker 卷需求符合此限制，则 `ontap-nas`由于 FlexVols 提供了 Docker 卷粒度快照和克隆等附加功能，因此驱动程序是首选的 NAS 解决方案。

如果您需要的 Docker 卷数量超过了FlexVol 的限制，请选择 `ontap-nas-economy`或者 `ontap-san-economy`司机。

这 `ontap-nas-economy`驱动程序在自动管理的FlexVol卷池中创建作为ONTAP Qtree 的 Docker 卷。 Qtree 提供了更大的可扩展性，每个集群节点最多可达 100,000 个节点，每个集群最多可达 2,400,000 个节点，但代价是牺牲了一些功能。这 `ontap-nas-economy`驱动程序不支持 Docker 卷粒度快照或克隆。

> 这 `ontap-nas-economy`Docker Swarm 目前不支持该驱动程序，因为 Docker Swarm 无法协调跨多个节点的卷创建。

这 `ontap-san-economy`驱动程序在自动管理的FlexVol卷的共享池中创建 Docker 卷作为ONTAP LUN。这样一来，每个FlexVol就不局限于一个 LUN，并且为 SAN 工作负载提供了更好的可扩展性。根据存储阵列的不同，ONTAP每个集群最多支持 16384 个 LUN。由于这些卷底层是 LUN，因此该驱动程序支持 Docker 卷粒度快照和克隆。

选择 `ontap-nas-flexgroup`驱动程序可提高单个卷的并行处理能力，该卷可以增长到 PB 级，包含数十亿个文件。 FlexGroups 的一些理想用例包括 AI/ML/DL、大数据和分析、软件构建、流媒体、文件存储库等等。 Trident在配置FlexGroup卷时会使用分配给 SVM 的所有聚合。 Trident中的FlexGroup支持还需考虑以下几点：

- 需要ONTAP版本 9.2 或更高版本。
- 截至撰写本文时，FlexGroups 仅支持 NFS v3。
- 建议为 SVM 启用 64 位 NFSv3 标识符。
- 建议的最小FlexGroup成员/卷大小为 100 GiB。
- FlexGroup卷不支持克隆。

有关 FlexGroup 以及适合 FlexGroup 的工作负载的信息，请参阅以下内容： "NetApp FlexGroup卷最佳实践和实施指南" 。

为了在同一环境中获得高级功能和大规模部署，您可以运行多个 Docker Volume Plugin 实例，其中一个使用 `ontap-nas`以及另一个使用 `ontap-nas-economy`。

为**Trident**定制**ONTAP**角色

您可以创建一个具有最低权限的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将使用您创建的ONTAP集群角色来执行操作。

请参阅"Trident自定义角色生成器"有关创建Trident自定义角色的更多信息。

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"
security login create -username <user_name\> -application http -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\>
```

使用系统管理器

在ONTAP系统管理器中执行以下步骤：

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“集群 > 设置”。

      （或者）要在 SVM 级别创建自定义角色，请选择“存储”>“存储虚拟机”> `required SVM` > 设置 > 用户和角色*。

   b. 选择“用户和角色”旁边的箭头图标（→）。
   c. 在“角色”下选择“+添加”。
   d. 定义角色规则，然后点击“保存”。
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. 在“用户”下方选择“添加”图标 **+**。
   b. 选择所需的用户名，然后在“角色”下拉菜单中选择角色。
   c. 单击“保存”。

更多信息请参阅以下页面：

- "用于管理ONTAP的自定义角色"或者"定义自定义角色"
- "与角色和用户协作"

ONTAP配置文件示例

<code>ontap-nas</code>驱动程序的NFS示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

<code>ontap-nas-flexgroup</code>驱动程序的NFS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",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

<code>ontap-nas-economy</code> 驱动程序的 NFS 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",
  "aggregate": "aggr1"
}
```

<code>ontap-san</code> 驱动程序的 iSCSI 示例

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

<code>ontap-san-economy</code> 驱动程序的 NFS 示例

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

<code>ontap-san</code> 驱动程序的 NVMe/TCP 示例

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

<code>ontap-san</code>驱动程序的SCSI over FC示例

```
{
  "version": 1,
  "backendName": "ontap-san-backend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "sanType": "fcp",
  "svm": "trident_svm",
  "username": "vsadmin",
  "password": "password",
  "useREST": true
}
```

Element软件配置

除了全局配置值之外，在使用 Element 软件（NetApp HCI/ SolidFire）时，还可以使用这些选项。

选项	描述	示例
`Endpoint`	<a href="https://<登录名>:<密码>@<mvip>/json-rpc/<元素版本&gt" class="bare">https://<登录名>:<密码>@<mvip>/json-rpc/<元素版本&gt;	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IP 地址和端口	10.0.0.7:3260

选项	描述	示例
`TenantName`	要使用的 SolidFireF 租户（如果未找到则创建）	`docker`
`InitiatorIFace`	将 iSCSI 流量限制在非默认接口时，请指定接口。	`default`
`Types`	QoS规范	参见下面的示例
`LegacyNamePrefix`	升级版Trident安装的前缀。如果您使用的是 1.3.2 之前的Trident版本，并且使用现有卷执行升级，则需要设置此值才能访问通过卷名称方法映射的旧卷。	`netappdvp-`

这 `solidfire-san`驱动程序不支持 Docker Swarm。

示例元素软件配置文件

```
{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
  "SVIP": "10.0.0.7:3260",
  "TenantName": "docker",
  "InitiatorIFace": "default",
  "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
      }
    }
  ]
}
```

已知问题和限制

查找有关将Trident与 Docker 结合使用时已知问题和限制的信息。

将**Trident Docker Volume Plugin** 从旧版本升级到 **20.10** 及更高版本会导致升级失败，并出现“没有这样的文件或目录”错误。

临时解决策

1. 禁用插件。

```
docker plugin disable -f netapp:latest
```

2. 移除插件。

```
docker plugin rm -f netapp:latest
```

3. 重新安装插件，并提供额外信息 `config`范围。

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

卷名长度至少为 **2** 个字符。

这是 Docker 客户端的限制。客户端会将单个字符名称解释为 Windows 路径。 "参见错误 25773" 。

Docker Swarm 的某些行为导致**Trident**无法支持它与所有存储和驱动程序的组合。

- Docker Swarm 目前使用卷名称而不是卷 ID 作为其唯一的卷标识符。
- 卷请求会同时发送到 Swarm 集群中的每个节点。
- 卷插件（包括Trident）必须在 Swarm 集群中的每个节点上独立运行。由于ONTAP 的工作方式以及 `ontap-nas`和 `ontap-san`驾驶员之所以能发挥作用，是因为他们是唯一能够在这些限制条件下操作的人。

其余驱动程序会受到诸如竞争条件之类的问题的影响，这可能会导致单个请求创建大量卷，而没有明确的“赢家”；例如，Element 具有允许卷具有相同名称但 ID 不同的功能。

NetApp已向 Docker 团队提供了反馈，但目前没有任何迹象表明未来会采取什么措施。

如果正在配置**FlexGroup** ，而第二个**FlexGroup**与正在配置的**FlexGroup**有一个或多个共同的聚合，则**ONTAP**不会配置第二个**FlexGroup** 。

最佳实践和建议

部署

部署Trident时，请遵循此处列出的建议。

部署到专用命名空间

"命名空间"造成不同应用程序之间的管理隔离，并阻碍资源共享。例如，一个命名空间中的 PVC 不能在另一个命名空间中使用。 Trident为 Kubernetes 集群中的所有命名空间提供 PV 资源，因此利用了具有提升权限的服务帐户。

此外，访问Trident pod 可能会使用户能够访问存储系统凭据和其他敏感信息。必须确保应用程序用户和管理应用程序无法访问Trident对象定义或 pod 本身。

使用配额和范围限制来控制存储消耗

Kubernetes 具有两个特性，这两个特性结合起来，为限制应用程序的资源消耗提供了一种强大的机制。这 "存储配额机制"管理员可以按命名空间实施全局和存储类特定的容量和对象数量消耗限制。此外，使用 "射程限制"确保 PVC 请求在转发给配置器之前，其值既在最小值也在最大值范围内。

这些值是按命名空间定义的，这意味着每个命名空间都应该定义与其资源需求相符的值。请点击此处查看相关信息 "如何利用配额"。

存储配置

NetApp产品组合中的每个存储平台都具有独特的功能，无论应用程序是否采用容器化，都能从中受益。

平台概览

Trident可与ONTAP和 Element 配合使用。没有哪个平台比其他平台更适合所有应用和场景，但是，在选择平台时，应该考虑应用的需求以及管理设备的团队的需求。

您应该遵循您所使用协议的主机操作系统的最佳实践。 （可选）您可以考虑将应用程序最佳实践（如果可用）与后端、存储类别和 PVC 设置相结合，以优化特定应用程序的存储。

ONTAP和Cloud Volumes ONTAP最佳实践

了解配置ONTAP和Cloud Volumes ONTAP for Trident 的最佳实践。

以下建议是为容器化工作负载配置ONTAP的指导原则，这些工作负载使用由Trident动态配置的卷。每项都应根据您的环境进行考虑和评估，以确定其适用性。

使用专用于**Trident** 的**SVM**。

存储虚拟机 (SVM) 为ONTAP系统上的租户提供隔离和管理分离。将 SVM 专用于应用程序可以实现权限委派，并可以应用限制资源消耗的最佳实践。

SVM的管理有多种选择：

- 在后端配置中提供集群管理接口，以及相应的凭据，并指定 SVM 名称。
- 使用ONTAP系统管理器或 CLI 为 SVM 创建专用管理界面。
- 与 NFS 数据接口共享管理角色。

在每种情况下，接口都应该在 DNS 中，并且在配置Trident时应该使用 DNS 名称。这有助于实现某些灾难恢复场景，例如无需网络身份保留的 SVM-DR。

对于 SVM 而言，采用专用管理 LIF 还是共享管理 LIF 没有偏好，但是，您应该确保您的网络安全策略与您选择的方法保持一致。无论如何，管理 LIF 应该可以通过 DNS 访问，以实现最大的灵活性。 "SVM-DR"可与Trident配合使用。

限制最大体积数

ONTAP存储系统有最大卷数限制，该限制会根据软件版本和硬件平台而有所不同。请参阅 "NetApp Hardware Universe"请根据您的具体平台和ONTAP版本确定确切的限制。当卷计数耗尽时，不仅Trident的配置操作会失败，而且所有存储请求的配置操作都会失败。

三叉戟的 `ontap-nas`和 `ontap-san`驱动程序为每个创建的 Kubernetes 持久卷 (PV) 提供 FlexVolume。这 `ontap-nas-economy`驱动程序大约每 200 个 PV 创建一个 FlexVolume（可在 50 到 300 之间配置）。这 `ontap-san-economy`驱动程序大约每 100 个 PV 创建一个 FlexVolume（可在 50 到 200 之间配置）。为了防止Trident消耗存储系统上所有可用的卷，您应该对 SVM 设置限制。您可以从命令行执行此操作：

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

价值 `max-volumes`价格会根据您所在环境的几个具体因素而有所不同：

- ONTAP集群中现有卷的数量
- 您预计在Trident之外为其他应用程序配置的卷数
- Kubernetes 应用程序预计使用的持久卷数量

这 `max-volumes`该值是ONTAP集群中所有节点上配置的总卷数，而不是单个ONTAP节点上的卷数。因此，您可能会遇到某些情况，即ONTAP集群节点的Trident配置卷可能比其他节点多得多或少得多。

例如，一个双节点ONTAP集群最多可以托管 2000 个FlexVol卷。将最大音量计数设置为 1250 似乎非常合理。然而，如果仅仅 "聚合"如果从一个节点分配给 SVM，或者从一个节点分配的聚合无法进行配置（例如，由于容量不足），则另一个节点将成为所有Trident配置卷的目标。这意味着该节点的容量限制可能在达到上限之前就已经达到。 `max-volumes`达到值后，将影响Trident和使用该节点的其他卷操作。 您可以通过确保集群中每个节点的聚合数据以相等的数量分配给**Trident**使用的**SVM**来避免这种情况。

克隆卷

NetApp Trident在使用时支持克隆卷 `ontap-nas`， `ontap-san` ， `solidfire-san` ， 和 `gcp-cvs`存储驱动程序。使用时 `ontap-nas-flexgroup`或者 `ontap-nas-economy`驱动程序不支持克隆。从现有卷创建新卷将创建一个新的快照。

避免克隆与不同存储类关联的PVC。在同一个 StorageClass 内执行克隆操作，以确保兼容性并防止出现意外行为。

限制**Trident**创建的最大体积。

要配置Trident可创建的卷的最大大小，请使用以下方法： `limitVolumeSize`参数 `backend.json`定义。

除了控制存储阵列的卷大小之外，您还应该利用 Kubernetes 功能。

限制**Trident**创建的**FlexVols**的最大尺寸。

要配置用作 ontap-san-economy 和 ontap-nas-economy 驱动程序池的 FlexVols 的最大大小，请使用以下方法： `limitVolumePoolSize`参数 `backend.json`定义。

配置**Trident**使用双向 **CHAP**

您可以在后端定义中指定 CHAP 发起方和目标用户名和密码，并让Trident在 SVM 上启用 CHAP。使用 `useCHAP`在后端配置中设置参数， Trident使用 CHAP 对ONTAP后端的 iSCSI 连接进行身份验证。

创建并使用 **SVM QoS** 策略

利用应用于 SVM 的ONTAP QoS 策略，可以限制Trident已配置卷可消耗的 IOPS 数量。这有助于 "阻止欺凌行为"或者失控的容器影响Trident SVM 之外的工作负载。

只需几个步骤即可为 SVM 创建 QoS 策略。请参阅您所用ONTAP版本的文档以获取最准确的信息。下面的示例创建了一个 QoS 策略，将 SVM 可用的总 IOPS 限制为 5000。

```
# create the policy group for the SVM
qos policy-group create -policy-group <policy_name> -vserver <svm_name>
-max-throughput 5000iops

# assign the policy group to the SVM, note this will not work
# if volumes or files in the SVM have existing QoS policies
vserver modify -vserver <svm_name> -qos-policy-group <policy_name>
```

此外，如果您的ONTAP版本支持，您可以考虑使用 QoS 最低要求来保证容器化工作负载的吞吐量。自适应QoS与SVM层策略不兼容。

分配给容器化工作负载的 IOPS 数量取决于许多方面。其中包括：

- 使用存储阵列的其他工作负载。如果还有其他与 Kubernetes 部署无关的工作负载正在使用存储资源，则应注意确保这些工作负载不会意外受到不利影响。
- 预期工作负载将在容器中运行。如果对 IOPS 要求高的工作负载将在容器中运行，则低 QoS 策略会导致糟糕的用户体验。

需要注意的是，在 SVM 级别分配的 QoS 策略会导致分配给 SVM 的所有卷共享同一个 IOPS 池。如果一个或少数几个容器化应用程序对 IOPS 有很高的要求，它可能会欺负其他容器化工作负载。如果情况属实，您可能需要考虑使用外部自动化工具来分配按卷划分的 QoS 策略。

只有当您的ONTAP版本低于 9.8 时，才应将 QoS 策略组分配给 SVM。

为**Trident**创建 **QoS** 策略组

服务质量 (QoS) 保证关键工作负载的性能不会因竞争工作负载而降低。 ONTAP QoS 策略组为卷提供 QoS 选项，并允许用户为一个或多个工作负载定义吞吐量上限。有关 QoS 的更多信息，请参阅 "通过服务质量 (QoS) 保证吞吐量。"。您可以在后端或存储池中指定 QoS 策略组，这些策略组将应用于在该池或后端中创建的每个卷。

ONTAP有两种类型的 QoS 策略组：传统型和自适应型。传统策略组在 IOPS 中提供固定的最大（或最小，在后续版本中）吞吐量。自适应 QoS 可根据工作负载大小自动调整吞吐量，随着工作负载大小的变化，保持 IOPS 与 TB|GB 的比率不变。在大型部署中管理成百上千个工作负载时，这提供了显著的优势。

创建QoS策略组时，请考虑以下事项：

- 你应该设置 `qosPolicy`关键在于 `defaults`后端配置块。请参见以下后端配置示例：

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 0.0.0.0
dataLIF: 0.0.0.0
svm: svm0
username: user
password: pass
defaults:
  qosPolicy: standard-pg
storage:
  - labels:
      performance: extreme
    defaults:
      adaptiveQosPolicy: extremely-adaptive-pg
  - labels:
      performance: premium
    defaults:
      qosPolicy: premium-pg
```

- 您应该按卷应用策略组，以便每个卷都能获得策略组指定的全部吞吐量。不支持共享策略组。

有关 QoS 策略组的更多信息，请参阅 "ONTAP命令参考"。

限制对 **Kubernetes** 集群成员的存储资源访问

限制对Trident创建的 NFS 卷、iSCSI LUN 和 FC LUN 的访问是 Kubernetes 部署安全态势的关键组成部分。这样做可以防止不属于 Kubernetes 集群的主机访问卷并可能意外修改数据。

重要的是要理解命名空间是 Kubernetes 中资源的逻辑边界。假设同一命名空间内的资源可以共享，但是，重要的是，没有跨命名空间的功能。这意味着，即使 PV 是全局对象，当绑定到 PVC 时，也只能由同一命名空间中的 pod 访问。 务必确保在适当情况下使用命名空间来实现代码分离。

对于大多数组织而言，在 Kubernetes 环境中数据安全的主要担忧是容器中的进程可以访问挂载到主机上的存储，但该存储并非为容器所预期的。 "命名空间"旨在防止此类妥协。但是，特权容器是一个例外。

特权容器是指拥有比普通容器高得多的主机级权限的容器。这些功能默认情况下不会被禁用，因此请确保使用以下命令禁用该功能： "Pod 安全策略" 。

对于需要从 Kubernetes 和外部主机访问的卷，存储应以传统方式进行管理，PV 由管理员引入，而不是由Trident管理。这样可以确保只有当 Kubernetes 和外部主机都已断开连接并且不再使用该卷时，才会销毁存储卷。此外，还可以应用自定义导出策略，从而允许从 Kubernetes 集群节点和 Kubernetes 集群外部的目标服务器进行访问。

对于具有专用基础架构节点（例如 OpenShift）或其他无法调度用户应用程序的节点的部署，应使用单独的导出策略来进一步限制对存储资源的访问。这包括为部署到这些基础设施节点的服务（例如，OpenShift Metrics 和 Logging 服务）以及部署到非基础设施节点的标准应用程序创建导出策略。

使用专门的出口政策

您应该确保每个后端都存在导出策略，该策略仅允许访问 Kubernetes 集群中的节点。Trident可以自动创建和管理出口政策。这样， Trident将其提供的卷的访问限制到 Kubernetes 集群中的节点，并简化了节点的添加/删除。

或者，您也可以手动创建导出策略，并向其中填充一条或多条导出规则，以处理每个节点访问请求：

- 使用 `vserver export-policy create`ONTAP CLI 命令创建导出策略。
- 使用以下方式向出口策略添加规则： `vserver export-policy rule create` ONTAP CLI 命令。

运行这些命令可以限制哪些 Kubernetes 节点可以访问数据。

禁用 **`showmount`**对于应用**SVM**

这 `showmount`此功能使 NFS 客户端能够向 SVM 查询可用 NFS 导出列表。部署到 Kubernetes 集群的 pod 可以发出 `showmount -e`对目标执行命令，并接收可用挂载点列表，包括它无法访问的挂载点。虽然这本身并不构成安全隐患，但它确实提供了不必要的信息，可能会帮助未经授权的用户连接到 NFS 导出。

您应该禁用 `showmount`通过使用 SVM 级ONTAP CLI 命令：

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire最佳实践

了解配置Trident的SolidFire存储的最佳实践。

创建 **Solidfire** 帐户

每个SolidFire帐户代表一个唯一的卷所有者，并接收自己的一组质询握手身份验证协议 (CHAP) 凭证。您可以通过使用帐户名称和相应的 CHAP 凭据或通过卷访问组来访问分配给帐户的卷。一个账户最多可以分配两千卷，但一卷只能属于一个账户。

创建 **QoS** 策略

如果您想要创建并保存可应用于多个卷的标准化服务质量设置，请使用SolidFire服务质量 (QoS) 策略。

您可以按卷设置 QoS 参数。通过设置定义 QoS 的三个可配置参数，可以保证每个卷的性能：最小 IOPS、最大 IOPS 和突发 IOPS。

以下是 4Kb 块大小的可能最小、最大和突发 IOPS 值。

IOPS 参数	定义	最小值	默认值	最大值（**4Kb**）
最小 IOPS	保证一定容量的性能水平。	50	50	15000
最大 IOPS	性能不会超过此限制。	50	15000	200,000
突发 IOPS	短时突发场景下允许的最大IOPS。	50	15000	200,000

尽管最大 IOPS 和突发 IOPS 可以设置为高达 200,000，但卷的实际最大性能受集群使用情况和每个节点性能的限制。

数据块大小和带宽对 IOPS 数量有直接影响。随着数据块大小的增加，系统会将带宽增加到足以处理更大数据块大小的水平。随着带宽的增加，系统能够达到的IOPS数量会减少。请参阅 "SolidFire服务质量"有关服务质量和性能的更多信息。

SolidFire身份验证

Element 支持两种身份验证方法：CHAP 和卷访问组 (VAG)。 CHAP 使用 CHAP 协议对主机进行后端身份验证。卷访问组控制对其所配置卷的访问。 NetApp建议使用 CHAP 进行身份验证，因为它更简单且没有扩展限制。

带有增强型 CSI 配置器的Trident支持使用 CHAP 身份验证。 VAG 只能在传统的非 CSI 模式下使用。

CHAP 身份验证（验证发起者是否为预期的卷用户）仅在基于帐户的访问控制中受支持。如果您使用 CHAP 进行身份验证，则有两种选择：单向 CHAP 和双向 CHAP。单向 CHAP 通过使用SolidFire帐户名和发起方密钥来验证卷访问。双向 CHAP 选项提供了最安全的卷身份验证方式，因为卷通过帐户名和发起方密钥验证主机身份，然后主机通过帐户名和目标密钥验证卷身份。

但是，如果无法启用 CHAP 并且需要 VAG，则创建访问组并将主机启动器和卷添加到访问组。添加到访问组的每个 IQN 都可以使用或不使用 CHAP 身份验证来访问组中的每个卷。如果 iSCSI 发起程序配置为使用 CHAP 身份验证，则使用基于帐户的访问控制。如果 iSCSI 发起程序未配置为使用 CHAP 身份验证，则使用卷访问组访问控制。

哪里可以找到更多信息？

下面列出了一些最佳实践文档。搜索 "NetApp库"适用于最新版本。

ONTAP

- "NFS 最佳实践和实施指南"
- "SAN 管理"（适用于 iSCSI）
- "RHEL 的 iSCSI Express 配置"

Element软件

- "配置适用于 Linux 的SolidFire"

NetApp HCI

- "NetApp HCI部署先决条件"
- "访问NetApp部署引擎"

应用最佳实践信息

- "ONTAP上 MySQL 的最佳实践"
- "SolidFire上 MySQL 的最佳实践"
- "NetApp SolidFire和 Cassandra"
- "Oracle 在SolidFire的最佳实践"
- "SolidFire上的PostgreSQL最佳实践"

并非所有应用程序都有具体的指导原则，因此与您的NetApp团队合作并使用以下方法至关重要： "NetApp库"查找最新文档。

整合Trident

要集成Trident，需要集成以下设计和架构元素：驱动程序选择和部署、存储类设计、虚拟池设计、持久卷声明 (PVC) 对存储配置的影响、卷操作以及使用Trident部署 OpenShift 服务。

驾驶员选择和部署

为您的存储系统选择并部署后端驱动程序。

ONTAP后端驱动程序

ONTAP后端驱动程序的区别在于所使用的协议以及在存储系统上配置卷的方式。因此，在决定部署哪位驾驶员时要仔细考虑。

从更高的层面来看，如果您的应用程序具有需要共享存储的组件（多个 pod 访问同一个 PVC），则基于 NAS 的驱动程序将是默认选择，而基于块的 iSCSI 驱动程序则满足非共享存储的需求。根据应用程序的要求以及存储和基础设施团队的接受程度来选择协议。一般来说，对于大多数应用来说，它们之间几乎没有区别，因此通常取决于是否需要共享存储（多个 pod 需要同时访问）。

可用的ONTAP后端驱动程序有：

- `ontap-nas`每个已配置的 PV 都是一个完整的ONTAP FlexVolume。
- `ontap-nas-economy`每个已配置的 PV 都是一个 qtree，每个 FlexVolume 的 qtree 数量可配置（默认值为 200）。
- `ontap-nas-flexgroup`每个 PV 都配置为一个完整的ONTAP FlexGroup，并且分配给 SVM 的所有聚合都将被使用。
- `ontap-san`每个已配置的 PV 都是其自身 FlexVolume 中的一个 LUN。
- `ontap-san-economy`每个已配置的 PV 都是一个 LUN，每个 FlexVolume 的 LUN 数量可配置（默认值为 100）。

在三个 NAS 驱动程序之间进行选择会对应用程序可用的功能产生一些影响。

请注意，在下表中，并非所有功能都通过Trident公开。如果需要某些功能，则必须由存储管理员在配置完成后应用这些功能。上标脚注区分了每个功能和驱动程序的具体功能。

ONTAP NAS 驱动程序	**Snapshot**	克隆	动态出口策略	多附件	**QoS**	调整大小	复制
`ontap-nas`	是	是	是的脚注：5[]	是	是的脚注：1[]	是	是的脚注：1[]
`ontap-nas-economy`	无脚注：3[]	无脚注：3[]	是的脚注：5[]	是	无脚注：3[]	是	无脚注：3[]
`ontap-nas-flexgroup`	是的脚注：1[]	否	是的脚注：5[]	是	是的脚注：1[]	是	是的脚注：1[]

Trident为ONTAP提供 2 款 SAN 驱动程序，其功能如下所示。

ONTAP SAN 驱动程序	**Snapshot**	克隆	多附件	双向 **CHAP**	**QoS**	调整大小	复制
`ontap-san`	是	是	是的脚注：4[]	是	是的脚注：1[]	是	是的脚注：1[]
`ontap-san-economy`	是	是	是的脚注：4[]	是	无脚注：3[]	是	无脚注：3[]

> 以上表格的脚注：是脚注1[]：非Trident管理；是脚注2[]：由Trident管理，但不支持PV粒度；否脚注3[]：非Trident管理且不支持PV粒度；是脚注4[]：支持原始块卷；是脚注5[]： Trident支持。

非 PV 粒度的功能将应用于整个 FlexVolume，所有 PV（即共享 FlexVol 中的 qtree 或 LUN）将共享一个共同的计划。

如上表所示，大部分功能都体现在以下方面： `ontap-nas`和 `ontap-nas-economy`是一样的。然而，因为 `ontap-nas-economy`驱动程序限制了以 `PV` 粒度控制计划的能力，这可能会特别影响您的灾难恢复和备份计划。对于希望在ONTAP存储上利用 `PVC` 克隆功能的开发团队而言，只有在使用以下配置时才有可能： `ontap-nas` ， `ontap-san`或者 `ontap-san-economy`司机。

这 `solidfire-san`驱动程序还能够克隆PVC。

Cloud Volumes ONTAP后端驱动程序

Cloud Volumes ONTAP提供数据控制以及企业级存储功能，适用于各种用例，包括文件共享和块级存储，支持 NAS 和 SAN 协议（NFS、SMB / CIFS 和 iSCSI）。 Cloud Volume ONTAP的兼容驱动程序有： `ontap-nas` ， `ontap-nas-economy` ， `ontap-san`和 `ontap-san-economy`。这些适用于 Azure 版 Cloud Volume ONTAP和 GCP 版 Cloud Volume ONTAP 。

Amazon FSx for ONTAP后端驱动程序

Amazon FSx for NetApp ONTAP让您能够利用您熟悉的NetApp功能、性能和管理能力，同时享受在 AWS 上存储数据的简单性、敏捷性、安全性和可扩展性。 FSx for ONTAP支持许多ONTAP文件系统功能和管理 API。 Cloud Volume ONTAP的兼容驱动程序有： `ontap-nas` ， `ontap-nas-economy` ， `ontap-nas-flexgroup` ， `ontap-san`和 `ontap-san-economy`。

NetApp HCI/ SolidFire后端驱动程序

这 `solidfire-san`该驱动程序与NetApp HCI/ SolidFire平台配合使用，可帮助管理员根据 QoS 限制为Trident配置 Element 后端。如果您希望设计后端以设置Trident所配置卷的特定 QoS 限制，请使用以下方法： `type`后端文件中的参数。管理员还可以使用以下方法限制可在存储上创建的卷大小： `limitVolumeSize`范围。目前 ，Element 存储功能（例如卷调整大小和卷复制）尚不支持通过以下方式进行存储： `solidfire-san`司机。这些操作应该通过 Element Software 的 Web 用户界面手动完成。

SolidFire驱动程序	Snapshot	克隆	多附件	CHAP	QoS	调整大小	复制
`solidfire-san`	是	是	是的脚注：2[]	是	是	是	是的脚注：1[]

> 脚注：是脚注1： Trident不管理 是脚注2：支持原始块卷

Azure NetApp Files后端驱动程序

Trident使用 `azure-netapp-files`司机管理"Azure NetApp Files"服务。

有关此驱动程序及其配置方法的更多信息，请参见此处。"用于Azure NetApp Files的Trident后端配置" 。

Azure NetApp Files驱动程序	Snapshot	克隆	多附件	QoS	展开	复制
`azure-netapp-files`	是	是	是	是	是	是的脚注：1[]

> 脚注：是脚注：1[]：并非由Trident管理

Google Cloud 后端驱动程序上的Cloud Volumes Service

Trident使用 `gcp-cvs`用于连接 Google Cloud 上的Cloud Volumes Service的驱动程序。

这 `gcp-cvs`驱动程序使用虚拟池来抽象后端，并允许Trident确定卷放置位置。管理员在以下位置定义虚拟池： `backend.json`文件。存储类使用选择器按标签识别虚拟池。

- 如果后端定义了虚拟池， Trident将尝试在 Google Cloud 存储池中创建卷，而这些虚拟池仅限于这些存储池。

- 如果后端未定义虚拟池， Trident将从该区域的可用存储池中选择一个 Google Cloud 存储池。

要在Trident上配置 Google Cloud 后端，您必须指定 `projectNumber`， `apiRegion` ， 和 `apiKey`在后端文件中。您可以在 Google Cloud 控制台中找到项目编号。 API 密钥取自您在 Google Cloud 上为Cloud Volumes Service设置 API 访问权限时创建的服务帐户私钥文件。

有关 Google Cloud 上的Cloud Volumes Service服务类型和服务级别的详细信息，请参阅："了解Trident对 GCP 版 CVS 的支持" 。

适用于 **Google Cloud Drive** 的**Cloud Volumes Service**	**Snapshot**	克隆	多附件	**QoS**	展开	复制
`gcp-cvs`	是	是	是	是	是	仅适用于 CVS-Performance 服务类型。

复制说明

- 复制功能并非由Trident管理。
- 克隆卷将创建在与源卷相同的存储池中。

存储类设计

要创建 Kubernetes 存储类对象，需要对各个存储类进行配置和应用。本节讨论如何为您的应用程序设计存储类。

具体后端利用

在特定的存储类对象中可以使用过滤来确定要与该特定存储类一起使用的存储池或存储池集。存储类中可以设置三组过滤器： `storagePools` ， `additionalStoragePools`和/或 `excludeStoragePools`。

这 `storagePools`该参数有助于将存储限制在与任何指定属性匹配的存储池集合中。这 `additionalStoragePools`该参数用于扩展Trident用于配置的池集，以及由属性选择的池集。 `storagePools`参数。您可以单独使用其中一个参数，也可以同时使用这两个参数，以确保选择合适的存储池集。

这 `excludeStoragePools`该参数用于专门排除符合属性要求的已列出泳池集合。

模拟 **QoS** 策略

如果您希望设计存储类来模拟服务质量策略，请创建一个具有以下功能的存储类： `media`属性为 `hdd`或者 `ssd`。基于 `media`根据存储类中提到的属性， Trident将选择合适的后端来提供服务。 `hdd`或者 `ssd`将聚合与媒体属性匹配，然后将卷的配置定向到特定的聚合上。因此，我们可以创建一个 PREMIUM 存储类，它将具有 `media`属性集为 `ssd`这可以归类为高级 QoS 策略。我们可以创建另一个存储类 STANDARD，其媒体属性设置为“hdd”，这可以归类为 STANDARD QoS 策略。我们还可以使用存储类中的“IOPS”属性将配置重定向到 Element 设备，该设备可以定义为 QoS 策略。

根据特定功能使用后端

存储类可以设计为在特定的后端上指导卷配置，其中启用了精简配置和厚配置、快照、克隆和加密等功能。要指

定要使用的存储，请创建存储类，指定启用所需功能的相应后端。

虚拟池

所有Trident后端均可使用虚拟池。你可以使用Trident提供的任何驱动程序，为任何后端定义虚拟池。

虚拟池允许管理员在后端之上创建一个抽象层，可以通过存储类引用该抽象层，从而在后端上更灵活、更高效地放置卷。同一服务类别可以定义不同的后端。此外，可以在同一个后端创建多个具有不同特性的存储池。当使用具有特定标签的选择器配置存储类时， Trident会选择与所有选择器标签匹配的后端来放置卷。如果存储类别选择器标签与多个存储池匹配， Trident将从中选择其中一个来配置卷。

虚拟泳池设计

创建后端时，通常可以指定一组参数。管理员无法创建具有相同存储凭据和不同参数集的另一个后端。随着虚拟池的引入，这个问题得到了缓解。虚拟池是在后端和 Kubernetes 存储类之间引入的级别抽象，以便管理员可以定义参数以及可以通过 Kubernetes 存储类作为选择器引用的标签，以与后端无关的方式。可以使用Trident为所有受支持的NetApp后端定义虚拟池。该列表包括SolidFire/ NetApp HCI、 ONTAP、GCP 上的Cloud Volumes Service以及Azure NetApp Files。

> 定义虚拟池时，建议不要尝试在后端定义中重新排列现有虚拟池的顺序。此外，建议不要编辑/修改现有虚拟池的属性，而是定义一个新的虚拟池。

模拟不同的服务级别/**QoS**

可以设计虚拟池来模拟服务类。使用Azure NetApp Files云卷服务的虚拟池实现，让我们来研究如何设置不同的服务类。配置Azure NetApp Files后端，使用多个标签来表示不同的性能级别。放 `servicelevel`将各个方面调整到相应的性能水平，并在每个标签下添加其他所需的方面。现在创建不同的 Kubernetes 存储类，将它们映射到不同的虚拟池。使用 `parameters.selector`字段中，每个 StorageClass 都会指定哪些虚拟池可用于托管卷。

指定一组特定的方面

可以从单个存储后端设计多个具有特定功能的虚拟池。为此，请在后端配置多个标签，并在每个标签下设置所需的方面。现在使用以下方式创建不同的 Kubernetes 存储类： `parameters.selector`该字段将映射到不同的虚拟池。后端配置的卷将具有所选虚拟池中定义的方面。

影响存储供应的**PVC**特性

在创建 PVC 时，除请求的存储类别之外的某些参数可能会影响Trident 的配置决策过程。

访问模式

通过 PVC 请求存储时，必填字段之一是访问模式。所需的模式可能会影响用于托管存储请求的后端选择。

Trident将尝试根据以下矩阵将所使用的存储协议与指定的访问方法进行匹配。这与底层存储平台无关。

	读写一次	只读多	读写多
iSCSI	是	是	是的（原始块）
NFS	是	是	是

如果向未配置 NFS 后端的Trident部署提交 ReadWriteMany PVC 请求，则不会配置任何卷。因此，请求者应该

使用适合其应用的访问模式。

卷操作

修改持久卷

除两种例外情况外，持久卷是 Kubernetes 中的不可变对象。创建完成后，可以修改回收策略和大小。然而，这并不能阻止在 Kubernetes 之外修改卷的某些方面。为了针对特定应用定制卷，确保容量不会意外耗尽，或者出于任何原因将卷移动到不同的存储控制器，这样做可能是可取的。

目前 Kubernetes 树内配置器不支持对 NFS、iSCSI 或 FC PV 进行卷大小调整操作。 Trident支持扩展 NFS、iSCSI 和 FC 卷。

PV的连接详情创建后无法修改。

创建按需卷快照

Trident支持按需创建卷快照，并支持使用 CSI 框架从快照创建 PVC。快照提供了一种便捷的方法来维护数据在特定时间点的副本，并且在 Kubernetes 中具有独立于源 PV 的生命周期。这些快照可用于克隆PVC。

从快照创建卷

Trident还支持从卷快照创建持久卷。要实现这一点，只需创建一个 PersistentVolumeClaim 并指定 `datasource` 作为创建卷所需的快照。 Trident将通过创建一个包含快照中数据的卷来处理此 PVC。借助此功能，可以跨区域复制数据、创建测试环境、完全替换损坏或已损坏的生产卷，或者检索特定文件和目录并将其传输到另一个附加卷。

在集群中移动卷

存储管理员能够在ONTAP集群中，在聚合和控制器之间移动卷，而不会对存储用户造成任何干扰。只要目标聚合是Trident使用的 SVM 可以访问的目标聚合，此操作就不会影响Trident或 Kubernetes 集群。重要的是，如果聚合体是新添加到 SVM 中的，则需要通过将其重新添加到Trident来刷新后端。这将触发Trident重新清点 SVM ，以便识别新的聚合体。

但是， Trident并不自动支持跨后端移动卷。这包括在同一集群中的 SVM 之间、集群之间，或者到不同的存储平台（即使该存储系统是连接到Trident 的存储系统）。

如果将卷复制到另一个位置，则可以使用卷导入功能将当前卷导入到Trident中。

扩大规模

Trident支持调整 NFS、iSCSI 和 FC PV 的大小。这样用户就可以直接通过 Kubernetes 层调整卷的大小。所有主流NetApp存储平台，包括ONTAP、 SolidFire/ NetApp HCI和Cloud Volumes Service后端，均可进行卷扩展。为了便于日后扩展，请设置 `allowVolumeExpansion`到 `true`在与该卷关联的 StorageClass 中。每当需要调整持久卷的大小时，请编辑以下内容： `spec.resources.requests.storage`在持久卷声明中添加对所需卷大小的注释。 Trident会自动处理存储集群上卷的大小调整。

将现有卷导入 **Kubernetes**

卷导入功能允许将现有存储卷导入 Kubernetes 环境。目前这得到了以下方面的支持： `ontap-nas` ， `ontap-nas-flexgroup` ， `solidfire-san` ， `azure-netapp-files` ， 和 `gcp-cvs`司机。将现有应用程序移植到 Kubernetes 或灾难恢复场景中，此功能非常有用。

使用ONTAP时 `solidfire-san`司机们，请使用该命令 `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`将现有卷导入 Kubernetes 以便由Trident管理。导入卷命令中使用的 PVC YAML 或 JSON 文件指向一个存储类，该存储类将Trident标识为配置程序。使用NetApp HCI/ SolidFire后端时，请确保卷名称是唯一的。如果卷名称重复，请将卷克隆为唯一名称，以便卷导入功能可以区分它们。

如果 `azure-netapp-files`或者 `gcp-cvs`使用了驱动程序，请使用命令 `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`将卷导入 Kubernetes 以便由Trident管理。这样就确保了卷号的唯一性。

执行上述命令后， Trident将在后端找到该卷并读取其大小。它会自动添加（必要时会覆盖）已配置的PVC的容积大小。然后Trident创建新的 PV，Kubernetes 将 PVC 绑定到 PV。

如果部署的集装箱需要特定的进口 PVC，则该集装箱将保持待定状态，直到通过批量进口流程绑定 PVC/PV 对为止。 PVC/PV管对连接好后，如果没有其他问题，容器应该就能升起来。

注册服务

注册表存储的部署和管理已在文档中记录。"netapp.io"在"博客"。

日志服务

与其他 OpenShift 服务一样，日志服务使用 Ansible 进行部署，配置参数由提供给 playbook 的清单文件（又名 hosts）提供。本文将介绍两种安装方法：在 OpenShift 初始安装期间部署日志记录和在 OpenShift 安装完成后部署日志记录。

> 从 Red Hat OpenShift 版本 3.9 开始，官方文档建议不要使用 NFS 作为日志服务，因为存在数据损坏方面的担忧。这是基于红帽公司对其产品的测试结果。 ONTAP NFS 服务器不存在这些问题，并且可以轻松地支持日志部署。最终，日志服务的协议选择取决于您，但要知道，在使用NetApp平台时，这两种协议都能很好地工作，如果您更喜欢 NFS，也没有理由避免使用 NFS。

如果选择将 NFS 与日志服务一起使用，则需要设置 Ansible 变量。 `openshift_enable_unsupported_configurations`到 `true`防止安装程序运行失败。

开始使用

日志服务可以选择性地部署到应用程序以及 OpenShift 集群本身的核心操作中。如果您选择部署操作日志记录，请指定以下变量 `openshift_logging_use_ops`作为 `true`将创建该服务的两个实例。控制操作日志实例的变量中包含“ops”，而控制应用程序日志实例的变量则不包含。

根据部署方法配置 Ansible 变量非常重要，以确保底层服务使用正确的存储。让我们来看看每种部署方法的选项。

> 下表仅包含与日志服务相关的存储配置变量。您还可以找到其他选择"Red Hat OpenShift 日志记录文档"应根据您的部署情况进行审查、配置和使用。

下表中的变量将使 Ansible playbook 使用提供的详细信息为日志服务创建 PV 和 PVC。虽然这种方法不如在 OpenShift 安装后使用组件安装 playbook 灵活，但如果您有现有的卷可用，这也不失为一个选择。

变量	详细信息
`openshift_logging_storage_kind`	设置为 `nfs`让安装程序为日志服务创建 NFS PV。

变量	详细信息
`openshift_logging_storage_host`	NFS主机的主机名或IP地址。这应该设置为虚拟机的dataLIF。
`openshift_logging_storage_nfs_directory`	NFS导出挂载路径。例如，如果体积连接为`/openshift_logging`你会将该路径用于此变量。
`openshift_logging_storage_volume_name`	名称，例如 `pv_ose_logs`，创建 PV。
`openshift_logging_storage_volume_size`	例如，NFS 导出的大小。 `100Gi` 。

如果您的 OpenShift 集群已经运行，因此Trident已经部署和配置，安装程序可以使用动态配置来创建卷。需要配置以下变量。

变量	详细信息
`openshift_logging_es_pvc_dynamic`	设置为 true 以使用动态配置卷。
`openshift_logging_es_pvc_storage_class_name`	PVC 中将使用的存储类别的名称。
`openshift_logging_es_pvc_size`	PVC中要求的体积大小。
`openshift_logging_es_pvc_prefix`	日志记录服务使用的 PVC 前缀。
`openshift_logging_es_ops_pvc_dynamic`	设置为 `true`为运维日志实例使用动态配置的卷。
`openshift_logging_es_ops_pvc_storage_class_name`	运维日志实例的存储类名称。
`openshift_logging_es_ops_pvc_size`	操作实例的卷请求大小。
`openshift_logging_es_ops_pvc_prefix`	操作实例 PVC 的前缀。

部署日志堆栈

如果您将日志记录部署作为 OpenShift 初始安装过程的一部分，那么您只需要遵循标准部署过程即可。 Ansible 将配置和部署所需的服务和 OpenShift 对象，以便在 Ansible 完成后立即提供该服务。

但是，如果在初始安装之后进行部署，则需要使用 Ansible 的组件 playbook。此过程可能因 OpenShift 版本不同而略有差异，因此请务必阅读并遵循相关说明。"Red Hat OpenShift 容器平台 3.11 文档"适用于您的版本。

指标服务

指标服务为管理员提供有关 OpenShift 集群的状态、资源利用率和可用性的宝贵信息。此外，它对于 pod 自动扩展功能也是必要的，许多组织使用指标服务中的数据进行费用分摊和/或收益展示应用程序。

与日志服务以及整个 OpenShift 一样，Ansible 也用于部署指标服务。此外，与日志服务一样，指标服务可以在集群的初始设置期间或集群运行后使用组件安装方法进行部署。以下表格包含了为指标服务配置持久存储时重要的变量。

下表仅包含与指标服务相关的存储配置变量。文档中还有许多其他选项，应根据您的部署情况进行查看、配置和使用。

变量	详细信息
`openshift_metrics_storage_kind`	设置为 `nfs`让安装程序为日志服务创建 NFS PV。
`openshift_metrics_storage_host`	NFS主机的主机名或IP地址。这应该设置为你的 SVM 的 dataLIF。
`openshift_metrics_storage_nfs_directory`	NFS导出挂载路径。例如，如果体积连接为 `/openshift_metrics`你会将该路径用于此变量。
`openshift_metrics_storage_volume_name`	名称，例如 `pv_ose_metrics`，用于创建 PV。
`openshift_metrics_storage_volume_size`	例如，NFS 导出的大小。 `100Gi` 。

如果您的 OpenShift 集群已经运行，因此Trident已经部署和配置，安装程序可以使用动态配置来创建卷。需要配置以下变量。

变量	详细信息
`openshift_metrics_cassandra_pvc_prefix`	用于指标 PVC 的前缀。
`openshift_metrics_cassandra_pvc_size`	请求的卷册数量。
`openshift_metrics_cassandra_storage_type`	用于指标的存储类型，必须将其设置为动态，以便 Ansible 创建具有适当存储类的 PVC。
`openshift_metrics_cassanda_pvc_storage_class_name`	要使用的存储类的名称。

部署指标服务

在 hosts/inventory 文件中定义了相应的 Ansible 变量后，使用 Ansible 部署服务。如果您在安装 OpenShift 时进行部署，则 PV 将自动创建和使用。如果您使用组件 playbook 进行部署，则在 OpenShift 安装之后，Ansible 会创建所需的任何 PVC，并在Trident为其配置存储之后部署服务。

上述变量以及部署过程可能会随着 OpenShift 的每个版本而改变。请务必查看并遵循"红帽 OpenShift 部署指南"请根据您的环境配置您的版本。

数据保护和灾难恢复

了解Trident的保护和恢复选项以及使用Trident创建的卷。对于每个有持久化需求的应用程序，都应该制定数据保护和恢复策略。

Trident复制和恢复

您可以创建备份，以便在发生灾难时恢复Trident 。

Trident复制

Trident使用 Kubernetes CRD 来存储和管理自己的状态，并使用 Kubernetes 集群 etcd 来存储其元数据。

步骤

1. 使用以下命令备份 Kubernetes 集群 etcd"Kubernetes：备份 etcd 集群" 。

2. 将备份文件放置在FlexVol volume上

> NetApp建议您使用SnapMirror关系将FlexVol所在的 SVM 与其他 SVM 连接起来，从而保护该 SVM。

Trident恢复

使用 Kubernetes CRD 和 Kubernetes 集群 etcd 快照，您可以恢复Trident。

步骤

1. 从目标 SVM 上，将包含 Kubernetes etcd 数据文件和证书的卷挂载到将要设置为主节点的主机上。
2. 复制 Kubernetes 集群所需的所有证书。 `/etc/kubernetes/pki`以及 `etcd` 成员文件 `/var/lib/etcd`。
3. 使用 etcd 备份恢复 Kubernetes 集群"Kubernetes：恢复 etcd 集群"。
4. 跑步 `kubectl get crd`验证所有Trident自定义资源是否已启动，并检索Trident对象以验证所有数据是否可用。

SVM复制和恢复

Trident无法配置复制关系，但是存储管理员可以使用 "ONTAP SnapMirror"复制 SVM。

发生灾难时，您可以激活SnapMirror目标 SVM 开始提供数据服务。系统恢复后，您可以切换回主服务器。

关于此任务

使用SnapMirror SVM 复制功能时，请考虑以下事项：

- 对于启用了 SVM-DR 的每个 SVM，您应该创建一个独立的后端。
- 配置存储类，仅在需要时选择复制后端，以避免将不需要复制的卷配置到支持 SVM-DR 的后端上。
- 应用程序管理员应了解复制带来的额外成本和复杂性，并在开始此过程之前仔细考虑其恢复计划。

SVM复制

您可以使用"ONTAP： SnapMirror SVM 复制"创建 SVM 复制关系。

SnapMirror允许您设置选项来控制要复制的内容。您需要知道您在执行操作时选择了哪些选项。使用Trident进行 SVM 恢复 。

- "-identity-preserve true"复制整个 SVM 配置。
- "-丢弃网络配置"不包括 LIF 和相关网络设置。
- "-identity-preserve false"仅复制卷和安全配置。

使用**Trident**进行 **SVM** 恢复

Trident无法自动检测 SVM 故障。如果发生灾难，管理员可以手动启动Trident故障转移到新的 SVM。

步骤

1. 取消已安排和正在进行的SnapMirror传输，断开复制关系，停止源 SVM，然后激活SnapMirror目标 SVM。
2. 如果您指定 `-identity-preserve false`或者 `-discard-config network`配置 SVM 复制时，请更新以下内容：`managementLIF`和 `dataLIF`在Trident后端定义文件中。
3. 确认 `storagePrefix`存在于Trident后端定义文件中。此参数无法更改。省略 `storagePrefix`这将导致后端更新失败。
4. 使用以下命令更新所有必需的后端，以反映新的目标 SVM 名称：

```
./tridentctl update backend <backend-name> -f <backend-json-file> -n
<namespace>
```

5. 如果您指定 `-identity-preserve false`或者 `discard-config network`您必须重启所有应用程序 pod。

> 如果您指定 `-identity-preserve true`当目标 SVM 激活时， Trident提供的所有卷开始提供数据服务。

卷复制和恢复

Trident无法配置SnapMirror复制关系，但是存储管理员可以使用"ONTAP SnapMirror复制和恢复"复制Trident创建的卷。

然后，您可以使用以下方法将恢复的卷导入到Trident中："tridentctl 卷导入" 。

> 不支持导入 `ontap-nas-economy`， `ontap-san-economy` ， 或者 `ontap-flexgroup-economy`司机。

快照数据保护

您可以使用以下方法保护和恢复数据：

- 外部快照控制器和 CRD 用于创建持久卷 (PV) 的 Kubernetes 卷快照。

  "卷快照"

- ONTAP快照用于恢复卷的全部内容，或恢复单个文件或 LUN。

  "ONTAP快照"

安全性

安全性

请按照此处列出的建议，确保您的Trident安装安全可靠。

在Trident自身的命名空间中运行它

防止应用程序、应用程序管理员、用户和管理应用程序访问Trident对象定义或 pod 非常重要，以确保可靠的存储并阻止潜在的恶意活动。

为了将其他应用程序和用户与Trident隔离，请始终将Trident安装在其自身的 Kubernetes 命名空间中。(`trident`）。将Trident放在自己的命名空间中，可以确保只有 Kubernetes 管理员才能访问Trident pod 和存储在命名空间 CRD 对象中的工件（例如后端和 CHAP 密钥，如果适用）。您应该确保只有管理员才能访问Trident命名空间，从而获得对以下方面的访问权限： `tridentctl`应用。

使用 CHAP 身份验证与ONTAP SAN 后端配合使用

Trident支持基于 CHAP 的ONTAP SAN 工作负载身份验证（使用 `ontap-san`和 `ontap-san-economy`司机）。NetApp建议使用Trident的双向 CHAP 进行主机与存储后端之间的身份验证。

对于使用 SAN 存储驱动程序的ONTAP后端， Trident可以通过以下方式设置双向 CHAP 并管理 CHAP 用户名和密钥： `tridentctl`。请参阅"准备配置后端ONTAP SAN 驱动程序"了解Trident如何在ONTAP后端配置CHAP。

使用 CHAP 身份验证连接NetApp HCI和SolidFire后端

NetApp建议部署双向 CHAP，以确保主机与NetApp HCI和SolidFire后端之间的身份验证。 Trident使用一个包含每个租户两个 CHAP 密码的秘密对象。安装Trident后，它会管理 CHAP 密钥并将其存储在一个位置。`tridentvolume`相应 PV 的 CR 对象。创建 PV 时， Trident使用 CHAP 密钥启动 iSCSI 会话，并通过 CHAP 与NetApp HCI和SolidFire系统通信。

> Trident创建的卷不与任何卷访问组关联。

将Trident与 NVE 和 NAE 结合使用

NetApp ONTAP提供静态数据加密，以保护敏感数据，防止磁盘被盗、退回或重新利用。有关详细信息，请参阅"配置NetApp卷加密概述"。

- 如果后端启用了 NAE，则在Trident中配置的任何卷都将启用 NAE。
  - 您可以将 NVE 加密标志设置为 `""`创建支持 NAE 的卷。
- 如果后端未启用 NAE，则在Trident中配置的任何卷都将启用 NVE，除非 NVE 加密标志设置为 `false`（后端配置中的默认值）。

> 在启用 NAE 的后端上使用Trident创建的卷必须使用 NVE 或 NAE 加密。
>
> - 您可以将 NVE 加密标志设置为 `true`在Trident后端配置中，可以覆盖 NAE 加密，并按卷使用特定的加密密钥。
> - 将 NVE 加密标志设置为 `false`在启用 NAE 的后端上创建启用 NAE 的卷。您无法通过将 NVE 加密标志设置为 来禁用 NAE 加密。 `false`。

- 您可以通过显式设置 NVE 加密标志，在Trident中手动创建 NVE 卷。 `true`。

有关后端配置选项的更多信息，请参阅：

- "ONTAP SAN 配置选项"

- "ONTAP NAS 配置选项"

Linux 统一密钥设置 (LUKS)

您可以启用 Linux 统一密钥设置 (LUKS) 来加密Trident上的ONTAP SAN 和ONTAP SAN ECONOMY 卷。 Trident支持 LUKS 加密卷的密码短语轮换和卷扩展。

在Trident中，LUKS 加密卷使用 aes-xts-plain64 密码和模式，这是推荐的。"美国国家标准与技术研究院" 。

ASA r2 系统不支持 LUKS 加密。有关ASA r2 系统的信息，请参阅"了解ASA r2 存储系统"。

开始之前

- 工作节点必须安装 cryptsetup 2.1 或更高版本（但低于 3.0）。欲了解更多信息，请访问"GitLab：加密设置"。
- 出于性能方面的考虑， NetApp建议工作节点支持高级加密标准新指令 (AES-NI)。要验证是否支持 AES-NI，请运行以下命令：

```
grep "aes" /proc/cpuinfo
```

如果没有返回任何内容，则说明您的处理器不支持 AES-NI。有关 AES-NI 的更多信息，请访问："英特尔：高级加密标准指令集（AES-NI）"。

启用 LUKS 加密

您可以使用 Linux 统一密钥设置 (LUKS) 为ONTAP SAN 和ONTAP SAN ECONOMY 卷启用按卷主机端加密。

步骤

1. 在后端配置中定义 LUKS 加密属性。有关ONTAP SAN 后端配置选项的更多信息，请参阅"ONTAP SAN 配置选项"。

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. 使用 `parameters.selector`使用 LUKS 加密定义存储池。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. 创建一个包含 LUKS 密码短语的密钥。例如：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

限制

LUKS 加密卷无法利用ONTAP重复数据删除和压缩功能。

导入 **LUKS** 卷的后端配置

要导入 LUKS 卷，您必须设置 `luksEncryption`到（`true`在后端。这 `luksEncryption`此选项用于告知Trident该卷是否符合 LUKS 标准。（`true`或不符合 LUKS 标准（`false`）如下例所示。

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

用于导入 **LUKS** 卷的 **PVC** 配置

要动态导入 LUKS 卷，请设置注释 `trident.netapp.io/luksEncryption`到 `true`并在 PVC 中包含一个支持 LUKS 的存储类，如本例所示。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

轮换 **LUKS** 密码短语

您可以轮换 LUKS 密码短语并确认轮换。

> 在您确认任何卷、快照或密钥不再引用该密码短语之前，请勿忘记该密码短语。如果引用的密码短语丢失，您可能无法挂载卷，数据将保持加密状态且无法访问。

关于此任务

当指定新的 LUKS 密码短语后创建挂载卷的 pod 时，就会发生 LUKS 密码短语轮换。当创建一个新的 pod 时，Trident会将卷上的 LUKS 密码短语与密钥中的活动密码短语进行比较。

- 如果卷上的密码短语与密钥中的活动密码短语不匹配，则会发生轮换。
- 如果卷上的密码短语与密钥中的活动密码短语匹配，则 `previous-luks-passphrase`参数被忽略。

步骤

1. 添加 `node-publish-secret-name`和 `node-publish-secret-namespace`StorageClass 参数。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. 识别卷或快照上已存在的密码短语。

   卷

   ```
   tridentctl -d get volume luks-pvc1
   GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

   ...luksPassphraseNames:["A"]
   ```

   Snapshot

   ```
   tridentctl -d get snapshot luks-pvc1
   GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

   ...luksPassphraseNames:["A"]
   ```

3. 更新卷的 LUKS 密钥，以指定新的和以前的密码短语。确保 `previous-luke-passphrase-name`和 `previous-luks-passphrase`与之前的密码短语匹配。

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: luks-pvc1
   stringData:
     luks-passphrase-name: B
     luks-passphrase: secretB
     previous-luks-passphrase-name: A
     previous-luks-passphrase: secretA
   ```

4. 创建一个新的 pod，挂载该卷。这是启动旋转所必需的步骤。
5. 确认密码短语已轮换。

   卷

   ```
   tridentctl -d get volume luks-pvc1
   GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

   ...luksPassphraseNames:["B"]
   ```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

结果

当卷和快照上仅返回新密码短语时，密码短语就会轮换。

> 例如，如果返回两个密码短语。 `luksPassphraseNames: ["B", "A"]`旋转不完整。您可以触发一个新的舱体来尝试完成轮换。

启用卷扩展

您可以对 LUKS 加密卷启用卷扩展。

步骤

1. 启用 `CSINodeExpandSecret`功能门（beta 1.25+）。参考 "Kubernetes 1.25：使用密钥实现 CSI 卷的节点驱动扩展" 了解详情。
2. 添加 `node-expand-secret-name`和 `node-expand-secret-namespace`StorageClass 参数。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

结果

启动在线存储扩展时，kubelet 会将相应的凭据传递给驱动程序。

Kerberos 飞行中加密

使用 Kerberos 传输中加密，您可以对托管集群和存储后端之间的流量启用加密，从而提高数据访问安全性。

Trident支持以ONTAP作为存储后端的 Kerberos 加密：

- 本地**ONTAP** - Trident支持通过 NFSv3 和 NFSv4 连接对来自 Red Hat OpenShift 和上游 Kubernetes 集群的本地ONTAP卷进行 Kerberos 加密。

您可以创建、删除、调整大小、创建快照、克隆、只读克隆和导入使用 NFS 加密的卷。

配置本地**ONTAP**卷的飞行中 **Kerberos** 加密

您可以为托管集群和本地ONTAP存储后端之间的存储流量启用 Kerberos 加密。

> 仅使用以下方式支持对本地ONTAP存储后端的 NFS 流量进行 Kerberos 加密： `ontap-nas`存储驱动程序。

开始之前

- 确保您可以访问 `tridentctl`公用事业。
- 请确保您拥有ONTAP存储后端的管理员权限。
- 请确保您知道要从ONTAP存储后端共享的卷的名称。
- 请确保您已准备好ONTAP存储 VM，以支持 NFS 卷的 Kerberos 加密。请参阅 "在数据 LIF 上启用 Kerberos"以获取说明。
- 请确保所有与 Kerberos 加密一起使用的 NFSv4 卷都已正确配置。请参阅NetApp NFSv4 域配置部分（第 13 页）。 "NetApp NFSv4 增强功能和最佳实践指南" 。

添加或修改**ONTAP**导出策略

您需要向现有的ONTAP导出策略添加规则，或者创建新的导出策略，以支持对ONTAP存储 VM 根卷以及与上游 Kubernetes 集群共享的任何ONTAP卷进行 Kerberos 加密。您添加的导出策略规则或您创建的新导出策略需要支持以下访问协议和访问权限：

访问协议

配置导出策略，支持 NFS、NFSv3 和 NFSv4 访问协议。

访问详情

您可以根据卷的需求配置三种不同版本的 Kerberos 加密之一：

- **Kerberos 5** -（身份验证和加密）
- **Kerberos 5i** -（身份验证和加密，具有身份保护功能）
- **Kerberos 5p** -（身份验证和加密，提供身份和隐私保护）

配置ONTAP导出策略规则，使其具有适当的访问权限。例如，如果集群将使用 Kerberos 5i 和 Kerberos 5p 混合加密方式挂载 NFS 卷，请使用以下访问设置：

类型	只读访问权限	读/写权限	超级用户访问权限
UNIX	已启用	已启用	已启用
Kerberos 5i	已启用	已启用	已启用
Kerberos 5p	已启用	已启用	已启用

有关如何创建ONTAP导出策略和导出策略规则，请参阅以下文档：

- "创建导出策略"
- "向导出策略添加规则"

创建存储后端

您可以创建包含 Kerberos 加密功能的Trident存储后端配置。

关于此任务

创建配置 Kerberos 加密的存储后端配置文件时，可以使用以下方式指定三种不同版本的 Kerberos 加密之一：`spec.nfsMountOptions`范围：

- `spec.nfsMountOptions: sec=krb5`（身份验证和加密）
- `spec.nfsMountOptions: sec=krb5i`（身份验证和加密，以及身份保护）
- `spec.nfsMountOptions: sec=krb5p`（身份验证和加密，以及身份和隐私保护）

只能指定一个 Kerberos 级别。如果在参数列表中指定多个 Kerberos 加密级别，则仅使用第一个选项。

步骤

1. 在托管集群上，使用以下示例创建存储后端配置文件。将方括号 <> 中的值替换为您环境中的信息：

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. 使用上一步创建的配置文件创建后端：

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

如果后端创建失败，则后端配置存在问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在您发现并纠正配置文件中的问题后，您可以再次运行创建命令。

创建存储类

您可以创建存储类来配置具有 Kerberos 加密的卷。

关于此任务

创建存储类对象时，可以使用以下方式指定三种不同版本的 Kerberos 加密之一： `mountOptions`范围：

- `mountOptions: sec=krb5`（身份验证和加密）
- `mountOptions: sec=krb5i`（身份验证和加密，以及身份保护）
- `mountOptions: sec=krb5p`（身份验证和加密，以及身份和隐私保护）

只能指定一个 Kerberos 级别。如果在参数列表中指定多个 Kerberos 加密级别，则仅使用第一个选项。如果在存储后端配置中指定的加密级别与在存储类对象中指定的加密级别不同，则以存储类对象为准。

步骤

1. 创建一个 StorageClass Kubernetes 对象，请参考以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. 创建存储类：

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. 请确保已创建存储类：

```
kubectl get sc ontap-nas-sc
```

您应该会看到类似以下内容的输出：

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

供应量

创建存储后端和存储类之后，现在可以配置卷了。有关说明，请参阅 "提供一定量"。

使用**Azure NetApp Files**卷配置飞行中 **Kerberos** 加密

您可以为托管群集与单个Azure NetApp Files存储后端或Azure NetApp Files存储后端虚拟池之间的存储流量启用 Kerberos 加密。

开始之前

- 请确保已在受管 Red Hat OpenShift 集群上启用Trident 。
- 确保您可以访问 `tridentctl`公用事业。
- 请确保您已按照以下说明准备好Azure NetApp Files存储后端以进行 Kerberos 加密：注意相关要求并按照说明进行操作。 "Azure NetApp Files文档" 。
- 请确保所有与 Kerberos 加密一起使用的 NFSv4 卷都已正确配置。请参阅NetApp NFSv4 域配置部分（第 13 页）。 "NetApp NFSv4 增强功能和最佳实践指南" 。

创建存储后端

您可以创建包含 Kerberos 加密功能的Azure NetApp Files存储后端配置。

关于此任务

创建配置 Kerberos 加密的存储后端配置文件时，您可以将其定义为应用于以下两个级别之一：

- 使用*存储后端级别* `spec.kerberos`场地
- 使用*虚拟池级别* `spec.storage.kerberos`场地

在虚拟池级别定义配置时，使用存储类中的标签选择池。

无论在哪个级别，您都可以指定三种不同版本的 Kerberos 加密之一：

- `kerberos: sec=krb5`（身份验证和加密）
- `kerberos: sec=krb5i`（身份验证和加密，以及身份保护）
- `kerberos: sec=krb5p`（身份验证和加密，以及身份和隐私保护）

步骤

1. 在托管集群上，根据您需要定义存储后端的位置（存储后端级别或虚拟池级别），使用以下示例之一创建存储后端配置文件。将方括号 <> 中的值替换为您环境中的信息：

存储后端示例

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

虚拟池级别示例

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

2. 使用上一步创建的配置文件创建后端：

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

如果后端创建失败，则后端配置存在问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在您发现并纠正配置文件中的问题后，您可以再次运行创建命令。

创建存储类

您可以创建存储类来配置具有 Kerberos 加密的卷。

步骤

1. 创建一个 StorageClass Kubernetes 对象，请参考以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. 创建存储类：

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. 请确保已创建存储类：

```
kubectl get sc -sc-nfs
```

您应该会看到类似以下内容的输出：

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

供应量

创建存储后端和存储类之后，现在可以配置卷了。有关说明，请参阅 "提供一定量"。

使用Trident Protect 保护应用程序

了解Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强了由NetApp ONTAP存储系统和NetApp Trident CSI 存储供应器支持的有状态 Kubernetes 应用程序的功能和可用性。Trident Protect 简化了跨公有云和本地环境的容器化工作负载的管理、保护和迁移。它还通过其 API 和 CLI 提供自动化功能。

您可以通过创建自定义资源 (CR) 或使用Trident Protect CLI 来使用Trident Protect 保护应用程序。

下一步是什么?

您可以先了解Trident Protect 的相关要求，然后再进行安装：

- "Trident保护要求"

安装Trident Protect

Trident保护要求

首先，请验证您的运行环境、应用程序集群、应用程序和许可证是否准备就绪。确保您的环境满足部署和运行Trident Protect 的这些要求。

Trident Protect Kubernetes 集群兼容性

Trident Protect 与各种完全托管和自托管的 Kubernetes 产品兼容，包括：

- 亚马逊弹性 Kubernetes 服务 (EKS)
- 谷歌 Kubernetes 引擎 (GKE)
- Microsoft Azure Kubernetes 服务 (AKS)
- 红帽 OpenShift
- SUSE Rancher
- VMware Tanzu 产品组合
- 上游 Kubernetes

- Trident Protect备份仅支持Linux计算节点。Windows 计算节点不支持备份操作。
- 确保安装Trident Protect 的集群已配置运行中的快照控制器和相关的 CRD。要安装快照控制器，请参阅 "这些说明"。

Trident Protect 存储后端兼容性

Trident Protect 支持以下存储后端：

- Amazon FSx for NetApp ONTAP
- Cloud Volumes ONTAP
- ONTAP存储阵列
- Google Cloud NetApp Volumes
- Azure NetApp Files

请确保您的存储后端满足以下要求：

- 确保连接到集群的NetApp存储使用的是Trident 24.02 或更高版本（建议使用Trident 24.10）。
- 请确保您拥有NetApp ONTAP存储后端。
- 请确保您已配置用于存储备份的对象存储桶。
- 创建您计划用于应用程序或应用程序数据管理操作的任何应用程序命名空间。Trident Protect 不会为您创建这些命名空间；如果您在自定义资源中指定了不存在的命名空间，则操作将失败。

nas-economy 容量要求

Trident Protect 支持对 nas-economy 卷进行备份和恢复操作。目前不支持将快照、克隆和SnapMirror复制到 nas-economy 卷。您需要为计划与Trident Protect 一起使用的每个 nas-economy 卷启用快照目录。

某些应用程序与使用快照目录的卷不兼容。对于这些应用，您需要通过在ONTAP存储系统上运行以下命令来隐藏快照目录：

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

您可以通过对每个 nas-economy 卷运行以下命令来启用快照目录，并将命令替换为 `<volume-UUID>`使用要更改的卷的 UUID：

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

您可以通过设置Trident后端配置选项，为新卷默认启用快照目录。 `snapshotDir`到 `true`。现有卷数不受影响。

使用 **KubeVirt** 虚拟机保护数据

Trident Protect 24.10 和 24.10.1 及更高版本在保护运行在 KubeVirt VM 上的应用程序时，行为有所不同。对于这两个版本，您都可以在数据保护操作期间启用或禁用文件系统冻结和解冻结。

在恢复操作期间，任何 `VirtualMachineSnapshots`为虚拟机 (VM) 创建的配置文件无法恢复。

Trident Protect 24.10

Trident Protect 24.10 在数据保护操作期间不会自动确保 KubeVirt VM 文件系统的一致性状态。如果您想使用Trident Protect 24.10 保护您的 KubeVirt VM 数据，则需要在执行数据保护操作之前手动启用文件系统的冻结/解冻功能。这样可以确保文件系统处于一致状态。

您可以配置Trident Protect 24.10 来管理数据保护操作期间 VM 文件系统的冻结和解冻。"配置虚拟化"然后使用以下命令：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

Trident Protect 24.10.1 及更高版本

从Trident Protect 24.10.1 开始， Trident Protect 会在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。您也可以使用以下命令禁用此自动行为：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

SnapMirror复制的要求

NetApp SnapMirror复制功能可与Trident Protect 配合使用，适用于以下ONTAP解决方案：

- 本地部署的NetApp FAS、 AFF和ASA集群
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

ONTAP集群对SnapMirror复制的要求

如果您计划使用SnapMirror复制功能，请确保您的ONTAP集群满足以下要求：

- * NetApp Trident *：使用ONTAP作为后端服务的源 Kubernetes 集群和目标 Kubernetes 集群上都必须存在NetApp Trident 。Trident Protect 支持使用NetApp SnapMirror技术进行复制，该技术使用以下驱动程序支持的存储类：
  - `ontap-nas`：极品飞车
  - `ontap-san`：iSCSI
  - `ontap-san`：FC
  - `ontap-san`：NVMe/TCP（最低要求ONTAP版本 9.15.1）
- 许可证：使用数据保护包的ONTAP SnapMirror异步许可证必须在源 ONTAP 集群和目标ONTAP集群上启用。请参阅 "ONTAP中的SnapMirror许可概述"了解更多信息。

  从ONTAP 9.10.1 开始，所有许可证均以NetApp许可证文件 (NLF) 的形式交付，这是一个可以启用多种功能的单个文件。请参阅"ONTAP One 附带的许可证"了解更多信息。

  仅支持SnapMirror异步保护。

SnapMirror复制的对等连接注意事项

如果您计划使用存储后端对等互连，请确保您的环境满足以下要求：

- 集群和 **SVM**： ONTAP存储后端必须相互连接。请参阅 "集群和SVM对等连接概述"了解更多信息。

  确保两个ONTAP集群之间复制关系中使用的 SVM 名称是唯一的。

- * NetApp Trident和 SVM*：对等远程 SVM 必须可供目标集群上的NetApp Trident使用。
- 托管后端：您需要在Trident Protect 中添加和管理ONTAP存储后端，以创建复制关系。

用于SnapMirror复制的Trident / ONTAP配置

Trident Protect 要求您至少配置一个支持源集群和目标集群复制的存储后端。如果源集群和目标集群相同，为了获得最佳弹性，目标应用程序应该使用与源应用程序不同的存储后端。

SnapMirror复制的 Kubernetes 集群要求

请确保您的 Kubernetes 集群满足以下要求：

- **AppVault** 可访问性：源集群和目标集群都必须具有网络访问权限，才能从 AppVault 读取数据和写入数据，以实现应用程序对象复制。
- 网络连接：配置防火墙规则、存储桶权限和 IP 允许列表，以启用两个集群和 AppVault 之间通过 WAN 进行通信。

许多企业环境在广域网连接中实施严格的防火墙策略。在配置复制之前，请与您的基础架构团队确认这些网络要求。

安装并配置Trident Protect

如果您的环境满足Trident Protect 的要求，您可以按照以下步骤在集群上安装Trident Protect。您可以从NetApp获取Trident Protect，或者从您自己的私有注册表中安装它。如果您的集群无法访问互联网，从私有注册表安装会很有帮助。

安装Trident Protect

从NetApp安装Trident Protect

步骤

1. 添加Trident Helm 仓库：

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

2. 使用 Helm 安装Trident Protect。代替 `<name-of-cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2506.0 --create
-namespace --namespace trident-protect
```

从私有注册表安装Trident Protect

如果您的 Kubernetes 集群无法访问互联网，您可以从私有镜像仓库安装Trident Protect。在这些示例中，请将括号中的值替换为您环境中的信息：

步骤

1. 将以下镜像拉取到本地计算机，更新标签，然后将其推送到您的私有镜像仓库：

```
netapp/controller:25.06.0
netapp/restic:25.06.0
netapp/kopia:25.06.0
netapp/trident-autosupport:25.06.0
netapp/exechook:25.06.0
netapp/resourcebackup:25.06.0
netapp/resourcerestore:25.06.0
netapp/resourcedelete:25.06.0
bitnami/kubectl:1.30.2
kubebuilder/kube-rbac-proxy:v0.16.0
```

   例如：

```
docker pull netapp/controller:25.06.0
```

```
docker tag netapp/controller:25.06.0 <private-registry-
url>/controller:25.06.0
```

```
docker push <private-registry-url>/controller:25.06.0
```

2. 创建Trident Protect 系统命名空间：

```
kubectl create ns trident-protect
```

3. 登录注册表：

```
helm registry login <private-registry-url> -u <account-id> -p <api-token>
```

4. 创建用于私有注册表身份验证的拉取密钥：

```
kubectl create secret docker-registry regcred --docker-username=<registry-username> --docker-password=<api-token> -n trident-protect --docker-server=<private-registry-url>
```

5. 添加Trident Helm 仓库：

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

6. 创建一个名为的文件 `protectValues.yaml`。请确保其中包含以下Trident Protect 设置：

```
---
image:
  registry: <private-registry-url>
imagePullSecrets:
  - name: regcred
controller:
  image:
    registry: <private-registry-url>
rbacProxy:
  image:
    registry: <private-registry-url>
crCleanup:
  imagePullSecrets:
    - name: regcred
webhooksCleanup:
  imagePullSecrets:
    - name: regcred
```

7. 使用 Helm 安装Trident Protect。代替 `<name_of_cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name_of_cluster> --version 100.2506.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

安装Trident Protect CLI 插件

您可以使用Trident Protect 命令行插件，它是Trident的一个扩展。 `tridentctl`用于创建和与Trident Protect 自定义资源 (CR) 交互的实用程序。

安装Trident Protect CLI 插件

在使用命令行实用程序之前，需要将其安装在用于访问集群的计算机上。根据您的机器使用的是 x64 还是ARM CPU，请按照以下步骤操作。

下载适用于 **Linux AMD64 CPU** 的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-linux-amd64
```

下载适用于 **Linux ARM64 CPU** 的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-linux-arm64
```

下载适用于 **Mac AMD64 CPU** 的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-macos-amd64
```

下载适用于 **Mac ARM64 CPU** 的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.06.0/tridentctl-protect-macos-arm64
```

1. 启用插件二进制文件的执行权限：

```
chmod +x tridentctl-protect
```

2. 将插件二进制文件复制到 PATH 环境变量中定义的位置。例如， `/usr/bin`或者 `/usr/local/bin`（您可能需要更高的权限）：

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 您也可以选择将插件二进制文件复制到您主目录中的某个位置。在这种情况下，建议确保该位置已添加到您的 PATH 环境变量中：

```
cp ./tridentctl-protect ~/bin/
```

将插件复制到 PATH 环境变量中的某个位置，即可通过键入以下命令来使用该插件。 `tridentctl-protect`或者 `tridentctl protect`从任何地点。

查看**Trident CLI** 插件帮助

您可以使用插件内置的帮助功能来获取有关插件功能的详细帮助：

步骤

1. 使用帮助功能查看使用指南：

```
tridentctl-protect help
```

启用命令自动补全

安装Trident Protect CLI 插件后，您可以为某些命令启用自动补全功能。

为 **Bash shell** 启用自动补全功能

步骤

1. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/tridentctl-completion.bash
```

2. 在您的用户目录下创建一个新目录，用于存放脚本：

```
mkdir -p ~/.bash/completions
```

3. 将下载的脚本移动到 `~/.bash/completions`目录：

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. 将以下行添加到 `~/.bashrc`您主目录中的文件：

```
source ~/.bash/completions/tridentctl-completion.bash
```

为 **Z shell** 启用自动补全

步骤

1. 下载完成脚本：

```
curl -L -O https://github.com/NetApp/tridentctl-
protect/releases/download/25.06.0/tridentctl-completion.zsh
```

2. 在您的用户目录下创建一个新目录，用于存放脚本：

```
mkdir -p ~/.zsh/completions
```

3. 将下载的脚本移动到 `~/.zsh/completions`目录：

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. 将以下行添加到 `~/.zprofile`您主目录中的文件：

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

结果

下次登录 shell 时，您可以使用 tridentctl-protect 插件进行命令自动补全。

定制Trident Protect 安装

您可以自定义Trident Protect 的默认配置，以满足您环境的特定要求。

指定Trident Protect 容器资源限制

安装Trident Protect 后，您可以使用配置文件来指定Trident Protect 容器的资源限制。设置资源限制可以控制Trident Protect 操作消耗集群资源的程度。

步骤

1. 创建一个名为的文件 `resourceLimits.yaml`。
2. 根据您的环境需求，在文件中填充Trident Protect 容器的资源限制选项。

   以下示例配置文件显示了可用设置，并包含每个资源限制的默认值：

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
```

```
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. 应用以下值 `resourceLimits.yaml`文件：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

自定义安全上下文约束

安装Trident Protect 后，您可以使用配置文件来修改Trident Protect 容器的 OpenShift 安全上下文约束 (SCC)。这些约束定义了 Red Hat OpenShift 集群中 pod 的安全限制。

步骤

1. 创建一个名为的文件 `sccconfig.yaml`。
2. 在文件中添加 SCC 选项，并根据您的环境需要修改参数。

   以下示例显示了 SCC 选项的参数默认值：

```
scc:
  create: true
  name: trident-protect-job
  priority: 1
```

下表描述了SCC选项的参数：

参数	描述	默认
create	确定是否可以创建 SCC 资源。仅当 `scc.create`设置为 `true`Helm 安装过程会识别 OpenShift 环境。如果不是在 OpenShift 上运行，或者如果 `scc.create`设置为 `false`不会创建 SCC 资源。	true
name	指定SCC的名称。	三叉戟保护工作
优先事项	确定 SCC 的优先级。优先级较高的SCC会优先于优先级较低的SCC进行评估。	1

3. 应用以下值 `sccconfig.yaml`文件：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect -f
sccconfig.yaml --reuse-values
```

这将用指定的值替换默认值。 `sccconfig.yaml`文件。

配置其他**Trident Protect** 舵图设置

您可以自定义AutoSupport设置和命名空间过滤以满足您的特定要求。下表描述了可用的配置参数：

参数	类型	描述
自动支持代理	string	为NetApp AutoSupport连接配置代理 URL。使用此功能通过代理服务器路由支持包上传。例子：`http://my.proxy.url` 。
自动支持.不安全	布尔值	设置为“跳过AutoSupport代理连接的 TLS 验证” `true` 。仅用于不安全的代理连接。 （默认： `false` ）
自动支持已启用	布尔值	启用或禁用每日Trident Protect AutoSupport捆绑包上传。设置为 `false`每日定时上传功能已禁用，但您仍然可以手动生成支持包。 （默认： `true` ）

参数	类型	描述
恢复跳过命名空间注释	string	要从备份和恢复操作中排除的命名空间注释的逗号分隔列表。允许您根据注释过滤命名空间。
restoreSkipNamespaceLabels	string	要从备份和恢复操作中排除的命名空间标签的逗号分隔列表。允许您根据标签过滤命名空间。

您可以使用 YAML 配置文件或命令行标志配置这些选项：

使用 **YAML** 文件

步骤

1. 创建一个配置文件并将其命名为 `values.yaml`。
2. 在您创建的文件中，添加您想要自定义的配置选项。

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
```

3. 填写完之后 `values.yaml`将包含正确值的文件应用配置文件：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

使用 **CLI** 标志

步骤

1. 使用以下命令： `--set`用于指定各个参数的标志：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set restoreSkipNamespaceAnnotations="annotation1,annotation2" \
  --set restoreSkipNamespaceLabels="label1,label2" \
  --reuse-values
```

将Trident Protect Pod 限制在特定节点上

您可以使用 Kubernetes nodeSelector 节点选择约束，根据节点标签来控制哪些节点有资格运行Trident Protect pod。默认情况下， Trident Protect 仅限于运行 Linux 的节点。您可以根据需要进一步自定义这些限制条件。

步骤

1. 创建一个名为的文件 `nodeSelectorConfig.yaml`。
2. 在文件中添加 nodeSelector 选项，并修改文件以添加或更改节点标签，从而根据您的环境需要进行限制。例如，以下文件包含默认的操作系统限制，但同时也针对特定区域和应用程序名称：

```
nodeSelector:
  kubernetes.io/os: linux
  region: us-west
  app.kubernetes.io/name: mysql
```

3. 应用以下值 `nodeSelectorConfig.yaml`文件：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

这将默认限制替换为您在设置中指定的限制。 `nodeSelectorConfig.yaml`文件。

管理Trident Protect

管理Trident Protect 授权和访问控制

Trident Protect 使用 Kubernetes 的基于角色的访问控制 (RBAC) 模型。默认情况下，Trident Protect 提供一个系统命名空间及其关联的默认服务帐户。如果您的组织拥有众多用户或特定的安全需求，则可以使用Trident Protect 的 RBAC 功能来更精细地控制对资源和命名空间的访问。

集群管理员始终拥有对默认资源的访问权限。 `trident-protect`命名空间，并且可以访问所有其他命名空间中的资源。要控制对资源和应用程序的访问，您需要创建额外的命名空间，并将资源和应用程序添加到这些命名空间中。

请注意，默认情况下，任何用户都无法创建应用程序数据管理变更请求 (CR)。 `trident-protect`命名空间。您需要在应用程序命名空间中创建应用程序数据管理 CR（最佳实践是在与其关联的应用程序相同的命名空间中创建应用程序数据管理 CR）。

只有管理员才能访问具有特权的Trident Protect 自定义资源对象，其中包括：

- **AppVault**：需要存储桶凭证数据
- **AutoSupportBundle**：收集指标、日志和其他敏感的Trident Protect数据
- **AutoSupportBundleSchedule**：管理日志收集计划

最佳实践是使用基于角色的访问控制 (RBAC) 将对特权对象的访问限制在管理员范围内。

有关基于角色的访问控制 (RBAC) 如何管理对资源和命名空间的访问的更多信息，请参阅…… "Kubernetes RBAC 文档" 。

有关服务帐户的信息，请参阅 "Kubernetes 服务帐户文档"。

示例：管理两组用户的访问权限

例如，一个组织有集群管理员、一组工程用户和一组市场营销用户。集群管理员将完成以下任务，以创建一个环境，其中工程组和市场营销组各自只能访问分配给其各自命名空间的资源。

步骤 **1**：创建命名空间以包含每个组的资源

创建命名空间可以让你从逻辑上分离资源，并更好地控制谁可以访问这些资源。

步骤

1. 为工程组创建一个命名空间：

```
kubectl create ns engineering-ns
```

2. 为市场营销组创建命名空间：

```
kubectl create ns marketing-ns
```

步骤 **2**：创建新的服务帐户，以便与每个命名空间中的资源进行交互

您创建的每个新命名空间都带有一个默认服务帐户，但您应该为每个用户组创建一个服务帐户，以便将来必要时可以进一步在组之间划分权限。

步骤

1. 为工程团队创建一个服务帐户：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. 为市场营销团队创建一个服务帐户：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

步骤 **3**：为每个新服务帐户创建一个密钥

服务帐户密钥用于对服务帐户进行身份验证，如果遭到泄露，可以轻松删除并重新创建。

步骤

1. 为工程服务帐户创建一个密钥：

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. 为营销服务帐户创建一个密钥：

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: mkt-user
  name: mkt-user-secret
  namespace: marketing-ns
type: kubernetes.io/service-account-token
```

步骤 **4**：创建 **RoleBinding** 对象，将 **ClusterRole** 对象绑定到每个新的服务帐户。

安装Trident Protect 时会创建一个默认的 ClusterRole 对象。您可以通过创建和应用 RoleBinding 对象将此 ClusterRole 绑定到服务帐户。

步骤

1. 将集群角色绑定到工程服务帐户：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-ns-tenant-rolebinding
  namespace: engineering-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

2. 将集群角色绑定到营销服务帐户：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: marketing-ns-tenant-rolebinding
  namespace: marketing-ns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-protect-tenant-cluster-role
subjects:
- kind: ServiceAccount
  name: mkt-user
  namespace: marketing-ns
```

步骤 **5**：测试权限

测试权限是否正确。

步骤

1. 确认工程用户可以访问工程资源：

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n engineering-ns
```

2. 确认工程用户无法访问市场营销资源：

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

步骤 **6**：授予对 **AppVault** 对象的访问权限

要执行备份和快照等数据管理任务，集群管理员需要授予各个用户对 AppVault 对象的访问权限。

步骤

1. 创建并应用 AppVault 和密钥组合的 YAML 文件，以授予用户对 AppVault 的访问权限。例如，以下 CR 授予用户对 AppVault 的访问权限 `eng-user`：

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 创建并应用角色 CR，使集群管理员能够授予对命名空间中特定资源的访问权限。例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. 创建并应用角色绑定 CR，将权限绑定到用户 eng-user。例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eng-user-read-appvault-binding
  namespace: trident-protect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: eng-user-appvault-reader
subjects:
- kind: ServiceAccount
  name: eng-user
  namespace: engineering-ns
```

4. 请确认权限是否正确。

   a. 尝试检索所有命名空间的 AppVault 对象信息：

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   您应该会看到类似以下内容的输出：

```
Error from server (Forbidden): appvaults.protect.trident.netapp.io is
forbidden: User "system:serviceaccount:engineering-ns:eng-user"
cannot list resource "appvaults" in API group
"protect.trident.netapp.io" in the namespace "trident-protect"
```

b. 测试用户是否可以获取他们现在有权访问的 AppVault 信息：

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
trident-protect
```

您应该会看到类似以下内容的输出：

```
yes
```

结果

您授予 AppVault 权限的用户应该能够使用授权的 AppVault 对象进行应用程序数据管理操作，并且不应该能够访问分配的命名空间之外的任何资源，或者创建他们无权访问的新资源。

监控Trident保护资源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 开源工具来监控Trident Protect 保护的资源的健康状况。

kube-state-metrics 服务从 Kubernetes API 通信生成指标。将其与Trident Protect 结合使用，可以显示有关环境中资源状态的有用信息。

Prometheus 是一个工具包，它可以接收 kube-state-metrics 生成的数据，并将其呈现为关于这些对象的易于阅读的信息。kube-state-metrics 和 Prometheus 共同提供了一种方法，让您可以监控使用Trident Protect 管理的资源的健康状况和状态。

Alertmanager 是一项服务，它可以接收 Prometheus 等工具发送的警报，并将它们路由到您配置的目标位置。

> 这些步骤中包含的配置和指导仅供参考；您需要根据自己的环境进行自定义。请参阅以下官方文档以获取具体说明和支持：
>
> - "kube-state-metrics 文档"
> - "普罗米修斯文档"
> - "Alertmanager 文档"

步骤 1：安装监控工具

要在Trident Protect 中启用资源监控，您需要安装和配置 kube-state-metrics、Promethus 和 Alertmanager。

安装 **kube-state-metrics**

您可以使用 Helm 安装 kube-state-metrics。

步骤

1. 添加 kube-state-metrics Helm chart。例如：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. 将 Prometheus ServiceMonitor CRD 应用到集群：

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. 为 Helm chart 创建一个配置文件（例如， `metrics-config.yaml` ）。您可以根据自身环境自定义以下示例配置：

metrics-config.yaml：kube-state-metrics Helm chart 配置

```
---
extraArgs:
  # Collect only custom metrics
  - --custom-resource-state-only=true

customResourceState:
  enabled: true
  config:
    kind: CustomResourceStateMetrics
    spec:
      resources:
      - groupVersionKind:
          group: protect.trident.netapp.io
          kind: "Backup"
          version: "v1"
        labelsFromPath:
          backup_uid: [metadata, uid]
          backup_name: [metadata, name]
          creation_time: [metadata, creationTimestamp]
        metrics:
        - name: backup_info
          help: "Exposes details about the Backup state"
          each:
            type: Info
            info:
              labelsFromPath:
                appVaultReference: ["spec", "appVaultRef"]
                appReference: ["spec", "applicationRef"]
rbac:
  extraRules:
  - apiGroups: ["protect.trident.netapp.io"]
    resources: ["backups"]
    verbs: ["list", "watch"]

# Collect metrics from all namespaces
namespaces: ""

# Ensure that the metrics are collected by Prometheus
prometheus:
  monitor:
    enabled: true
```

4. 通过部署 Helm chart 来安装 kube-state-metrics。例如：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. 按照以下说明配置 kube-state-metrics，以生成Trident Protect 使用的自定义资源的指标： "kube-state-metrics 自定义资源文档" 。

安装 Prometheus

您可以按照以下说明安装 Prometheus： "普罗米修斯文档" 。

安装 Alertmanager

您可以按照以下说明安装 Alertmanager： "Alertmanager 文档" 。

步骤 2：配置监控工具以协同工作

安装完监控工具后，需要配置它们以使其协同工作。

步骤

1. 将 kube-state-metrics 与 Prometheus 集成。编辑 Prometheus 配置文件(`prometheus.yaml`）并添加 kube-state-metrics 服务信息。例如：

prometheus.yaml：kube-state-metrics 服务与 Prometheus 的集成

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: trident-protect
data:
  prometheus.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.trident-protect.svc:8080']
```

2. 配置 Prometheus 将警报路由到 Alertmanager。编辑 Prometheus 配置文件(`prometheus.yaml`）并添加以下部分：

prometheus.yaml：向 **Alertmanager** 发送警报

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - alertmanager.trident-protect.svc:9093
```

结果

Prometheus 现在可以从 kube-state-metrics 收集指标，并可以向 Alertmanager 发送警报。现在您可以配置哪些条件会触发警报以及警报应该发送到哪里。

步骤 **3**：配置警报和警报目标

配置好工具协同工作后，还需要配置哪些类型的信息会触发警报，以及警报应该发送到哪里。

警报示例：备份失败

以下示例定义了一个关键警报，当备份自定义资源的状态设置为“是”时，该警报将被触发。 `Error`持续5秒或更长时间。您可以自定义此示例以匹配您的环境，并将此 YAML 代码片段包含在您的项目中。 `prometheus.yaml`配置文件：

rules.yaml：定义备份失败的 **Prometheus** 警报

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

配置 **Alertmanager** 以将警报发送到其他渠道

您可以配置 Alertmanager，使其将通知发送到其他渠道，例如电子邮件、PagerDuty、Microsoft Teams 或其他通知服务，只需在配置文件中指定相应的配置即可。 `alertmanager.yaml`文件。

以下示例配置 Alertmanager 向 Slack 频道发送通知。要根据您的环境自定义此示例，请替换以下值： `api_url`密钥包含您环境中使用的 Slack webhook URL：

alertmanager.yaml：向 **Slack** 频道发送警报

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

生成**Trident Protect** 支持包

Trident Protect 使管理员能够生成包含对NetApp支持有用的信息的捆绑包，包括有关受管理集群和应用程序的日志、指标和拓扑信息。如果您已连接到互联网，则可以使用自定义资源 (CR) 文件将支持包上传到NetApp支持站点 (NSS)。

使用 **CR** 创建支持包

步骤

1. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-support-bundle.yaml` ）。
2. 配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.triggerType**: (*Required*) 确定支持包是立即生成还是按计划生成。计划的数据包生成时间为世界协调时凌晨 12 点。可能值：
     - 已计划
     - 手动
   - **spec.uploadEnabled**: (可选) 控制生成支持包后是否应将其上传到NetApp支持站点。如果未指定，则默认为 `false`。可能值：
     - true
     - false（默认值）
   - **spec.dataWindowStart**: (可选) RFC 3339 格式的日期字符串，指定支持包中包含的数据窗口应开始的日期和时间。如果未指定，则默认为 24 小时前。您最早可以指定的日期范围是 7 天前。

     YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. 填写完之后 `trident-protect-support-bundle.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-support-bundle.yaml -n trident-
protect
```

使用 **CLI** 创建支持包

步骤

1. 创建支持包，将括号中的值替换为您环境中的信息。这 `trigger-type`决定捆绑包是立即创建还是由计划安排决定创建时间，并且可以是 `Manual`或者 `Scheduled`。默认设置是 `Manual`。

   例如：

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

监视和检索支持包

使用任一方法创建支持包后，您可以监视其生成进度并将其检索到本地系统。

步骤

1. 等待 `status.generationState`到达 `Completed`状态。您可以使用以下命令监控生成进度：

```
kubectl get autosupportbundle trident-protect-support-bundle -n trident-
protect
```

2. 将支持包检索到您的本地系统。从已完成的AutoSupport包中获取复制命令：

```
kubectl describe autosupportbundle trident-protect-support-bundle -n
trident-protect
```

找到 `kubectl cp`从输出中读取命令并运行它，将目标参数替换为您首选的本地目录。

升级**Trident**保护

您可以将Trident Protect 升级到最新版本，以享受新功能或修复错误。

> 从 24.10 版本升级时，升级期间运行的快照可能会失败。此故障不会阻止将来创建快照，无论是手动创建还是计划创建。如果在升级过程中快照失败，您可以手动创建一个新的快照，以确保您的应用程序受到保护。
>
> 为避免潜在的故障，您可以在升级前禁用所有快照计划，并在升级后重新启用它们。但是，这会导致在升级期间错过任何计划的快照。

要升级Trident Protect，请执行以下步骤。

步骤

1. 更新Trident Helm 仓库：

```
helm repo update
```

2. 升级Trident Protect CRD：

如果您是从 25.06 之前的版本升级，则需要执行此步骤，因为 CRD 现在已包含在Trident Protect Helm 图表中。

a. 运行此命令以将 CRD 的管理权从 `trident-protect-crds`到 `trident-protect`：

```
kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
{"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
```

b. 运行此命令以删除 Helm 密钥 `trident-protect-crds`图表：

不要卸载 `trident-protect-crds`使用 Helm 构建图表可能会删除您的 CRD 和任何相关数据。

```
kubectl delete secret -n trident-protect -l name=trident-protect-
crds,owner=helm
```

3. 升级Trident保护：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2506.0 --namespace trident-protect
```

管理和保护应用程序

使用**Trident Protect AppVault** 对象来管理存储桶。

Trident Protect 的存储桶自定义资源 (CR) 被称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含存储桶在保护操作（例如备份、快照、恢复操作和SnapMirror复制）中使用的必要配置。只有管理员才能创建应用保险库。

在应用程序上执行数据保护操作时，您需要手动或从命令行创建 AppVault CR。AppVaultCR 特定于您的环境，您可以使用本页上的示例作为创建 AppVault CR 的指南。

确保 AppVault CR 位于安装了Trident Protect 的集群上。如果 AppVault CR 不存在或您无法访问，命令行会显示错误。

配置 **AppVault** 身份验证和密码

在创建 AppVault CR 之前，请确保您选择的 AppVault 和数据移动器可以向提供商和任何相关资源进行身份验证。

数据迁移器存储库密码

当您使用 CR 或Trident Protect CLI 插件创建 AppVault 对象时，您可以为 Restic 和 Kopia 加密指定带有自定义密码的 Kubernetes 密钥。如果您不指定密钥， Trident Protect 将使用默认密码。

- 手动创建 AppVault CR 时，请使用 **spec.dataMoverPasswordSecretRef** 字段指定密钥。
- 使用Trident Protect CLI 创建 AppVault 对象时，请使用 `--data-mover-password-secret-ref`用于指定密钥的参数。

创建数据迁移存储库密码密钥

请参考以下示例创建密码密钥。创建 AppVault 对象时，您可以指示Trident Protect 使用此密钥向数据移动器存储库进行身份验证。

- 根据您使用的数据传输工具，您只需输入该数据传输工具对应的密码即可。例如，如果您正在使用 Restic，并且将来不打算使用 Kopia，则在创建密钥时，您可以只包含 Restic 密码。
- 请将密码保存在安全的地方。您将需要它来还原同一集群或其他集群上的数据。如果集群或`trident-protect`命名空间已被删除，没有密码您将无法恢复备份或快照。

使用 **CR**

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

使用 **CLI**

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

S3 兼容存储 **IAM** 权限

当您访问与 S3 兼容的存储（例如 Amazon S3、通用 S3）时， "StorageGrid S3" ， 或者 "ONTAP S3"使用Trident Protect 时，您需要确保提供的用户凭据具有访问存储桶的必要权限。以下是授予使用Trident Protect 进行访问所需的最低权限的策略示例。您可以将此策略应用于管理 S3 兼容存储桶策略的用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

有关 Amazon S3 策略的更多信息，请参阅以下示例： "Amazon S3 文档" 。

用于 **Amazon S3**（**AWS**）身份验证的 **EKS Pod Identity**

Trident Protect 支持 Kopia 数据移动器操作的 EKS Pod Identity。此功能可实现对 S3 存储桶的安全访问，而无需将 AWS 凭证存储在 Kubernetes 机密中。

EKS Pod Identity 与**Trident Protect** 的要求

在将 EKS Pod Identity 与Trident Protect 结合使用之前，请确保以下事项：

- 您的 EKS 集群已启用 Pod Identity。
- 您已创建具有必要的 S3 存储桶权限的 IAM 角色。欲了解更多信息，请参阅 "S3 兼容存储 IAM 权限"。
- IAM 角色与以下Trident Protect 服务帐户关联：
  - `<trident-protect>-controller-manager`
  - `<trident-protect>-resource-backup`
  - `<trident-protect>-resource-restore`
  - `<trident-protect>-resource-delete`

有关启用 Pod 身份并将 IAM 角色与服务帐户关联的详细说明，请参阅 "AWS EKS Pod Identity 文档"。

AppVault 配置 使用 EKS Pod Identity 时，请配置您的 AppVault CR。 `useIAM: true`使用标志而不是显式凭据：

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: eks-protect-vault
  namespace: trident-protect
spec:
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-aws
      endpoint: s3.example.com
      useIAM: true
```

适用于云提供商的 **AppVault** 密钥生成示例

定义 AppVault CR 时，您需要包含凭证以访问提供商托管的资源，除非您使用 IAM 身份验证。如何生成凭证密钥将根据提供商的不同而有所不同。以下是几个提供商的命令行密钥生成示例。您可以使用以下示例为每个云提供商的凭证创建密钥。

Google Cloud

```
kubectl create secret generic <secret-name> \
--from-file=credentials=<mycreds-file.json> \
-n trident-protect
```

亚马逊 S3 (AWS)

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<amazon-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

Microsoft Azure

```
kubectl create secret generic <secret-name> \
--from-literal=accountKey=<secret-name> \
-n trident-protect
```

通用S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<generic-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

ONTAP S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<ontap-s3-trident-protect-src-bucket
-secret> \
-n trident-protect
```

StorageGrid S3

```
kubectl create secret generic <secret-name> \
--from-literal=accessKeyID=<objectstorage-accesskey> \
--from-literal=secretAccessKey=<storagegrid-s3-trident-protect-src
-bucket-secret> \
-n trident-protect
```

AppVault 创建示例

以下是每个提供商的 AppVault 定义示例。

AppVault CR 示例

您可以使用以下 CR 示例为每个云提供商创建 AppVault 对象。

- 您可以选择指定一个 Kubernetes secret，其中包含 Restic 和 Kopia 存储库加密的自定义密码。请参阅[数据迁移器存储库密码]了解更多信息。
- 对于 Amazon S3 (AWS) AppVault 对象，您可以选择性地指定 sessionToken，如果您使用单点登录 (SSO) 进行身份验证，这将非常有用。当您为提供程序生成密钥时，将创建此令牌。适用于云提供商的 AppVault 密钥生成示例 。
- 对于 S3 AppVault 对象，您可以选择性地使用以下方式指定出站 S3 流量的出口代理 URL：`spec.providerConfig.S3.proxyURL`钥匙。

Google Cloud

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: gcp-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GCP
  providerConfig:
    gcp:
      bucketName: trident-protect-src-bucket
      projectID: project-id
  providerCredentials:
    credentials:
      valueFromSecret:
        key: credentials
        name: gcp-trident-protect-src-bucket-secret
```

亚马逊 S3 (AWS)

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: amazon-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: AWS
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
    sessionToken:
      valueFromSecret:
        key: sessionToken
        name: s3-secret
```

对于使用 Pod Identity 和 Kopia 数据移动器的 EKS 环境，您可以移除 `providerCredentials` 部分并添加 `useIAM: true`在 `s3`改为配置。

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

通用S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGrid S3

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

使用**Trident Protect CLI** 创建 **AppVault** 的示例

您可以使用以下 CLI 命令示例为每个提供商创建 AppVault CR。

- 您可以选择指定一个 Kubernetes secret，其中包含 Restic 和 Kopia 存储库加密的自定义密码。请参阅[数据迁移器存储库密码]了解更多信息。
- 对于 S3 AppVault 对象，您可以选择性地使用以下方式指定出站 S3 流量的出口代理 URL：`--proxy-url <ip_address:port>`争论。

Google Cloud

```
tridentctl-protect create vault GCP <vault-name> \
--bucket <mybucket> \
--project <my-gcp-project> \
--secret <secret-name>/credentials \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

亚马逊 S3 (AWS)

```
tridentctl-protect create vault AWS <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

Microsoft Azure

```
tridentctl-protect create vault Azure <vault-name> \
--account <account-name> \
--bucket <bucket-name> \
--secret <secret-name> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

通用S3

```
tridentctl-protect create vault GenericS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

ONTAP S3

```
tridentctl-protect create vault OntapS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

StorageGrid S3

```
tridentctl-protect create vault StorageGridS3 <vault-name> \
--bucket <bucket-name> \
--secret  <secret-name>  \
--endpoint <s3-endpoint> \
--data-mover-password-secret-ref <my-optional-data-mover-secret> \
-n trident-protect
```

查看 **AppVault** 信息

您可以使用Trident Protect CLI 插件查看有关您在集群上创建的 AppVault 对象的信息。

步骤

1. 查看 AppVault 对象的内容：

   ```
   tridentctl-protect get appvaultcontent gcp-vault \
   --show-resources all \
   -n trident-protect
   ```

   示例输出：

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+-----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+-----------------------------
+---------------------------+
```

2. （可选）要查看每个资源的 AppVaultPath，请使用该标志 `--show-paths`。

   只有在Trident Protect helm 安装中指定了集群名称时，表格第一列中的集群名称才可用。例如： `--set clusterName=production1` 。

移除 **AppVault**

您可以随时删除 AppVault 对象。

> 不要移除 `finalizers`在删除 AppVault 对象之前，请在 AppVault CR 中输入密钥。如果这样做，可能会导致 AppVault 存储桶中残留数据，集群中出现孤立资源。

开始之前

请确保您已删除要删除的 AppVault 使用的所有快照和备份 CR。

使用 **Kubernetes CLI** 删除 **AppVault**

1. 移除 AppVault 对象，并替换 `appvault-name`要删除的 AppVault 对象的名称：

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

使用**Trident Protect CLI** 删除 **AppVault**

1. 移除 AppVault 对象，并替换 `appvault-name`要删除的 AppVault 对象的名称：

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

使用**Trident Protect** 定义管理应用程序

您可以通过创建应用程序 CR 和关联的 AppVault CR 来定义要使用Trident Protect 管理的应用程序。

创建 **AppVault CR**

您需要创建一个 AppVault CR，该 CR 将在对应用程序执行数据保护操作时使用，并且 AppVault CR 需要位于安装了Trident Protect 的集群上。AppVault CR 是针对您的特定环境的；有关 AppVault CR 的示例，请参阅："AppVault 自定义资源。"

定义应用程序

您需要定义要使用Trident Protect 管理的每个应用程序。您可以通过手动创建应用程序 CR 或使用Trident Protect CLI 来定义要管理的应用程序。

使用 **CR** 添加应用程序

步骤

1. 创建目标应用程序 CR 文件：
   a. 创建自定义资源 (CR) 文件并将其命名为（例如， `maria-app.yaml` ）。
   b. 配置以下属性：
      - **metadata.name**: (必填) 应用程序自定义资源的名称。请注意您选择的名称，因为保护操作所需的其他 CR 文件会引用此值。
      - **spec.includedNamespaces**: (必需) 使用命名空间和标签选择器来指定应用程序使用的命名空间和资源。应用程序命名空间必须包含在此列表中。标签选择器是可选的，可用于筛选每个指定命名空间内的资源。
      - **spec.includedClusterScopedResources**: (可选) 使用此属性指定要包含在应用程序定义中的集群范围资源。此属性允许您根据资源的组、版本、种类和标签来选择这些资源。
        - **groupVersionKind**: (必需) 指定集群范围资源的 API 组、版本和类型。
        - **labelSelector**: (可选) 根据标签筛选集群范围的资源。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (可选) 此注解仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。指定此应用程序在快照期间是否可以写入文件系统。如果设置为 true，应用程序将忽略全局设置，并且可以在快照期间写入文件系统。如果设置为 false，应用程序将忽略全局设置，并且在快照期间文件系统将被冻结。如果指定了注解，但应用程序定义中没有虚拟机，则忽略该注解。如未特别说明，则申请流程如下："全球Trident Protect 冷冻设置" 。

如果需要在应用程序创建后应用此注解，可以使用以下命令：

```
kubectl annotate application -n <application CR namespace> <application CR name> protect.trident.netapp.io/skip-vm-freeze="true"
```

+
YAML 示例：

+

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  annotations:
    protect.trident.netapp.io/skip-vm-freeze: "false"
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: namespace-1
      labelSelector:
        matchLabels:
          app: example-app
    - namespace: namespace-2
      labelSelector:
        matchLabels:
          app: another-example-app
  includedClusterScopedResources:
    - groupVersionKind:
        group: rbac.authorization.k8s.io
        kind: ClusterRole
        version: v1
      labelSelector:
        matchLabels:
          mylabel: test
```

1. （可选）添加筛选条件，包含或排除带有特定标签的资源：

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：

     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。

       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。

- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

> 当两者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`先运行，然后 `labelSelector`应用于生成的资源。

例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

2. 创建与您的环境相匹配的应用程序变更请求后，应用该变更请求。例如：

```
kubectl apply -f maria-app.yaml
```

步骤

1. 使用以下示例之一创建并应用应用程序定义，将括号中的值替换为您环境中的信息。您可以使用逗号分隔的列表，并在示例中所示的参数中，将命名空间和资源包含在应用程序定义中。

   创建应用时，您可以选择使用注解来指定应用在快照期间是否可以写入文件系统。这仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。如果您将注释设置为 `true`该应用程序忽略全局设置，可以在快照期间写入文件系统。如果你把它设置为 `false`该应用程序忽略全局设置，导致文件系统在快照期间冻结。如果使用了注解，但应用程序定义中没有虚拟机，则该注解将被忽略。如果您不使用注解，应用程序将遵循以下规则："全球Trident Protect 冷冻设置" 。

   在使用 CLI 创建应用程序时，要指定注解，可以使用以下方法： `--annotation`旗帜。

   - 创建应用程序并使用文件系统冻结行为的全局设置：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace>
```

- 创建应用程序并配置本地应用程序设置以控制文件系统冻结行为：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --annotation protect.trident.netapp.io/skip-vm-freeze
=<"true"|"false">
```

您可以使用 `--resource-filter-include`和 `--resource-filter-exclude`用于根据以下条件包含或排除资源的标志 `resourceSelectionCriteria`例如组、种类、版本、标签、名称和命名空间，如下例所示：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-namespace>
--resource-filter-include
'[{"Group":"apps","Kind":"Deployment","Version":"v1","Names":["my-
deployment"],"Namespaces":["my-
namespace"],"LabelSelectors":["app=my-app"]}]'
```

使用Trident Protect 保护应用程序

您可以使用自动保护策略或临时保护策略，通过拍摄快照和备份来保护Trident Protect 管理的所有应用程序。

您可以配置Trident Protect 在数据保护操作期间冻结和解冻文件系统。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

创建按需快照

您可以随时创建按需快照。

如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

使用 **CR** 创建快照

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: 要创建快照的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储快照内容（元数据）的 AppVault 的名称。
   - **spec.reclaimPolicy**: (可选) 定义当快照 CR 被删除时，快照的 AppArchive 会发生什么情况。这意味着即使设置为 `Retain`快照将被删除。有效选项：
     - `Retain`（默认）
     - `Delete`

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Snapshot
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  reclaimPolicy: Delete
```

3. 填写完之后 `trident-protect-snapshot-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-cr.yaml
```

使用 **CLI** 创建快照

步骤

1. 创建快照，将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

创建按需备份

您可以随时备份应用程序。

如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

开始之前

确保 AWS 会话令牌过期时间足以满足任何长时间运行的 S3 备份操作。如果在备份操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

使用 **CR** 创建备份

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储备份内容的 AppVault 的名称。
   - **spec.dataMover**: (可选) 指示要用于备份操作的备份工具的字符串。可能的值（区分大小写）：
     - `Restic`
     - `Kopia`（默认）
   - **spec.reclaimPolicy**: (可选) 定义从其声明中释放备份时会发生什么。可能值：
     - `Delete`
     - `Retain`（默认）
   - **spec.snapshotRef**: (可选): 要用作备份源的快照名称。如果未提供，则会创建并备份临时快照。
   - **metadata.annotations.protect.trident.netapp.io/full-backup** : (可选) 此注释用于指定备份是否应为非增量备份。默认情况下，所有备份都是增量的。但是，如果此注释设置为 `true`这样，备份就变成了非增量备份。如果未指定，备份将遵循默认的增量备份设置。最佳实践是定期执行完整备份，然后在两次完整备份之间执行增量备份，以最大限度地降低与恢复相关的风险。

     YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
  annotations:
    protect.trident.netapp.io/full-backup: "true"
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. 填写完之后 `trident-protect-backup-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-backup-cr.yaml
```

使用命令行创建备份

步骤

1. 创建备份，将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up> --data-mover
<Kopia_or_Restic> -n <application_namespace>
```

您也可以选择使用 `--full-backup`用于指定备份是否应为非增量备份的标志。默认情况下，所有备份都是增量的。使用此标志后，备份将变为非增量备份。最佳实践是定期执行完整备份，然后在两次完整备份之间执行增量备份，以最大限度地降低与恢复相关的风险。

制定数据保护计划

保护策略通过按照定义的计划创建快照、备份或两者来保护应用程序。您可以选择每小时、每天、每周和每月创建快照和备份，并可以指定要保留的副本数量。您可以使用 full-backup-rule 注释来安排非增量式完整备份。默认情况下，所有备份都是增量的。定期执行完整备份以及其间的增量备份有助于降低与恢复相关的风险。

- 您只能通过设置来创建快照计划。 `backupRetention`归零和 `snapshotRetention`取大于零的值。环境 `snapshotRetention`将快照值设为零意味着任何计划备份仍会创建快照，但这些快照是临时的，会在备份完成后立即删除。
- 如果集群范围的资源在应用程序定义中被明确引用，或者被引用到任何应用程序命名空间，则这些资源将被包含在备份、快照或克隆中。

使用变更请求 **(CR)** 创建计划。

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-schedule-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.dataMover**: (可选) 指示要用于备份操作的备份工具的字符串。可能的值（区分大小写）：
     - `Restic`
     - `Kopia`（默认）
   - **spec.applicationRef**: 要备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 应存储备份内容的 AppVault 的名称。
   - **spec.backupRetention**：要保留的备份数量。零表示不应创建备份（仅快照）。
   - **spec.snapshotRetention**：要保留的快照数量。零表示不创建任何快照。
   - **spec.granularity**: 计划运行的频率。可能的值，以及相应的必填字段：
     - `Hourly`（需要您指定） `spec.minute` ）
     - `Daily`（需要您指定） `spec.minute`和 `spec.hour`）
     - `Weekly`（需要您指定） `spec.minute, spec.hour` ， 和 `spec.dayOfWeek`）
     - `Monthly`（需要您指定） `spec.minute, spec.hour` ， 和 `spec.dayOfMonth`）
     - `Custom`
   - **spec.dayOfMonth**：（可选）计划应运行的月份日期（1 - 31）。如果粒度设置为，则此字段为必填项。 `Monthly` 。该值必须以字符串形式提供。
   - **spec.dayOfWeek**：（可选）计划应运行的星期几（0 - 7）。值 0 或 7 表示星期日。如果粒度设置为，则此字段为必填项。 `Weekly` 。该值必须以字符串形式提供。
   - **spec.hour**：（可选）计划应运行的小时数（0 - 23）。如果粒度设置为，则此字段为必填项。 `Daily` ， `Weekly` ， 或者 `Monthly`。该值必须以字符串形式提供。
   - **spec.minute**：（可选）计划应运行的小时中的分钟数（0 - 59）。如果粒度设置为，则此字段为必填项。 `Hourly` ， `Daily` ， `Weekly` ， 或者 `Monthly`。该值必须以字符串形式提供。
   - **metadata.annotations.protect.trident.netapp.io/full-backup-rule**: (可选) 此注解用于指定安排完整备份的规则。你可以将其设置为 `always`您可以根据需要进行持续完整备份或自定义备份。例如，如果您选择按日粒度进行备份，则可以指定应进行完整备份的星期几。

     备份和快照计划的示例 YAML：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
  annotations:
    protect.trident.netapp.io/full-backup-rule: "Monday,Thursday"
spec:
  dataMover: Kopia
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "15"
  snapshotRetention: "15"
  granularity: Daily
  hour: "0"
  minute: "0"
```

仅快照计划的示例 YAML：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

3. 填写完之后 `trident-protect-schedule-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

使用 **CLI** 创建计划任务

步骤

1. 创建保护计划，将括号中的值替换为您环境中的信息。例如：

您可以使用 `tridentctl-protect create schedule --help`查看此命令的详细帮助信息。

```
tridentctl-protect create schedule <my_schedule_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> --backup
-retention <how_many_backups_to_retain> --data-mover
<Kopia_or_Restic> --day-of-month <day_of_month_to_run_schedule>
--day-of-week <day_of_month_to_run_schedule> --granularity
<frequency_to_run> --hour <hour_of_day_to_run> --minute
<minute_of_hour_to_run> --recurrence-rule <recurrence> --snapshot
-retention <how_many_snapshots_to_retain> -n <application_namespace>
--full-backup-rule <string>
```

您可以设置 `--full-backup-rule`标记 `always`您可以根据需要进行持续完整备份或自定义备份。例如，如果您选择按天粒度进行备份，则可以指定应在哪些工作日进行完整备份。例如，使用 `--full-backup-rule "Monday,Thursday"`安排每周一和周四进行全面备份。

对于仅快照计划，请设置 `--backup-retention 0`并指定一个大于 `0` 的值 `--snapshot-retention`。

删除快照

删除不再需要的已安排或按需快照。

步骤

1. 删除与快照关联的快照 CR：

```
kubectl delete snapshot <snapshot_name> -n my-app-namespace
```

删除备份

删除不再需要的计划备份或按需备份。

确保回收策略设置为 `Delete`从对象存储中删除所有备份数据。该策略的默认设置为： `Retain`避免意外数据丢失。如果政策不改变 `Delete`备份数据将保留在对象存储中，需要手动删除。

步骤

1. 删除与备份关联的备份 CR：

```
kubectl delete backup <backup_name> -n my-app-namespace
```

检查备份操作的状态

您可以使用命令行来检查正在进行、已完成或已失败的备份操作的状态。

步骤

1. 使用以下命令检索备份操作的状态，将方括号中的值替换为您环境中的信息：

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

启用 **Azure NetApp** 文件 **(ANF)** 操作的备份和还原

如果您已安装Trident Protect，则可以为使用 azure-netapp-files 存储类且在Trident 24.06 之前创建的存储后端启用节省空间的备份和还原功能。此功能适用于 NFSv4 卷，并且不会占用容量池中的额外空间。

开始之前

确保以下事项：

- 您已安装Trident Protect。
- 您已在Trident Protect中定义了一个应用程序。在您完成此步骤之前，此应用程序的保护功能将受到限制。
- 你有 `azure-netapp-files`已选为存储后端的默认存储类。

展开查看配置步骤

1. 如果 ANF 卷是在升级到Trident 24.10 之前创建的，请在Trident中执行以下操作：

   a. 为每个基于 azure-netapp-files 且与应用程序关联的 PV 启用快照目录：

   ```
   tridentctl update volume <pv name> --snapshot-dir=true -n trident
   ```

   b. 确认已为每个关联的 PV 启用快照目录：

   ```
   tridentctl get volume <pv name> -n trident -o yaml | grep
   snapshotDir
   ```

   回复：

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未启用快照目录，则选择常规备份功能，该功能在备份过程中会暂时占用容量池中的空间。在这种情况下，请确保容量池中有足够的空间来创建与被备份卷大小相同的临时卷。

结果

该应用程序已准备好使用Trident Protect 进行备份和恢复。每个 PVC 也可供其他应用程序用于备份和恢复。

恢复应用程序

使用**Trident Protect** 恢复应用程序

您可以使用Trident Protect 从快照或备份中恢复您的应用程序。将应用程序恢复到同一集群时，从现有快照恢复速度会更快。

- 恢复应用程序时，为该应用程序配置的所有执行钩子都会随应用程序一起恢复。如果存在恢复后执行钩子，它将作为恢复操作的一部分自动运行。
- 支持从备份恢复到不同的命名空间或恢复到原始命名空间（适用于 qtree 卷）。但是，对于 qtree 卷，不支持从快照恢复到不同的命名空间或恢复到原始命名空间。
- 您可以使用高级设置来自定义恢复操作。欲了解更多信息，请参阅 "使用高级Trident Protect 恢复设置"。

从备份还原到不同的命名空间

当您使用 BackupRestore CR 将备份还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或

者制定保护计划。

> 将备份还原到具有现有资源的不同命名空间不会更改与备份中资源同名的任何资源。要恢复备份中的所有资源，要么删除并重新创建目标命名空间，要么将备份恢复到新的命名空间。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

> 当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "科皮亚文件"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-restore-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （可选）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   > Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。

- **resourceMatchers[].version**: (可选) 要筛选的资源版本。
- **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 填写完之后 `trident-protect-backup-restore-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用 CLI

步骤

1. 将备份还原到不同的命名空间，并将括号中的值替换为您环境中的信息。这 `namespace-mapping`该参数使用冒号分隔的命名空间，将源命名空间映射到正确的目标命名空间，格式如下：`source1:dest1,source2:dest2` 。例如：

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

从备份恢复到原始命名空间

您可以随时将备份恢复到原始命名空间。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

> 当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "科皮亚文件"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-ipr-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。

     例如：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appArchivePath: my-backup-path
  appVaultRef: appvault-name
```

3. （可选）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名

称。

- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 填写完之后 `trident-protect-backup-ipr-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

使用 **CLI**

步骤

1. 将备份恢复到原始命名空间，并将括号中的值替换为您环境中的信息。这 `backup`参数使用命名空间和备份名称，格式如下 `<namespace>/<name>`。例如：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

如果原始集群出现问题，您可以将备份还原到其他集群。

> 当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "科皮亚文件"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

开始之前

确保满足以下先决条件：

- 目标集群已安装Trident Protect。
- 目标集群可以访问与源集群相同的 AppVault 的存储桶路径，备份就存储在该存储桶中。
- 确保在运行 AppVault CR 时，本地环境可以连接到 AppVault CR 中定义的对象存储桶。 `tridentctl-protect get appvaultcontent`命令。如果网络限制阻止访问，请改为从目标集群上的 pod 内运行Trident Protect CLI。
- 确保 AWS 会话令牌的过期时间足以满足任何长时间运行的恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。
  - 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
  - 请参阅 "AWS 文档"有关 AWS 资源凭证的更多信息。

步骤

1. 使用Trident Protect CLI 插件检查目标集群上 AppVault CR 的可用性：

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

> 确保用于应用程序还原的命名空间存在于目标集群上。

2. 查看目标集群中可用 AppVault 的备份内容：

```
tridentctl-protect get appvaultcontent <appvault_name> \
--show-resources backup \
--show-paths \
--context <destination_cluster_name>
```

运行此命令将显示 AppVault 中可用的备份，包括其来源集群、相应的应用程序名称、时间戳和归档路径。

示例输出：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |        TIMESTAMP
|    PATH     |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
```

3. 使用 AppVault 名称和归档路径将应用程序还原到目标集群：

使用 **CR**

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-backup-restore-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     如果 BackupRestore CR 不可用，您可以使用步骤 2 中提到的命令来查看备份内容。

   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     例如：

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-backup-path
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. 填写完之后 `trident-protect-backup-restore-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用 **CLI**

1. 使用以下命令恢复应用程序，将括号中的值替换为您环境中的信息。命名空间映射参数使用冒号分隔的命名空间，将源命名空间映射到正确的目标命名空间，格式为 source1:dest1,source2:dest2。例如：

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--context <destination_cluster_name> \
-n <application_namespace>
```

从快照还原到不同的命名空间

您可以使用自定义资源 (CR) 文件从快照恢复数据，恢复到不同的命名空间或原始源命名空间。当您使用 SnapshotRestore CR 将快照还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或者制定保护计划。

SnapshotRestore 支持 `spec.storageClassMapping`属性，但仅当源存储类和目标存储类使用相同的存储后端时才有效。如果您尝试恢复到 `StorageClass`如果使用不同的存储后端，则恢复操作将失败。

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内存储快照内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get snapshots <SNAPHOT_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-app-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. （可选）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。

- **resourceMatchers[].version**: (可选) 要筛选的资源版本。
- **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
- **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 填写完之后 `trident-protect-snapshot-restore-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用 CLI

步骤

1. 将快照还原到不同的命名空间，并将括号中的值替换为您环境中的信息。
   - 这 `snapshot`参数使用命名空间和快照名称，格式如下 `<namespace>/<name>`。
   - 这 `namespace-mapping`该参数使用冒号分隔的命名空间，将源命名空间映射到正确的目标命名空间，格式如下： `source1:dest1,source2:dest2` 。

     例如：

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

从快照恢复到原始命名空间

您可以随时将快照恢复到原始命名空间。

如果您的应用程序使用多个命名空间，并且这些命名空间具有同名的PVC，则快照
地和新命名空间）将无法正常工作。所有还原的卷将具有相同的数据，而不是每个
确数据。使用备份还原而不是快照还原，或升级到修复此问题的26.02版或更高版

开始之前

确保 AWS 会话令牌的过期时间足以满足任何长时间运行的 S3 恢复操作。如果在恢复操作期间令牌过期，则操作可能会失败。

- 请参阅 "AWS API 文档"有关检查当前会话令牌过期时间的更多信息。
- 请参阅 "AWS IAM 文档"有关 AWS 资源凭证的更多信息。

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-ipr-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内存储快照内容的路径。您可以使用以下命令查找此路径：

```
kubectl get snapshots <SNAPSHOT_NAME> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

```
---
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotInplaceRestore
metadata:
  name: my-cr-name
  namespace: my-app-namespace
spec:
  appVaultRef: appvault-name
    appArchivePath: my-snapshot-path
```

3. （可选）如果您只需要选择应用程序中的某些资源进行恢复，请添加筛选条件，以包含或排除带有特定标签的资源：

> Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `Include`或者 `Exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 填写完之后 `trident-protect-snapshot-ipr-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

使用 **CLI**

步骤

1. 将快照恢复到原始命名空间，并将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <snapshot_to_restore> \
-n <application_namespace>
```

检查还原操作的状态

您可以使用命令行来检查正在进行、已完成或已失败的还原操作的状态。

步骤

1. 使用以下命令检索恢复操作的状态，将方括号中的值替换为您环境中的信息：

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

使用高级**Trident Protect** 恢复设置

您可以使用高级设置（例如注释、命名空间设置和存储选项）自定义恢复操作，以满足您的特定要求。

恢复和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释相匹配。源命名空间中不存在的目标命名空间中的标签或注释将被添加，并且任何已存在的标签或注释都将被覆盖以匹配源命名空间中的值。仅存在于目标命名空间中的标签或注释保持不变。

> 如果您使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。欲了解更多信息，请参阅 "OpenShift 安全上下文约束文档"。

您可以通过设置 Kubernetes 环境变量来防止目标命名空间中的特定注解被覆盖。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在执行恢复或故障转移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

> 执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置AutoSupport和命名空间过滤选项"。

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例展示了源命名空间和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作前后目标命名空间的状态，以及目标命名空间中的注释和标签是如何组合或覆盖的。

在恢复或故障转移操作之前

下表说明了恢复或故障转移操作之前示例源命名空间和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 环境=生产 • 合规性=HIPAA • 名称=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

恢复操作后

下表说明了恢复或故障转移操作后示例目标命名空间的状态。有些键已被添加，有些键已被覆盖，并且 `name` 标签已更新，以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名称=ns-2 • 合规性=HIPAA • 环境=生产 • 角色=数据库

支持的字段

本节介绍可用于恢复操作的其他字段。

存储类别映射

这 `spec.storageClassMapping`属性定义了从源应用程序中存在的存储类到目标集群上新存储类的映射。您可以在具有不同存储类别的集群之间迁移应用程序时或更改 BackupRestore 操作的存储后端时使用此功能。

例子：

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

支持的注释

本节列出了系统中用于配置各种行为的支持注解。如果用户没有明确设置注释，系统将使用默认值。

标注	类型	描述	默认值
protect.trident.netapp.io/data-mover-timeout-sec	string	数据移动器操作允许停止的最长时间（以秒为单位）。	“300”
protect.trident.netapp.io/kopia-content-cache-size-limit-mb	string	Kopia 内容缓存的最大大小限制（以兆字节为单位）。	“1000”

使用NetApp SnapMirror和Trident Protect 复制应用程序

使用Trident Protect，您可以利用NetApp SnapMirror技术的异步复制功能，将数据和应用程序更改从一个存储后端复制到另一个存储后端，无论是在同一集群内还是在不同集群之间。

恢复和故障转移操作期间的命名空间注释和标签

在恢复和故障转移操作期间，目标命名空间中的标签和注释将与源命名空间中的标签和注释相匹配。源命名空间中不存在的目标命名空间中的标签或注释将被添加，并且任何已存在的标签或注释都将被覆盖以匹配源命名空间中的值。仅存在于目标命名空间中的标签或注释保持不变。

> 如果您使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。欲了解更多信息，请参阅 "OpenShift 安全上下文约束文档"。

您可以通过设置 Kubernetes 环境变量来防止目标命名空间中的特定注解被覆盖。`RESTORE_SKIP_NAMESPACE_ANNOTATIONS`在执行恢复或故障转移操作之前。例如：

```
helm upgrade trident-protect --set
restoreSkipNamespaceAnnotations=<annotation_key_to_skip_1>,<annotation_key
_to_skip_2> --reuse-values
```

> 执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置AutoSupport和命名空间过滤选项"。

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例展示了源命名空间和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作前后目标命名空间的状态，以及目标命名空间中的注释和标签是如何组合或覆盖的。

在恢复或故障转移操作之前

下表说明了恢复或故障转移操作之前示例源命名空间和目标命名空间的状态：

命名空间	标注	标签
命名空间 ns-1（源）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true"	• 环境=生产 • 合规性=HIPAA • 名称=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

恢复操作后

下表说明了恢复或故障转移操作后示例目标命名空间的状态。有些键已被添加，有些键已被覆盖，并且 `name` 标签已更新，以匹配目标命名空间：

命名空间	标注	标签
命名空间 ns-2（目标）	• annotation.one/key: "updatedvalue" • annotation.two/key: "true" • annotation.three/key: "false"	• 名称=ns-2 • 合规性=HIPAA • 环境=生产 • 角色=数据库

您可以配置Trident Protect 在数据保护操作期间冻结和解冻文件系统。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

故障转移和反向操作期间的执行钩子

使用 AppMirror 关系保护应用程序时，在故障转移和反向操作期间，您应该注意与执行钩子相关的特定行为。

- 故障转移期间，执行钩子会自动从源集群复制到目标集群。您无需手动重新创建它们。故障转移后，应用程序上会存在执行钩子，这些钩子将在任何相关操作期间执行。
- 在反向同步或反向重同步期间，应用程序上任何现有的执行钩子都会被移除。当源应用程序变为目标应用程序时，这些执行钩子将不再有效，并会被删除以防止其执行。

要了解有关执行钩子的更多信息，请参阅"管理Trident Protect 执行钩子"。

建立复制关系

建立复制关系涉及以下步骤：

- 选择Trident Protect 拍摄应用程序快照的频率（包括应用程序的 Kubernetes 资源以及应用程序每个卷的卷快照）。
- 选择复制计划（包括 Kubernetes 资源以及持久卷数据）

- 设置拍摄快照的时间

步骤

1. 在源集群上，为源应用程序创建一个 AppVault。根据您的存储提供商，修改示例中的内容。"AppVault 自定义资源"为了适应您的环境：

使用 **CR** 创建 **AppVault**

a. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-appvault-primary-source.yaml` ）。

b. 配置以下属性：

- **metadata.name**: (必填) AppVault 自定义资源的名称。请记下您选择的名称，因为复制关系所需的其他 CR 文件会引用此值。
- **spec.providerConfig**: (必需) 存储使用指定提供程序访问 AppVault 所需的配置。选择存储桶名称以及提供商所需的其他任何详细信息。请记下您选择的值，因为复制关系所需的其他 CR 文件会引用这些值。请参阅"AppVault 自定义资源"例如，AppVault CR 与其他提供商的合作案例。
- **spec.providerCredentials**: (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
  - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示凭据值应来自密钥。
    - **key**: (必填) 要从中选择的有效密钥。
    - **name**: (必填) 包含此字段值的密钥的名称。必须位于同一命名空间中。
  - **spec.providerCredentials.secretAccessKey**: (必需) 用于访问提供程序的访问密钥。 名称 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。
- **spec.providerType**: (必需) 确定备份的提供方；例如， NetApp ONTAP S3、通用 S3 、Google Cloud 或 Microsoft Azure。可能值：
  - AWS
  - 蔚蓝
  - 通用控制协议
  - 通用-s3
  - ontap-s3
  - 存储网格-s3

c. 填写完之后 `trident-protect-appvault-primary-source.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

使用 **CLI** 创建 **AppVault**

a. 创建 AppVault，并将括号中的值替换为您环境中的信息：

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. 在源集群上，创建源应用程序 CR：

使用 **CR** 创建源应用程序

a. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-app-source.yaml` ）。

b. 配置以下属性：

- **metadata.name**: (必填) 应用程序自定义资源的名称。请记下您选择的名称，因为复制关系所需的其他 CR 文件会引用此值。
- **spec.includedNamespaces**: (必需) 命名空间及其关联标签的数组。使用命名空间名称，并可选择使用标签缩小命名空间的范围，以指定此处列出的命名空间中存在的资源。应用程序命名空间必须包含在此数组中。

  YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-app-name
  namespace: my-app-namespace
spec:
  includedNamespaces:
    - namespace: my-app-namespace
      labelSelector: {}
```

c. 填写完之后 `trident-protect-app-source.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-
namespace
```

使用 **CLI** 创建源应用程序

a. 创建源应用程序。例如：

```
tridentctl-protect create app <my-app-name> --namespaces
<namespaces-to-be-included> -n <my-app-namespace>
```

3. （可选）在源集群上，对源应用程序进行快照。此快照将用作目标集群上应用程序的基础。如果跳过此步骤，则需要等待下一次计划快照运行，以便获得最新的快照。要创建按需快照，请参阅 "创建按需快照"。

4. 在源集群上，创建复制计划 CR：

除了下面提供的计划之外，建议创建一个单独的每日快照计划，保留期为 7 天，以在对等ONTAP集群之间保持共同的快照。这样可以确保快照最多保留 7 天，但保留期限可以根据用户需求进行自定义。

如果发生故障转移，系统可以使用这些快照最多 7 天进行逆向操作。这种方法使得逆向过程更快、更高效，因为只会传输自上次快照以来所做的更改，而不是所有数据。

如果应用程序的现有计划已经满足所需的保留要求，则无需制定其他计划。

使用 **CR** 创建复制计划

a. 为源应用程序创建复制计划：

i. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-schedule.yaml` ）。

ii. 配置以下属性：

- **metadata.name**: (必填) 计划自定义资源的名称。
- **spec.appVaultRef**: (必需) 此值必须与源应用程序的 AppVault 的 metadata.name 字段匹配。
- **spec.applicationRef**: (必需) 此值必须与源应用程序 CR 的 metadata.name 字段匹配。
- **spec.backupRetention**: (*Required*) 此字段为必填项，其值必须设置为 0。
- **spec.enabled**：必须设置为 true。
- **spec.granularity**：必须设置为 `Custom`。
- **spec.recurrenceRule**: 定义 UTC 时间的开始日期和重复间隔。
- **spec.snapshotRetention**：必须设置为 2。

YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule
  namespace: my-app-namespace
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

i. 填写完之后 `trident-protect-schedule.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-
namespace
```

使用 **CLI** 创建复制计划

a. 创建复制计划，并将括号中的值替换为您环境中的信息：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

例子：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule  "DTSTART:20220101T000200Z
\nRRULE:FREQ=MINUTELY;INTERVAL=5" --snapshot-retention 2 -n
<my_app_namespace>
```

5. 在目标集群上，创建一个与在源集群上应用的 AppVault CR 完全相同的源应用程序 AppVault CR，并将其命名为（例如， `trident-protect-appvault-primary-destination.yaml` ）。
6. 应用 CR：

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
trident-protect
```

7. 在目标集群上为目标应用程序创建目标 AppVault CR。根据您的存储提供商，修改示例中的内容。"AppVault 自定义资源"为了适应您的环境：
   a. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-appvault-secondary-destination.yaml` ）。
   b. 配置以下属性：
      - **metadata.name**: (必填) AppVault 自定义资源的名称。请记下您选择的名称，因为复制关系所需的其他 CR 文件会引用此值。
      - **spec.providerConfig**: (必需) 存储使用指定提供程序访问 AppVault 所需的配置。选择一个 `bucketName`以及其他任何您需要提供给服务提供商的详细信息。请记下您选择的值，因为复制关系所需的其他 CR 文件会引用这些值。请参阅"AppVault 自定义资源"例如，AppVault CR 与其他提供商的合作案例。
      - **spec.providerCredentials**: (*Required*) 存储使用指定提供程序访问 AppVault 所需的任何凭据的引用。
        - **spec.providerCredentials.valueFromSecret**: (*Required*) 表示凭据值应来自密钥。
          - **key**: (必填) 要从中选择的有效密钥。
          - **name**: (必填) 包含此字段值的密钥的名称。必须位于同一命名空间中。
        - **spec.providerCredentials.secretAccessKey**: (必需) 用于访问提供程序的访问密钥。 名称 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。

- **spec.providerType**: (必需) 确定备份的提供方；例如， NetApp ONTAP S3、通用 S3、Google Cloud 或 Microsoft Azure。可能值：
  - AWS
  - 蔚蓝
  - 通用控制协议
  - 通用-s3
  - ontap-s3
  - 存储网格-s3

c. 填写完之后 `trident-protect-appvault-secondary-destination.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-appvault-secondary-destination.yaml
-n trident-protect
```

8. 在目标集群上，创建 AppMirrorRelationship CR 文件：

使用 **CR** 创建 **AppMirrorRelationship**

a. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-relationship.yaml` ）。

b. 配置以下属性：

- **metadata.name:**（必填）AppMirrorRelationship 自定义资源的名称。
- **spec.destinationAppVaultRef**: (必需) 此值必须与目标集群上目标应用程序的 AppVault 名称匹配。
- **spec.namespaceMapping**: (必需) 目标命名空间和源命名空间必须与相应应用程序 CR 中定义的应用程序命名空间匹配。
- **spec.sourceAppVaultRef**: (必需) 此值必须与源应用程序的 AppVault 名称匹配。
- **spec.sourceApplicationName**: (必需) 此值必须与您在源应用程序 CR 中定义的源应用程序的名称匹配。
- **spec.sourceApplicationUID**: （必需）此值必须与您在源应用程序 CR 中定义的源应用程序的 UID 匹配。
- **spec.storageClassName**: (可选) 选择集群上有效的存储类的名称。存储类必须链接到与源环境建立对等连接的ONTAP存储 VM。如果未提供存储类，则默认使用集群上的默认存储类。
- **spec.recurrenceRule**: 定义 UTC 时间的开始日期和重复间隔。

YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. 填写完之后 `trident-protect-relationship.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

使用 **CLI** 创建 **AppMirrorRelationship**

a. 创建并应用 AppMirrorRelationship 对象，并将括号中的值替换为您环境中的信息：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirorrelationship> --destination-app-vault
<my_vault_name> --source-app-vault <my_vault_name> --recurrence
-rule <rule> --namespace-mapping <ns_mapping> --source-app-id
<source_app_UID> --source-app <my_source_app_name> --storage
-class <storage_class_name> -n <application_namespace>
```

例子：

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. （可选）在目标集群上，检查复制关系的状态：

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

故障转移到目标集群

使用Trident Protect，您可以将复制的应用程序故障转移到目标集群。此过程会停止复制关系，并将应用程序在目标集群上联机。如果源集群上的应用程序正在运行， Trident Protect 不会停止该应用程序。

步骤

1. 在目标集群上，编辑 AppMirrorRelationship CR 文件（例如， `trident-protect-relationship.yaml` ）并将 **spec.desiredState** 的值更改为 `Promoted`。
2. 保存CR文件。

3. 应用 CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. （可选）在故障转移应用程序上创建所需的任何保护计划。
5. （可选）检查复制关系的状态：

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

重新同步失败的复制关系

重新同步操作会重新建立复制关系。执行重新同步操作后，原始源应用程序将成为正在运行的应用程序，对目标集群上正在运行的应用程序所做的任何更改都将被丢弃。

该过程会在重新建立复制之前停止目标集群上的应用程序。

故障转移期间写入目标应用程序的任何数据都将丢失。

步骤

1. （可选）在源集群上，创建源应用程序的快照。这样可以确保捕获源集群的最新更改。
2. 在目标集群上，编辑 AppMirrorRelationship CR 文件（例如， `trident-protect-relationship.yaml` ）并将 spec.desiredState 的值更改为 `Established`。
3. 保存CR文件。
4. 应用 CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. 如果您在目标集群上创建了任何保护计划来保护故障转移应用程序，请将其删除。任何残留的计划都会导致卷快照失败。

反向重新同步失败的复制关系

当您反向同步故障转移复制关系时，目标应用程序将变为源应用程序，而源应用程序将变为目标应用程序。故障转移期间对目标应用程序所做的更改将被保留。

步骤

1. 在原始目标集群上，删除 AppMirrorRelationship CR。这导致目的地变成了出发地。如果新目标集群上还有任何剩余的保护计划，请将其删除。
2. 通过将最初用于建立关系的 CR 文件应用到相反的集群来建立复制关系。
3. 确保新目标（原始源集群）配置了两个 AppVault CR。
4. 在相反的集群上建立复制关系，配置反向的值。

反向应用程序复制方向

当您反转复制方向时， Trident Protect 会将应用程序移动到目标存储后端，同时继续复制回原始源存储后端。Trident Protect 会停止源应用程序并将数据复制到目标位置，然后再故障转移到目标应用程序。

在这种情况下，你交换了源地址和目标地址。

步骤

1. 在源集群上，创建关机快照：

使用 **CR** 创建关机快照

a. 禁用源应用程序的保护策略计划。

b. 创建 ShutdownSnapshot CR 文件：

i. 创建自定义资源 (CR) 文件并将其命名为（例如， `trident-protect-shutdownsnapshot.yaml` ）。

ii. 配置以下属性：

- **metadata.name**: (必填) 自定义资源的名称。
- **spec.AppVaultRef**: (必需) 此值必须与源应用程序的 AppVault 的 metadata.name 字段匹配。
- **spec.ApplicationRef**: (必需) 此值必须与源应用程序 CR 文件的 metadata.name 字段匹配。

YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. 填写完之后 `trident-protect-shutdownsnapshot.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-namespace
```

使用 **CLI** 创建关机快照

a. 创建关机快照，将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot> -n <application_namespace>
```

2. 在源集群上，关机快照完成后，获取关机快照的状态：

```
kubectl get shutdownsnapshot -n my-app-namespace
<shutdown_snapshot_name> -o yaml
```

3. 在源集群上，使用以下命令查找 **shutdownsnapshot.status.appArchivePath** 的值，并记录文件路径的最后一部分（也称为基本名称；这将是最后一个斜杠之后的所有内容）：

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o
jsonpath='{.status.appArchivePath}'
```

4. 从新的目标集群到新的源集群执行故障转移，并进行以下更改：

> 在故障转移流程的第 2 步中，包括： `spec.promotedSnapshot`在 AppMirrorRelationship CR 文件中，将该字段的值设置为您在上面的步骤 3 中记录的基本名称。

5. 执行反向重新同步步骤[反向重新同步失败的复制关系]。
6. 在新源集群上启用保护计划。

结果

由于反向复制，会发生以下操作：

- 对原始源应用程序的 Kubernetes 资源进行快照。
- 通过删除应用程序的 Kubernetes 资源（保留 PVC 和 PV），优雅地停止原始源应用程序的 pod。
- 在 pod 关闭后，会对应用程序的卷进行快照并进行复制。
- SnapMirror关系已断开，目标卷已准备好进行读/写操作。
- 该应用程序的 Kubernetes 资源是从关闭前的快照中恢复的，使用的是在原始源应用程序关闭后复制的卷数据。
- 复制过程以相反的方向重新建立。

将应用程序故障恢复到原始源集群

使用Trident Protect，您可以通过以下步骤序列在故障转移操作后实现“故障恢复”。在此恢复原始复制方向的工作流程中， Trident Protect 会将任何应用程序更改复制（重新同步）回原始源应用程序，然后再反转复制方向。

该过程从已完成故障转移至目标位置的关系开始，并涉及以下步骤：

- 从故障转移状态开始。
- 反向重新同步复制关系。

> 不要执行正常的重新同步操作，因为这将丢弃在故障转移过程中写入目标集群的数据。

- 反转复制方向。

步骤

1. 执行[反向重新同步失败的复制关系]步骤。
2. 执行[反向应用程序复制方向]步骤。

删除复制关系

您可以随时删除复制关系。删除应用程序复制关系后，将生成两个彼此独立的应用程序，它们之间没有任何关系。

步骤

1. 在当前目标集群上，删除 AppMirrorRelationship CR：

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

使用Trident Protect 迁移应用程序

您可以通过恢复备份数据在集群之间或不同的存储类之间迁移您的应用程序。

迁移应用程序时，为该应用程序配置的所有执行钩子都会随应用程序一起迁移。如果存在恢复后执行钩子，它将作为恢复操作的一部分自动运行。

备份和恢复操作

针对以下场景，您可以自动执行特定的备份和恢复任务，以执行备份和恢复操作。

克隆到同一集群

要将应用程序克隆到同一集群，请创建快照或备份，然后将数据还原到同一集群。

步骤

1. 执行以下操作之一：
   a. "创建快照"。
   b. "创建备份"。
2. 在同一集群上，根据您创建的是快照还是备份，执行以下操作之一：
   a. "从快照恢复数据"。
   b. "从备份中恢复数据"。

克隆到不同的集群

要将应用程序克隆到不同的集群（执行跨集群克隆），请在源集群上创建备份，然后将备份还原到不同的集群。请确保目标集群上已安装Trident Protect。

您可以使用以下方法在不同的集群之间复制应用程序"SnapMirror 复制"。

步骤

1. "创建备份"。
2. 确保目标集群上已配置包含备份的对象存储桶的 AppVault CR。
3. 在目标集群上，"从备份中恢复数据" 。

将应用程序从一个存储类迁移到另一个存储类

您可以通过将备份还原到目标存储类，将应用程序从一个存储类迁移到另一个存储类。

例如（不包括恢复 CR 中的密钥）：

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    - destination: "${destinationNamespace}"
      source: "${sourceNamespace}"
  storageClassMapping:
    - destination: "${destinationStorageClass}"
      source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
      kind: Secret
      version: v1
    resourceSelectionCriteria: exclude
```

使用 **CR** 恢复快照

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-snapshot-restore-cr.yaml`。
2. 在您创建的文件中，配置以下属性：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.appArchivePath**: AppVault 内存储快照内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get snapshots <my-snapshot-name> -n trident-protect -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.appVaultRef**: (必需) 存储快照内容的 AppVault 的名称。
   - **spec.namespaceMapping**：恢复操作的源命名空间到目标命名空间的映射。代替 `my-source-namespace`和 `my-destination-namespace`利用来自周围环境的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
       appArchivePath: my-snapshot-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. 如果您需要仅选择要恢复的应用程序的某些资源，则可以添加筛选条件，以包含或排除带有特定标签的资源：
   - **resourceFilter.resourceSelectionCriteria**：（筛选时必需）使用 `include or exclude`包含或排除 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包含或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在该数组中定义多个元素，则它们之间按 OR 运算匹配，每个元素内的字段（组、种类、版本）之间按 AND 运算匹配。
       - **resourceMatchers[].group**: (可选) 要筛选的资源组。
       - **resourceMatchers[].kind**: (可选) 要筛选的资源类型。
       - **resourceMatchers[].version**: (可选) 要筛选的资源版本。
       - **resourceMatchers[].names**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (可选) 要过滤的资源的 Kubernetes 元数据.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (可选) 资源的 Kubernetes 元数据.name 字段中的标签选择器字符串，如在以下位置定义： "Kubernetes 文档" 。例如： `"trident.netapp.io/os=linux"` 。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 填写完之后 `trident-protect-snapshot-restore-cr.yaml`将文件的值正确后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用 **CLI** 恢复快照

步骤

1. 将快照还原到不同的命名空间，并将括号中的值替换为您环境中的信息。

   - 这 `snapshot`参数使用命名空间和快照名称，格式如下 `<namespace>/<name>`。
   - 这 `namespace-mapping`该参数使用冒号分隔的命名空间，将源命名空间映射到正确的目标命名空间，格式如下： `source1:dest1,source2:dest2` 。

     例如：

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

管理Trident Protect 执行钩子

执行钩子是一种自定义操作，您可以将其配置为与受管应用程序的数据保护操作一起运行。例如，如果您有一个数据库应用程序，则可以使用执行挂钩在快照之前暂停所有数据库事务，并在快照完成后恢复事务。这确保了应用程序一致的快照。

执行钩子的类型

Trident Protect 支持以下几种执行钩子类型，具体取决于它们的运行时机：

- 预快照
- 快照后
- 预备份
- 备份后
- 恢复后
- 故障转移后

执行顺序

当运行数据保护操作时，执行挂钩事件按以下顺序发生：

1. 任何适用的自定义预操作执行挂钩都在适当的容器上运行。您可以根据需要创建和运行任意数量的自定义预操作挂钩，但这些挂钩在操作之前的执行顺序既无法保证也无法配置。
2. 如果适用，则会发生文件系统冻结。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。
3. 执行数据保护操作。
4. 如果适用，冻结的文件系统将被解冻。
5. 任何适用的自定义后操作执行挂钩都在适当的容器上运行。您可以根据需要创建和运行任意数量的自定义后操作挂钩，但操作后这些挂钩的执行顺序既无法保证也无法配置。

如果您创建多个相同类型的执行挂钩（例如，预快照），则无法保证这些挂钩的执行顺序。但是，不同类型的钩子的执行顺序是有保证的。例如，以下是具有所有不同类型钩子的配置的执行顺序：

1. 快照前钩子执行
2. 快照后钩子执行
3. 执行备份前挂钩
4. 执行备份后钩子

前面的顺序示例仅适用于运行不使用现有快照的备份时。

在生产环境中启用执行挂钩脚本之前，您应该始终对其进行测试。您可以使用“kubectl exec”命令方便地测试脚本。在生产环境中启用执行挂钩后，测试生成的快照和备份以确保它们一致。您可以通过将应用程序克隆到临时命名空间、恢复快照或备份，然后测试应用程序来执行此操作。

如果快照前执行钩子添加、更改或删除 Kubernetes 资源，则这些更改将包含在快照或备份以及任何后续恢复操作中。

关于自定义执行钩子的重要说明

在为您的应用程序规划执行挂钩时，请考虑以下事项。

- 执行钩子必须使用脚本来执行操作。许多执行钩子可以引用同一个脚本。
- Trident Protect 要求执行钩子使用的脚本以可执行 shell 脚本的格式编写。
- 脚本大小限制为 96KB。
- Trident Protect 使用执行钩子设置和任何匹配条件来确定哪些钩子适用于快照、备份或恢复操作。

由于执行钩子通常会减少或完全禁用其所针对的应用程序的功能，因此您应该始终尝试尽量减少自定义执行钩子的运行时间。如果您启动带有相关执行挂钩的备份或快照操作，但随后取消它，则如果备份或快照操作已经开始，则仍允许挂钩运行。这意味着备份后执行挂钩中使用的逻辑不能假定备份已完成。

执行钩子过滤器

当您为应用程序添加或编辑执行挂钩时，您可以向执行挂钩添加过滤器来管理该挂钩将匹配哪些容器。过滤器对于在所有容器上使用相同容器镜像但可能将每个镜像用于不同目的的应用程序（例如 Elasticsearch）很有用。过滤器允许您创建执行挂钩在某些（但不一定是所有）相同的容器上运行的场景。如果为单个执行挂钩创建多个过滤器，它们将通过逻辑 AND 运算符组合在一起。每个执行挂钩最多可以有 10 个活动过滤器。

添加到执行挂钩的每个过滤器都使用正则表达式来匹配集群中的容器。当钩子与容器匹配时，钩子将在该容器上运行其关联的脚本。过滤器的正则表达式使用正则表达式 2 (RE2) 语法，该语法不支持创建从匹配列表中排除容器的过滤器。有关Trident Protect 在执行钩子过滤器中支持的正则表达式语法的详细信息，请参阅 "正则表达式 2 (RE2) 语法支持"。

如果将命名空间过滤器添加到在恢复或克隆操作后运行的执行挂钩，并且恢复或克隆源和目标位于不同的命名空间中，则命名空间过滤器仅适用于目标命名空间。

执行钩子示例

访问 "NetApp Verda GitHub 项目"下载流行应用程序（如 Apache Cassandra 和 Elasticsearch）的真实执行挂钩。您还可以查看示例并获得构建您自己的自定义执行挂钩的想法。

创建执行钩子

您可以使用以下方法为应用程序创建自定义执行钩子。您需要拥有所有者、管理员或成员权限才能创建执行钩子。

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-hook.yaml`。
2. 配置以下属性以匹配您的Trident Protect 环境和集群配置：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要运行执行钩子的应用程序的 Kubernetes 名称。
   - **spec.stage**: (*Required*) 一个字符串，指示执行钩子应该在操作的哪个阶段运行。可能值：
     - 预
     - 发布
   - **spec.action**: (*Required*) 一个字符串，指示执行钩子将采取什么操作，假设指定的任何执行钩子过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.enabled**: (可选) 指示此执行钩子是否已启用或禁用。如果未指定，则默认值为 true。
   - **spec.hookSource**: (必需) 包含 base64 编码的 hook 脚本的字符串。
   - **spec.timeout**: (可选) 定义执行钩子允许运行的分钟数的数字。最小值为 1 分钟，如果未指定，则默认值为 25 分钟。
   - **spec.arguments**: (可选) 一个 YAML 列表，用于指定执行钩子的参数。
   - **spec.matchingCriteria**: (可选) 一个可选的条件键值对列表，每个键值对构成一个执行钩子过滤器。每个执行钩子最多可以添加 10 个过滤器。
   - **spec.matchingCriteria.type**: (可选) 用于标识执行钩过滤器类型的字符串。可能值：
     - 容器图像
     - 容器名称
     - Pod名称
     - PodLabel
     - 命名空间名称
   - **spec.matchingCriteria.value**: (可选) 用于标识执行钩过滤器值的字符串或正则表达式。

     YAML 示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. 在用正确的值填充 CR 文件后，应用 CR：

```
kubectl apply -f trident-protect-hook.yaml
```

使用 **CLI**

步骤

1. 创建执行钩子，将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

手动运行执行钩子

您可以手动运行执行钩子进行测试，或者在失败后需要手动重新运行钩子时也可以这样做。您需要拥有所有者、管理员或成员权限才能手动运行执行钩子。

手动运行执行钩子包含两个基本步骤：

1. 创建资源备份，该备份会收集资源并创建它们的备份，从而确定钩子函数的运行位置。
2. 针对备份运行执行钩子

步骤 **1**：创建资源备份

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-resource-backup.yaml`。
2. 配置以下属性以匹配您的Trident Protect 环境和集群配置：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 要为其创建资源备份的应用程序的 Kubernetes 名称。
   - **spec.appVaultRef**: (必需) 存储备份内容的 AppVault 的名称。
   - **spec.appArchivePath**: AppVault 内存储备份内容的路径。您可以使用以下命令查找此路径：

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

     YAML 示例：

   ```
   ---
   apiVersion: protect.trident.netapp.io/v1
   kind: ResourceBackup
   metadata:
     name: example-resource-backup
   spec:
     applicationRef: my-app-name
     appVaultRef: my-appvault-name
     appArchivePath: example-resource-backup
   ```

3. 在用正确的值填充 CR 文件后，应用 CR：

   ```
   kubectl apply -f trident-protect-resource-backup.yaml
   ```

使用 **CLI**

步骤

1. 创建备份，将括号中的值替换为您环境中的信息。例如：

   ```
   tridentctl protect create resourcebackup <my_backup_name> --app
   <my_app_name> --appvault <my_appvault_name> -n
   <my_app_namespace> --app-archive-path <app_archive_path>
   ```

2. 查看备份状态。您可以重复使用此示例命令，直到操作完成：

```
tridentctl protect get resourcebackup -n <my_app_namespace>
<my_backup_name>
```

3. 确认备份是否成功：

```
kubectl describe resourcebackup <my_backup_name>
```

步骤 **2**：运行执行钩子

使用 **CR**

步骤

1. 创建自定义资源 (CR) 文件并将其命名为 `trident-protect-hook-run.yaml`。
2. 配置以下属性以匹配您的Trident Protect 环境和集群配置：
   - **metadata.name**: (必填) 此自定义资源的名称；请为您的环境选择一个唯一且有意义的名称。
   - **spec.applicationRef**: (必需) 确保此值与您在步骤 1 中创建的 ResourceBackup CR 中的应用程序名称匹配。
   - **spec.appVaultRef**: (必需) 确保此值与您在步骤 1 中创建的 ResourceBackup CR 中的 appVaultRef 匹配。
   - **spec.appArchivePath**：确保此值与您在步骤 1 中创建的 ResourceBackup CR 中的 appArchivePath 匹配。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.action**: (*Required*) 一个字符串，指示执行钩子将采取什么操作，假设指定的任何执行钩子过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.stage**: (*Required*) 一个字符串，指示执行钩子应该在操作的哪个阶段运行。这次钩子运行不会在任何其他阶段运行钩子。可能值：
     - 预
     - 发布

     YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

3. 在用正确的值填充 CR 文件后，应用 CR：

```
kubectl apply -f trident-protect-hook-run.yaml
```

使用 **CLI**

步骤

1. 创建手动执行钩子运行请求：

```
tridentctl protect create exechooksrun <my_exec_hook_run_name>
-n <my_app_namespace> --action snapshot --stage <pre_or_post>
--app <my_app_name> --appvault <my_appvault_name> --path
<my_backup_name>
```

2. 检查执行钩子运行状态。您可以重复运行此命令，直到操作完成：

```
tridentctl protect get exechooksrun -n <my_app_namespace>
<my_exec_hook_run_name>
```

3. 描述 exechooksrun 对象以查看最终详细信息和状态：

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

卸载Trident Protect

如果您要从试用版升级到完整版产品，可能需要移除Trident Protect 组件。

要移除Trident Protect，请执行以下步骤。

步骤

1. 删除Trident Protect CR 文件：

> 25.06 及更高版本不需要此步骤。

```
helm uninstall -n trident-protect trident-protect-crds
```

2. 移除Trident保护：

```
helm uninstall -n trident-protect trident-protect
```

3. 移除Trident Protect 命名空间：

```
kubectl delete ns trident-protect
```

Trident和Trident Protect 博客

您可以在这里找到一些很棒的NetApp Trident和Trident Protect 博客：

Trident博客

- 2025年5月9日："使用 Amazon EKS 插件为 FSx for ONTAP自动配置Trident后端"
- 2025年8月19日："增强数据一致性：使用Trident在 OpenShift 虚拟化中使用卷组快照"
- 2025年4月15日："NetApp Trident与Google Cloud NetApp Volumes for SMB 协议"
- 2025年4月14日："利用Trident 25.02 的光纤通道协议实现 Kubernetes 上的持久存储"
- 2025年4月14日："释放NetApp ASA r2 系统在 Kubernetes 块存储方面的强大功能"
- 2025年3月31日："使用新的认证操作员简化 Red Hat OpenShift 上的Trident安装"
- 2025年3月27日："使用Google Cloud NetApp Volumes为 SMB 配置Trident"
- 2025年3月5日："解锁无缝的 iSCSI 存储集成：AWS ROSA 集群上的 FSxN 指南"
- 2025年2月27日："使用Trident、GKE 和Google Cloud NetApp Volumes部署云身份"
- 2024年12月12日："Trident引入光纤通道支持"
- 2024年11月26日："Trident 25.01：通过新功能和增强功能增强 Kubernetes 存储体验"
- 2024年11月11日："NetApp Trident与Google Cloud NetApp Volumes"
- 2024年10月29日："使用Trident在 AWS 上将Amazon FSx for NetApp ONTAP与 Red Hat OpenShift 服务 (ROSA) 结合使用"
- 2024年10月29日："使用 ROSA 上的 OpenShift 虚拟化和Amazon FSx for NetApp ONTAP实时迁移虚拟机"
- 2024年7月8日："使用 NVMe/TCP 在 Amazon EKS 上为现代容器化应用程序使用ONTAP存储"
- 2024年7月1日："使用Google Cloud NetApp Volumes Flex 和Astra Trident实现 Kubernetes 无缝存储"
- 2024年6月11日："ONTAP作为 OpenShift 中集成映像注册表的后端存储"

Trident Protect博客

- 2025年5月16日："利用Trident Protect 的恢复后钩子实现注册表故障转移的自动化，以进行灾难恢复"
- 2025年5月16日："使用NetApp Trident Protect 进行 OpenShift 虚拟化灾难恢复"
- 2025年5月13日："使用Trident Protect 备份和恢复进行存储类迁移"
- 2025年5月9日："使用Trident Protect 恢复后钩子重新扩展 Kubernetes 应用程序"
- 2025年4月3日："Trident Protect 升级：Kubernetes 复制保护与灾难恢复"
- 2025年3月13日："OpenShift虚拟化虚拟机的崩溃一致性备份和恢复操作"
- 2025年3月11日："将 GitOps 模式扩展到使用NetApp Trident进行应用程序数据保护"
- 2025年3月3日："Trident 25.02：通过激动人心的新功能提升 Red Hat OpenShift 体验"
- 2025年1月15日："隆重推出Trident Protect 基于角色的访问控制"

- 2024年11月11日： "隆重推出 tridentctl protect： Trident Protect 的强大命令行界面"
- 2024年11月11日："Kubernetes驱动的数据管理： Trident Protect开启新时代"

知识和支持

常见问题解答

查找有关Trident 的安装、配置、升级和故障排除的常见问题解答。

一般性问题

Trident 的发布频率如何?

从 24.02 版本开始， Trident每四个月发布一次：二月、六月和十月。

Trident是否支持特定版本 **Kubernetes** 中发布的所有功能?

Trident通常不支持 Kubernetes 中的 alpha 功能。 Trident可能会在 Kubernetes beta 版本之后的两个Trident版本中支持 beta 功能。

Trident 的运行是否依赖于其他**NetApp**产品?

Trident不依赖于其他NetApp软件产品，它可以作为独立应用程序运行。但是，您应该使用NetApp后端存储设备。

如何获取完整的**Trident**配置详情?

使用 `tridentctl get`命令以获取有关您的Trident配置的更多信息。

我能否获取有关**Trident**如何配置存储的指标?

是可用于收集有关Trident操作的信息的 Prometheus 端点，例如管理的后端数量、已配置的卷数量、消耗的字节数等等。您还可以使用"Cloud Insights"用于监测和分析。

使用**Trident**作为 **CSI** 配置器时，用户体验是否会发生变化?

不，用户体验和功能方面没有任何变化。使用的配置程序名称是 `csi.trident.netapp.io`。如果您想使用当前和未来版本提供的所有新功能，建议使用此方法安装Trident 。

在 Kubernetes 集群上安装和使用Trident

Trident是否支持从私有注册表进行离线安装?

是的， Trident可以离线安装。参考"了解Trident安装"。

我可以远程安装**Trident**吗?

是Trident 18.10 及更高版本支持从任何具备以下功能的机器进行远程安装： `kubectl`访问集群。后 `kubectl`访问权限已验证（例如，发起一次访问） `kubectl get nodes`从远程机器发出命令进行验证），按照安装说明进行操作。

我可以使用**Trident**配置高可用性吗?

Trident以 Kubernetes Deployment（ReplicaSet）的形式安装，只有一个实例，因此它内置了高可用性 (HA)。您不应该增加 Deployment 中的副本数量。如果安装了Trident 的节点丢失或 pod 无法访问，Kubernetes 会自动将 pod 重新部署到集群中的健康节点。 Trident仅用于控制平面，因此如果重新部署Trident，当前已安装的 pod 不会受到影响。

Trident是否需要访问 **kube-system** 命名空间?

Trident从 Kubernetes API 服务器读取信息，以确定应用程序何时请求新的 PVC，因此它需要访问 kube-system。

Trident使用哪些角色和权限?

Trident安装程序会创建一个 Kubernetes ClusterRole，该角色对 Kubernetes 集群的 PersistentVolume、PersistentVolumeClaim、StorageClass 和 Secret 资源具有特定的访问权限。参考"自定义 tridentctl 安装"。

我可以在本地生成**Trident**安装时使用的确切清单文件吗?

如有需要，您可以在本地生成和修改Trident安装时使用的确切清单文件。参考"自定义 tridentctl 安装"。

我可以为两个独立的 **Kubernetes** 集群中的两个独立的**Trident**实例共享同一个**ONTAP**后端 **SVM** 吗?

虽然不建议这样做，但您可以为两个Trident实例使用同一个后端 SVM。安装期间为每个实例指定唯一的卷名称和/或指定唯一的 `StoragePrefix`参数 `setup/backend.json`文件。这是为了确保两个实例不会使用同一个FlexVol volume。

是否可以在 **ContainerLinux**（原 **CoreOS**）下安装**Trident** ?

Trident实际上就是一个 Kubernetes pod，可以安装在任何运行 Kubernetes 的地方。

Trident可以与**NetApp Cloud Volumes ONTAP**一起使用吗?

是的， Trident在 AWS、Google Cloud 和 Azure 上均受支持。

Trident是否与 **Cloud Volumes Services** 兼容?

是的， Trident支持 Azure 中的Azure NetApp Files服务以及 GCP 中的Cloud Volumes Service。

故障排除和支持

NetApp是否支持**Trident**?

虽然Trident是开源且免费提供的，但只要您的NetApp后端受支持， NetApp就会完全支持它。

我该如何提交支持申请?

如需提出援助申请，请执行以下操作之一：

1. 请联系您的支持客户经理，获取帮助以提交工单。
2. 请联系我们提交支持案例。 "NetApp 支持" 。

如何生成支持日志包?

您可以通过运行以下命令来创建支持包 `tridentctl logs -a`。除了捆绑包中捕获的日志外，还要捕获 kubelet 日志，以诊断 Kubernetes 端的挂载问题。获取 kubelet 日志的说明因 Kubernetes 的安装方式而异。

如果我需要提出新功能请求，该怎么办?

创建问题 "Trident Github"并在问题的主题和描述中注明 **RFE**。

我应该在哪里提交缺陷报告?

创建问题 "Trident Github"。请务必提供与问题相关的所有必要信息和日志。

如果我对**Trident**有个需要澄清的简短问题，该怎么办？这里有社区或论坛吗?

如果您有任何疑问、问题或请求，请通过我们的Trident联系我们。"Discord 频道"或者 GitHub。

我的存储系统密码已更改， **Trident**无法再工作，我该如何恢复?

使用以下方式更新后端密码 `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。代替 `myBackend`例如，请使用您的后端名称，并且 `/path/to_new_backend.json`通往正确路径 `backend.json`文件。

Trident找不到我的**Kubernetes**节点。我该如何解决这个问题?

Trident找不到 Kubernetes 节点可能有以下两种情况。这可能是由于 Kubernetes 内部的网络问题或 DNS 问题引起的。在每个 Kubernetes 节点上运行的Trident节点守护程序集必须能够与Trident控制器通信，以便将节点注册到Trident。如果在Trident安装后发生了网络变更，则只有在向集群中添加新的 Kubernetes 节点时才会遇到此问题。

如果**Trident**舱被摧毁，我会丢失数据吗?

即使Trident舱被摧毁，数据也不会丢失。 Trident元数据存储在 CRD 对象中。所有由Trident提供的PV都将正常运行。

升级版Trident

我可以直接从旧版本升级到新版本（跳过几个版本）吗?

NetApp支持将Trident从一个主要版本升级到下一个紧邻的主要版本。您可以从 18.xx 版本升级到 19.xx 版本，从 19.xx 版本升级到 20.xx 版本，依此类推。在生产环境部署之前，应该在实验室环境中测试升级。

是否可以将**Trident**降级到之前的版本?

如果您需要修复升级后发现的错误、依赖项问题或升级失败/不完整，您应该"卸载Trident"然后按照该版本的具体说明重新安装早期版本。这是降级到早期版本的唯一推荐方法。

管理后端和卷

我是否需要在**ONTAP**后端定义文件中同时定义 **ManagementLIF** 和 **DataLIF**?

管理层 LIF 是强制性的。 DataLIF 各不相同：

- ONTAP SAN：不要指定用于 iSCSI。Trident的使用"ONTAP选择性 LUN 地图"发现建立多路径会话所需的 iSCI LIF。如果出现以下情况，则会生成警告： `dataLIF`已明确定义。参考 "ONTAP SAN 配置选项和示例"了解详情。
- ONTAP NAS： NetApp建议指定 `dataLIF`。如果未提供， Trident将从 SVM 获取 dataLIF。您可以指定一个完全限定域名 (FQDN) 用于 NFS 挂载操作，从而创建轮询 DNS 以在多个 dataLIF 之间进行负载均衡。请参阅"ONTAP NAS 配置选项和示例"详情请见

Trident能否为**ONTAP**后端配置**CHAP**?

是Trident支持ONTAP后端的双向 CHAP。这需要设置 `useCHAP=true`在您的后端配置中。

如何使用**Trident**管理导出策略?

从 20.04 版本开始， Trident可以动态创建和管理导出策略。这样，存储管理员就可以在其后端配置中提供一个或多个 CIDR 块，并让Trident将落入这些范围内的节点 IP 添加到它创建的导出策略中。通过这种方式， Trident可以自动管理给定 CIDR 内 IP 地址的节点的规则添加和删除。

IPv6 地址可以用于管理和数据 **LIF** 吗?

Trident支持为以下设备定义 IPv6 地址：

- `managementLIF`和 `dataLIF`适用于ONTAP NAS 后端。
- `managementLIF`适用于ONTAP SAN 后端。您无法指定 `dataLIF`基于ONTAP SAN 后端。

必须使用标志安装Trident 。 `--use-ipv6` （为了 `tridentctl`安装）， `IPv6` （对于Trident操作员），或 `tridentTPv6`（用于 Helm 安装）使其能够通过 IPv6 运行。

是否可以在后端更新管理 **LIF**?

是的，可以使用以下方式更新后端管理 LIF： `tridentctl update backend`命令。

是否可以在后端更新 **DataLIF**?

您可以更新 DataLIF `ontap-nas`和 `ontap-nas-economy`仅有的。

我可以在**Trident for Kubernetes** 中创建多个后端吗?

Trident可以同时支持多个后端，既可以使用相同的驱动程序，也可以使用不同的驱动程序。

Trident是如何存储后端凭证的?

Trident将后端凭据存储为 Kubernetes Secrets。

Trident是如何选择特定后端的?

如果后端属性无法用于自动为类选择合适的连接池，则 `storagePools`和 `additionalStoragePools`参数用于选择一组特定的池子。

如何确保**Trident**不会从特定的后端进行配置？

这 `excludeStoragePools`该参数用于筛选Trident用于配置的池集，并将删除任何匹配的池。

如果存在多个同类型的后端， **Trident**如何选择使用哪个后端？

如果配置了多个相同类型的后端， Trident会根据配置中的参数选择合适的后端。 `StorageClass`和 `PersistentVolumeClaim`。例如，如果存在多个 ontap-nas 驱动程序后端， Trident会尝试匹配参数。 `StorageClass`和 `PersistentVolumeClaim`结合并匹配一个能够满足所列要求的后端 `StorageClass`和 `PersistentVolumeClaim`。如果存在多个后端与请求匹配， Trident会随机选择其中一个。

Trident是否支持与**Element/ SolidFire**的双向**CHAP**协议？

是

Trident如何在**ONTAP**卷上部署 **Qtree**？单个卷上最多可以部署多少个 **Qtree**？

这 `ontap-nas-economy`驱动程序在同一个FlexVol volume中最多可创建 200 个 Qtree（可在 50 到 300 之间配置），每个集群节点最多可创建 100,000 个 Qtree，每个集群最多可创建 240 万个 Qtree。当你进入一个新的 `PersistentVolumeClaim`如果由经济型驱动程序提供服务，则该驱动程序会查看是否已存在可以为新的 Qtree 提供服务的FlexVol volume。如果不存在可以为 Qtree 提供服务的FlexVol volume，则会创建一个新的FlexVol volume。

如何为**ONTAP NAS** 上配置的卷设置 **Unix** 权限？

您可以通过在后端定义文件中设置参数，为Trident提供的卷设置 Unix 权限。

在配置卷时，如何配置一组明确的**ONTAP NFS** 挂载选项？

默认情况下， Trident不会为 Kubernetes 设置任何挂载选项。要在 Kubernetes 存储类中指定挂载选项，请按照给出的示例进行操作。"此处" 。

如何将已配置的卷设置为特定的导出策略？

要允许相应的主机访问卷，请使用以下方法： `exportPolicy`在后端定义文件中配置的参数。

如何使用**Trident**和**ONTAP**设置卷加密？

您可以使用后端定义文件中的加密参数，对Trident提供的卷设置加密。更多信息，请参阅："Trident如何与 NVE 和 NAE 协同工作"

通过**Trident**为**ONTAP**实现 **QoS** 的最佳方法是什么？

使用 `StorageClasses`为ONTAP实现 QoS。

如何在**Trident**中指定精简配置或厚配置？

ONTAP驱动程序支持精简配置或厚配置。 ONTAP驱动程序默认采用精简配置。如果需要厚配置，则应配置后端定义文件或 `StorageClass`。如果两者都已配置， `StorageClass`优先考虑。为ONTAP配置以下内容：

1. 在 `StorageClass`，设置 `provisioningType`属性为厚。

2. 在后端定义文件中，通过设置启用厚卷 `backend spaceReserve parameter`作为体积。

如何确保即使我不小心删除了**PVC**，正在使用的卷也不会被删除?

从 Kubernetes 1.10 版本开始，PVC 保护功能会自动启用。

我可以种植由**Trident**生产的**NFS PVC**吗?

是您可以膨胀由Trident制造的 PVC。请注意，卷自动增长是ONTAP 的一项功能，不适用于Trident。

我可以在**SnapMirror**数据保护 **(DP)** 模式或离线模式下导入卷吗?

如果外部卷处于 DP 模式或离线，则卷导入失败。您收到以下错误信息：

```
Error: could not import volume: volume import failed to get size of
volume: volume <name> was not found (400 Bad Request) command terminated
with exit code 1.
Make sure to remove the DP mode or put the volume online before importing
the volume.
```

如何将资源配额转换为**NetApp**集群?

只要NetApp存储有足够的容量，Kubernetes 存储资源配额就应该能够正常工作。当NetApp存储由于容量不足而无法满足 Kubernetes 配额设置时， Trident会尝试进行配置，但会出错。

我可以使用**Trident**创建卷快照吗?

是Trident支持创建按需卷快照和从快照创建持久卷。要从快照创建 PV，请确保 `VolumeSnapshotDataSource`功能门已启用。

支持**Trident**卷快照的驱动程序有哪些?

截至今日，我们的产品已提供按需快照支持。 `ontap-nas` ， `ontap-nas-flexgroup` ， `ontap-san` ， `ontap-san-economy` ， `solidfire-san` ， `gcp-cvs` ， 和 `azure-netapp-files`后端驱动程序。

如何对通过**Trident ONTAP**配置的卷进行快照备份?

这可以在以下平台找到： `ontap-nas` ， `ontap-san` ， 和 `ontap-nas-flexgroup`司机。您还可以指定一个`snapshotPolicy`对于 `ontap-san-economy`FlexVol级别的驱动。

这也可以在以下平台找到： `ontap-nas-economy`驱动程序，但粒度是FlexVol volume级别，而不是 qtree 级别。要启用对Trident创建的卷进行快照的功能，请设置后端参数选项。 `snapshotPolicy`根据ONTAP后端定义的所需快照策略。 Trident无法获知存储控制器拍摄的任何快照。

我可以为通过**Trident**配置的卷设置快照保留百分比吗?

是的，您可以通过Trident设置预留特定百分比的磁盘空间来存储快照副本。 `snapshotReserve`后端定义文件中的属性。如果您已配置 `snapshotPolicy`和 `snapshotReserve`在后端定义文件中，快照保留百分比是根据以下方式设置的： `snapshotReserve`后端文件中提到的百分比。如果 `snapshotReserve`没有提及百分比数值，

ONTAP默认将快照保留百分比设为 5%。如果 `snapshotPolicy`如果选项设置为“无”，则快照保留百分比设置为 0。

我可以直接访问卷快照目录并复制文件吗?

是的，您可以通过设置来访问Trident配置的卷上的快照目录。 `snapshotDir`后端定义文件中的参数。

我可以通过**Trident**为卷设置**SnapMirror**吗?

目前， SnapMirror必须通过ONTAP CLI 或OnCommand System Manager在外部进行设置。

如何将持久卷恢复到特定的**ONTAP**快照?

要将卷还原到ONTAP快照，请执行以下步骤：

1. 使正在使用持久卷的应用程序 pod 静默。
2. 通过ONTAP CLI 或OnCommand System Manager恢复到所需的快照。
3. 重启应用程序 pod。

Trident能否在配置了负载均衡镜像的 **SVM** 上配置卷?

可以为通过 NFS 提供数据的 SVM 的根卷创建负载均衡镜像。 ONTAP会自动更新由Trident创建的卷的负载均衡镜像。这可能会导致卷安装延迟。使用Trident创建多个卷时，卷的配置取决于ONTAP更新负载均衡镜像。

如何将每个客户/租户的存储类别使用情况分开统计?

Kubernetes 不允许在命名空间中使用存储类。但是，您可以使用 Kubernetes 通过使用存储资源配额（每个命名空间一个配额）来限制每个命名空间中特定存储类的使用量。要拒绝特定命名空间对特定存储的访问，请将该存储类的资源配额设置为 0。

故障排除

使用此处提供的提示来解决您在安装和使用Trident时可能遇到的问题。

> 如需Trident的帮助，请使用以下命令创建支持包 `tridentctl logs -a -n trident`然后将其发送给NetApp支持部门。

常规故障排除

- 如果Trident舱无法正常升空（例如，当Trident舱卡在……中时 `ContainerCreating`阶段（准备容器少于两个）运行 `kubectl -n trident describe deployment trident`和 `kubectl -n trident describe pod trident-**-**`可以提供更多见解。获取 `kubelet` 日志（例如，通过 `journalctl -xeu kubelet`）也可能有所帮助。
- 如果Trident日志中的信息不足，您可以尝试通过传递参数来启用Trident的调试模式。 `-d`根据您的安装选项，向安装参数添加标志。

  然后确认调试模式已设置 `./tridentctl logs -n trident`并正在寻找 `level=debug msg`在日志中。

已安装操作员

```
kubectl patch torc trident -n <namespace> --type=merge -p
'{"spec":{"debug":true}}'
```

这将重启所有Trident pod，这可能需要几秒钟时间。您可以通过观察输出结果中的“AGE”列来验证这一点。 `kubectl get pod -n trident` 。

适用于Trident 20.07 和 20.10 版本 `tprov`代替 `torc`。

使用 **Helm** 安装

```
helm upgrade <name> trident-operator-21.07.1-custom.tgz --set
tridentDebug=true`
```

使用 **tridentctl** 安装

```
./tridentctl uninstall -n trident
./tridentctl install -d -n trident
```

- 您还可以通过添加以下命令来获取每个后端的调试日志。 `debugTraceFlags`在您的后端定义中。例如，包括 `debugTraceFlags: {"api":true, "method":true,}`获取Trident日志中的 `API` 调用和方法遍历。现有后端可以有 `debugTraceFlags`配置为 `tridentctl backend update`。
- 使用 Red Hat Enterprise Linux CoreOS (RHCOS) 时，请确保： `iscsid`在工作节点上已启用，并且默认情况下已启动。这可以通过使用 OpenShift MachineConfigs 或修改 Ignition 模板来实现。
- 使用Trident时可能会遇到的常见问题是…… "Azure NetApp Files"指的是租户和客户端密钥来自权限不足的应用程序注册。有关Trident要求的完整列表，请参阅"Azure NetApp Files"配置。
- 如果将光伏系统安装到集装箱上遇到问题，请确保： `rpcbind`已安装并正在运行。使用主机操作系统所需的软件包管理器并检查是否 `rpcbind`正在运行。您可以查看以下状态： `rpcbind`通过运行服务 `systemctl status rpcbind`或其等效物。
- 如果Trident后端报告它处于 `failed`尽管之前运行正常，但当前状态可能是由于更改了与后端关联的 SVM/管理员凭据所致。使用以下方式更新后端信息 `tridentctl update backend`或者晃动一下Trident烟囱就能解决这个问题。
- 如果在使用 Docker 作为容器运行时安装Trident时遇到权限问题，请尝试使用以下方式安装Trident： `--in cluster=false`旗帜。这样就不会使用安装程序 pod，从而避免因以下原因导致的权限问题： `trident-installer`用户。
- 使用 `uninstall parameter <Uninstalling Trident>`用于清理运行失败后的残局。默认情况下，该脚本不会删除Trident创建的 CRD，因此即使在运行的部署中，卸载和重新安装也是安全的。
- 如果您想降级到早期版本的Trident，请先运行以下命令： `tridentctl uninstall`移除Trident的命令。下载所需文件 "Trident版"并使用以下方式安装 `tridentctl install`命令。
- 安装成功后，如果PVC管卡在…… `Pending`阶段，运行 `kubectl describe pvc`可以提供更多关于Trident为何未能为此PVC配置PV的信息。

使用操作员部署Trident失败

如果您使用 Operator 部署Trident ，则状态为： `TridentOrchestrator`变化来自 `Installing`到 `Installed`。如果你观察 `Failed`如果操作员处于异常状态且无法自行恢复，则应运行以下命令检查操作员的日志：

```
tridentctl logs -l trident-operator
```

追踪 trident-operator 容器的日志可以指出问题所在。例如，在与外界隔离的环境中，可能无法从上游镜像仓库拉取所需的容器镜像。

要了解为什么Trident安装失败，您应该查看以下内容： `TridentOrchestrator`地位。

```
kubectl describe torc trident-2
Name:         trident-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Status:
  Current Installation Params:
    IPv6:
    Autosupport Hostname:
    Autosupport Image:
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:
    Image Pull Secrets:         <nil>
    Image Registry:
    k8sTimeout:
    Kubelet Dir:
    Log Format:
    Silence Autosupport:
    Trident Image:
  Message:                      Trident is bound to another CR 'trident'
  Namespace:                    trident-2
  Status:                       Error
  Version:
Events:
  Type     Reason  Age                From                          Message
  ----     ------  ----               ----                          -------
  Warning  Error   16s (x2 over 16s)  trident-operator.netapp.io  Trident
is bound to another CR 'trident'
```

此错误表明已存在 `TridentOrchestrator`用于安装Trident 的程序。由于每个 Kubernetes 集群只能有一个Trident 实例，因此 Operator 会确保在任何给定时间都只有一个活跃的 Trident 实例。 `TridentOrchestrator`它能够创造。

此外，观察Trident舱的状态通常可以表明是否存在异常情况。

```
kubectl get pods -n trident

NAME                                READY   STATUS             RESTARTS
AGE
trident-csi-4p5kq                   1/2     ImagePullBackOff   0
5m18s
trident-csi-6f45bfd8b6-vfrkw        4/5     ImagePullBackOff   0
5m19s
trident-csi-9q5xc                   1/2     ImagePullBackOff   0
5m18s
trident-csi-9v95z                   1/2     ImagePullBackOff   0
5m18s
trident-operator-766f7b8658-ldzsv   1/1     Running            0
8m17s
```

您可以清楚地看到，由于一个或多个容器镜像未获取，因此 pod 无法完全初始化。

要解决这个问题，你应该编辑 `TridentOrchestrator`CR。或者，您可以删除 `TridentOrchestrator`并根据修改后的准确定义创建一个新的定义。

使用**Trident**部署失败 `tridentctl`

为了帮助找出出错的原因，您可以再次运行安装程序：`-d`使用该参数将开启调试模式，帮助您了解问题所在：

```
./tridentctl install -n trident -d
```

解决问题后，您可以按以下步骤清理安装，然后运行： `tridentctl install`再次执行命令：

```
./tridentctl uninstall -n trident
INFO Deleted Trident deployment.
INFO Deleted cluster role binding.
INFO Deleted cluster role.
INFO Deleted service account.
INFO Removed Trident user from security context constraint.
INFO Trident uninstallation succeeded.
```

彻底移除Trident和CRDs

您可以完全移除Trident以及所有创建的 CRD 和相关的自定义资源。

> 此操作无法撤消。除非你想全新安装Trident，否则不要这样做。要在不删除 CRD 的情况下卸载Trident ，请参阅"卸载Trident"。

Trident操作员

要卸载Trident并使用Trident操作符彻底删除 CRD，请执行以下操作：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

舵

使用 Helm 卸载Trident并彻底删除 CRD：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>tridentctl</code>

卸载Trident后，要彻底删除 CRD，请使用 `tridentctl`

```
tridentctl obliviate crd
```

Kubernetes 1.26 版本中，使用 RWX 原始块命名空间时，NVMe 节点卸载失败

如果您运行的是 Kubernetes 1.26，则在使用 NVMe/TCP 和 RWX 原始块命名空间时，节点取消暂存可能会失败。以下方案提供了应对此故障的变通方法。或者，您可以将 Kubernetes 升级到 1.27 版本。

删除了命名空间和 **pod**。

设想这样一种场景：你将一个Trident管理的命名空间（NVMe 持久卷）附加到一个 pod 上。如果直接从ONTAP后端删除命名空间，则在尝试删除 pod 后，取消暂存过程会卡住。此情况不会影响 Kubernetes 集群或其他功能。

临时解决策

从相应的节点卸载持久卷（与该命名空间对应），并将其删除。

阻塞数据**LIF**

```
 If you block (or bring down) all the dataLIFs of the NVMe Trident
backend, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```
.临时解决策
启动 `dataLIFS` 以恢复全部功能。

已删除命名空间映射

```
 If you remove the `hostNQN` of the worker node from the corresponding
subsystem, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```
.临时解决策
添加 `hostNQN`返回子系统。

当预期启用“v4.2-xattrs”时，NFSv4.2 客户端在升级ONTAP后报告“无效参数”

升级ONTAP后，NFSv4.2 客户端在尝试挂载 NFSv4.2 导出时可能会报告“无效参数”错误。当 `v4.2-xattrs` SVM 上未启用该选项。.解决方法启用 `v4.2-xattrs` 选项或升级到ONTAP 9.12.1 或更高版本，默认情况下此选项处于启用状态。

支持

NetApp以多种方式为Trident提供支持。我们提供全天候 (24x7) 的丰富免费自助支持选项，例如知识库 (KB) 文章和 Discord 频道。

Trident支持

Trident根据您的版本提供三个级别的支持。参考"NetApp软件版本对定义的支持"。

全力支持
: Trident自发布之日起提供十二个月的全面支持。

有限的支持
: Trident在发布日期后的第 13 至 24 个月提供有限的支持。

自给自足
: Trident 的文档可在发布日期后的第 25 个月至第 36 个月内查阅。

版本	全力支持	有限的支持	自给自足
"25.06"	2026年6月	2027年6月	2028年6月

"25.02"	2026年2月	2027年2月	2028年2月
"24.10"	2025年10月	2026年10月	2027年10月
"24.06"	2025年6月	2026年6月	2027年6月
"24.02"	2025年2月	2026年2月	2027年2月
"23.10"	—	2025年10月	2026年10月
"23.07"	—	2025年7月	2026年7月
"23.04"	—	2025年4月	2026年4月
"23.01"	—	—	2026年1月
"22.10"	—	—	2025年10月

自给自足

如需查看完整的故障排除文章列表，请参阅 "NetApp知识库（需要登录）"。

社区支持

我们的平台上有一个活跃的容器用户公共社区（包括Trident开发人员）。"Discord 频道" 。这里是提出有关项目的一般性问题并与志同道合的同行讨论相关话题的好地方。

NetApp技术支持

如需Trident的帮助，请使用以下命令创建支持包 `tridentctl logs -a -n trident`并将其发送至 `NetApp Support <Getting Help>`。

了解更多信息

- "Trident资源"
- "Kubernetes Hub"

参考

Trident港口

了解更多关于Trident用于通信的端口的信息。

Trident港口

Trident使用以下端口在 Kubernetes 内部进行通信：

端口	目的
8443	反向通道 HTTPS
8001	Prometheus 指标端点
8000	Trident REST 服务器
17546	Trident守护进程集 pod 使用的存活/就绪探测端口

安装过程中可以使用以下方法更改活性/就绪探针端口： `--probe-port`旗帜。务必确保工作节点上的其他进程没有使用该端口。

Trident REST API

尽管"tridentctl 命令和选项"这是与Trident REST API 交互的最简单方法，如果您愿意，也可以直接使用 REST 端点。

何时使用 REST API

REST API 适用于在非 Kubernetes 部署中使用Trident作为独立二进制文件的高级安装。

为了更好的安全性，Trident `REST API`在 pod 内运行时，默认仅限于 localhost。要改变这种行为，你需要设置 Trident 的 `-address`在其 pod 配置中设置参数。

使用 REST API

要查看这些 API 的调用示例，请传递调试信息。(`-d`） 旗帜。更多信息，请参阅"使用 tridentctl 管理Trident"。

API 的工作原理如下：

GET

`GET <trident-address>/trident/v1/<object-type>`

列出该类型的所有对象。

`GET <trident-address>/trident/v1/<object-type>/<object-name>`

获取指定对象的详细信息。

POST

POST <trident-address>/trident/v1/<object-type>

创建指定类型的对象。

- 创建对象需要 JSON 配置。有关每种对象类型的详细说明，请参阅："使用 tridentctl 管理Trident" 。
- 如果对象已存在，则行为会有所不同：后端会更新现有对象，而所有其他对象类型都会使操作失败。

DELETE

DELETE <trident-address>/trident/v1/<object-type>/<object-name>

删除指定的资源。

> 与后端或存储类关联的卷将继续存在；这些卷必须单独删除。更多信息，请参阅"使用 tridentctl 管理Trident"。

命令行选项

Trident为Trident编排器提供了几个命令行选项。您可以使用这些选项来修改您的部署。

日志记录

-debug

启用调试输出。

-loglevel <level>

设置日志级别（debug、info、warn、error、fatal）。默认显示信息。

Kubernetes

-k8s_pod

使用此选项或 `-k8s_api_server`启用 Kubernetes 支持。设置此项后， Trident将使用其所在 pod 的 Kubernetes 服务帐户凭据来联系 API 服务器。只有当Trident作为 Pod 在启用了服务帐户的 Kubernetes 集群中运行时，此方法才有效。

-k8s_api_server <insecure-address:insecure-port>

使用此选项或 `-k8s_pod`启用 Kubernetes 支持。指定后， Trident将使用提供的不安全地址和端口连接到 Kubernetes API 服务器。这使得Trident可以部署在 pod 之外；但是，它仅支持与 API 服务器的不安全连接。为了安全连接，请将Trident部署在带有以下组件的 pod 中： `-k8s_pod`选项。

Docker

-volume_driver <name>

注册 Docker 插件时使用的驱动程序名称。默认为 `netapp`。

-driver_port <port-number>

监听此端口，而不是 UNIX 域套接字。

-config <file>

必填项；您必须指定后端配置文件的路径。

休息

-address <ip-or-host>

指定 Trident REST 服务器应该监听的地址。默认为本地主机。当监听本地主机并在 Kubernetes pod 内运行时，REST 接口无法从 pod 外部直接访问。使用 `-address ""`使 REST 接口能够通过 pod IP 地址访问。

> Trident REST 接口可以配置为仅监听和提供服务于 127.0.0.1（对于 IPv4）或 [::1]（对于 IPv6）。

-port <port-number>

指定 Trident REST 服务器应监听的端口。默认值为 8000。

-rest

启用 REST 接口。默认为真。

Kubernetes 和Trident对象

您可以使用 REST API 通过读取和写入资源对象与 Kubernetes 和Trident进行交互。有几个资源对象决定了 Kubernetes 与Trident、 Trident与存储以及 Kubernetes 与存储之间的关系。其中一些对象通过 Kubernetes 进行管理，另一些对象通过Trident进行管理。

这些物体之间是如何相互作用的?

要了解这些对象、它们的用途以及它们如何交互，最简单的方法或许是跟踪 Kubernetes 用户发出的单个存储请求：

1. 用户创建 `PersistentVolumeClaim`请求一个新的 `PersistentVolume`来自 Kubernetes 的特定大小 `StorageClass`这是管理员之前配置好的。
2. Kubernetes `StorageClass`将其配置器标识为Trident ，并包含告诉Trident如何为请求的类配置卷的参数。
3. Trident审视自身 `StorageClass`名称相同，用于标识匹配项 `Backends`和 `StoragePools`它可以用来为该类配置卷。
4. Trident在匹配的后端配置存储并创建两个对象：a `PersistentVolume`在 Kubernetes 中，它告诉 Kubernetes 如何查找、挂载和处理卷；在Trident中，它维护着两者之间的关系。 `PersistentVolume`以及实际存储。
5. Kubernetes 绑定了 `PersistentVolumeClaim`新的 `PersistentVolume`。包含以下部件的舱体 `PersistentVolumeClaim`将该持久卷挂载到它运行的任何主机上。
6. 用户创建 `VolumeSnapshot`利用现有的PVC管材， `VolumeSnapshotClass`这表明Trident存在问题。
7. Trident识别与 PVC 关联的卷，并在其后端创建该卷的快照。它还创造了一个 `VolumeSnapshotContent`指示 Kubernetes 如何识别快照。
8. 用户可以创建 `PersistentVolumeClaim`使用 `VolumeSnapshot`作为来源。
9. Trident会识别所需的快照，并执行与创建快照相同的步骤。 `PersistentVolume`和 `Volume`。

如需进一步了解 Kubernetes 对象，我们强烈建议您阅读以下内容： "持久卷" Kubernetes 文档的这一部分。

Kubernetes `PersistentVolumeClaim`对象

Kubernetes `PersistentVolumeClaim` object 是 Kubernetes 集群用户发出的存储请求。

除了标准规范之外， Trident还允许用户指定以下卷特定的注释，以便覆盖您在后端配置中设置的默认值：

标注	卷选项	支持的驱动程序
trident.netapp.io/fileSystem	文件系统	ontap-san、solidfire-san、ontap-san-economy
trident.netapp.io/cloneFromPVC	克隆源卷	ontap-nas、ontap-san、solidfire-san、azure-netapp-files、gcp-cvs、ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ontap-nas，ontap-san
trident.netapp.io/protocol	protocol	任何
trident.netapp.io/exportPolicy	出口政策	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	快照策略	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotReserve	快照储备	ontap-nas、ontap-nas-flexgroup 、ontap-san、gcp-cvs
trident.netapp.io/snapshotDirectory	快照目录	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unix权限	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup
trident.netapp.io/blockSize	块大小	solidfire-san

如果创建的PV具有 `Delete`根据回收策略，当 `PV` 被释放时（即用户删除 `PVC` 时）， `Trident`会同时删除 `PV` 和支持卷。如果删除操作失败， `Trident`会将 `PV` 标记为失败，并定期重试该操作，直到操作成功或手动删除 `PV` 为止。如果光伏发电使用 `Retain`策略方面， `Trident`会忽略它，并假定管理员会从 `Kubernetes` 和后端清理它，从而允许在删除卷之前对其进行备份或检查。请注意，删除 `PV` 不会导致`Trident`删除备份卷。您应该使用 `REST API` 将其删除。（`tridentctl`）。

Trident支持使用 CSI 规范创建卷快照：您可以创建卷快照并将其用作数据源来克隆现有的 PVC。这样，就可以将 PV 的特定时间点副本以快照的形式暴露给 Kubernetes。然后可以使用这些快照创建新的PV。看看 `On-Demand Volume Snapshots`看看这种方法是否可行。

Trident还提供 `cloneFromPVC`和 `splitOnClone`用于创建克隆的注释。您可以使用这些注解来克隆 PVC，而无需使用 CSI 实现。

例如：如果用户已经有一个名为 PVC 的 `mysql`用户可以创建一个名为"新`PVC`"的 `mysqlclone`通过使用注释，例如 `trident.netapp.io/cloneFromPVC: mysql`。有了这组注释， Trident会克隆与 mysql PVC 对应的卷，而不是从头开始配置卷。

请考虑以下几点：

- NetApp建议克隆空闲卷。
- PVC 及其克隆应该位于同一个 Kubernetes 命名空间中，并且具有相同的存储类。
- 随着 `ontap-nas`和 `ontap-san`对于驱动程序来说，设置 `PVC` 注释可能是有益的。`trident.netapp.io/splitOnClone`与 `trident.netapp.io/cloneFromPVC`。和 `trident.netapp.io/splitOnClone`设置为 `true`Trident将克隆卷与父卷分离，从而将克隆卷的生命周期与其父卷完全解耦，但代价是损失了一些存储效率。未设置 `trident.netapp.io/splitOnClone`或者将其设置为 `false`这样做可以减少后端空间占用，但代价是在父卷和克隆卷之间创建依赖关系，使得除非先删除克隆卷，否则无法删除父卷。在克隆空数据库卷时，拆分克隆是有意义的，因为预计该卷及其克隆卷将有很大的不同，并且无法从ONTAP提供的存储效率中受益。

这 `sample-input`目录包含可用于Trident的 PVC 定义示例。请参阅有关Trident音量相关参数和设置的完整说明。

Kubernetes `PersistentVolume`对象

Kubernetes `PersistentVolume`该对象代表一块可供 Kubernetes 集群使用的存储空间。它的生命周期与使用它的舱体无关。

> Trident创造 `PersistentVolume`根据其配置的卷，自动将对象注册到 Kubernetes 集群。您无需自行管理它们。

当您创建 PVC 时，指的是基于 Trident 的 `StorageClass`Trident使用相应的存储类配置一个新卷，并为该卷注册一个新的 PV。在配置已配置卷和相应的 PV 时， Trident遵循以下规则：

- Trident会为 Kubernetes 生成一个 PV 名称，以及一个用于配置存储的内部名称。无论哪种情况，名称在其范围内都是独一无二的，这一点都令人放心。
- 容量大小与 PVC 中要求的容量大小尽可能接近，但可能会向上取整到最接近的可分配数量，具体取决于平台。

Kubernetes `StorageClass`对象

Kubernetes `StorageClass`对象通过名称指定。 `PersistentVolumeClaims`为存储配置一组属性。存储类本身标识要使用的配置器，并以配置器能够理解的方式定义该组属性。

它是管理员需要创建和管理的两个基本对象之一。另一个是Trident后端对象。

Kubernetes `StorageClass`使用Trident的对象看起来像这样：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <Name>
provisioner: csi.trident.netapp.io
mountOptions: <Mount Options>
parameters: <Trident Parameters>
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

这些参数是 Trident 特有的，用于告诉Trident如何为该类配置卷。

存储类参数如下：

属性	类型	必填项	描述
属性	映射[字符串]字符串	不可以	请参阅以下属性部分。
存储池	map[string]StringList	不可以	后端名称到存储池列表的映射
附加存储池	map[string]StringList	不可以	后端名称到存储池列表的映射
排除存储池	map[string]StringList	不可以	后端名称到存储池列表的映射

存储属性及其可能的值可以分为存储池选择属性和 Kubernetes 属性。

存储池选择属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	类型	价值观	提供	要求	由.....支持
媒体[1]	string	机械硬盘、混合硬盘、固态硬盘	Pool 包含此类媒体；混合型媒体是指两者兼具。	指定的媒体类型	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup、ontap-san、solidfire-san
供应类型	string	薄的，厚的	池支持这种配置方法	指定的配置方法	厚：全部 ontap；薄：全部 ontap 和 solidfire-san

属性	类型	价值观	提供	要求	由……支持
后端类型	string	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、solidfire-san 、gcp-cvs 、azure-netapp-files、ontap-san-economy	池属于这种类型的后端	指定的后端	所有司机
snapshots	布尔值	真，假	存储池支持带快照的卷	已启用快照的卷	ontap-nas 、ontap-san 、solidfire-san 、gcp-cvs
个克隆	布尔值	真，假	存储池支持卷克隆	已启用克隆的卷	ontap-nas 、ontap-san 、solidfire-san 、gcp-cvs
加密	布尔值	真，假	存储池支持加密卷	已启用加密的卷	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroups、ontap-san
IOPS	整数	正整数	Pool 能够保证在此范围内的 IOPS	容量保证了这些IOPS	solidfire-san

[1]： ONTAP Select系统不支持此系统

在大多数情况下，所请求的值会直接影响配置；例如，请求厚配置会导致厚配置卷的生成。但是，Element 存储池使用其提供的最小和最大 IOPS 来设置 QoS 值，而不是使用请求的值。在这种情况下，请求的值仅用于选择存储池。

理想情况下，你可以使用 `attributes`单独建模，以满足特定类别的需求所需的存储质量。 Trident会自动发现并选择符合所有条件的存储池 `attributes`您指定的。

如果您发现自己无法使用 `attributes`要自动为课程选择合适的池子，您可以使用 `storagePools`和 `additionalStoragePools`参数用于进一步细化池子，甚至选择一组特定的池子。

您可以使用 `storagePools`参数用于进一步限制与任何指定参数匹配的池集合。 `attributes`。换句话说， Trident使用了由以下方式识别的池的交集： `attributes`和 `storagePools`配置参数。您可以单独使用其中一个参数，也可以同时使用两个参数。

您可以使用 `additionalStoragePools`此参数用于扩展Trident用于配置的池集，而不管 Trident 选择的任何池。 `attributes`和 `storagePools`参数。

您可以使用 `excludeStoragePools`用于筛选Trident用于资源配置的池集合的参数。使用此参数会移除所有匹配的池。

在 `storagePools`和 `additionalStoragePools`参数，每个条目都采用以下形式 `<backend>:<storagePoolList>`，在哪里 `<storagePoolList>`是指定后端存储池的逗号分隔列表。例如，一个值 `additionalStoragePools`可能看起来像 `ontapnas_192.168.1.100:aggr1,aggr2;solidfire_192.168.1.101:bronze`。这些列表接受后端值和列表值的正则表达式值。您可以使用 `tridentctl get backend`获取后端及其连接池的列表。

Kubernetes属性

这些属性对Trident在动态配置期间选择存储池/后端没有任何影响。相反，这些属性只是提供 Kubernetes 持久卷支持的参数。工作节点负责文件系统创建操作，可能需要文件系统实用程序，例如 xfsprogs。

属性	类型	价值观	描述	相关驱动因素	**Kubernetes** 版本
文件系统类型	string	ext4、ext3、xfs	块卷的文件系统类型	solidfire-san 、ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、ontap-san-economy	全部
允许卷扩展	布尔值	真，假	启用或禁用对增大PVC尺寸的支持	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、ontap-san-economy、solidfire-san、gcp-cvs、azure-netapp-files	1.11+
容量绑定模式	string	立即，等待首位消费者	选择何时进行卷绑定和动态配置	全部	1.19 - 1.26

- 这 `fsType`该参数用于控制 SAN LUN 所需的文件系统类型。此外，Kubernetes 还利用了以下信息： `fsType`在存储类中表示文件系统存在。可以通过以下方式控制卷所有权： `fsGroup`仅当 `fsType`已设置。请参阅"Kubernetes：为 Pod 或容器配置安全上下文"有关如何使用设置卷所有权的概述 `fsGroup`语境。 Kubernetes 将应用 `fsGroup`仅当满足以下条件时才有值：
  - `fsType`设置在存储类中。
  - PVC接入方式为RWO。

  对于 NFS 存储驱动程序，文件系统已作为 NFS 导出的一部分存在。为了使用 `fsGroup`存储类仍然需要指定一个 `fsType`您可以将其设置为 `nfs`或任何非空值。
- 请参阅"扩大规模"有关扩容的更多详情。
- Trident安装程序包提供了几个示例存储类定义，可供Trident使用。`sample-input/storage-class-*.yaml` 。删除 Kubernetes 存储类会导致相应的Trident存储类也被删除。

Kubernetes `VolumeSnapshotClass`对象

Kubernetes `VolumeSnapshotClass`物体类似于 `StorageClasses`。它们有助于定义多种存储类别，并被卷快照引用，以将快照与所需的快照类别关联起来。每个卷快照都与一个卷快照类相关联。

一个 `VolumeSnapshotClass`应由管理员定义以创建快照。创建卷快照类时，定义如下：

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

这 `driver`向 Kubernetes 指定请求卷快照的 `csi-snapclass`类由Trident处理。这 `deletionPolicy`指定必须删除快照时要执行的操作。什么时候 `deletionPolicy`设置为 `Delete`当删除快照时，存储集群上的卷快照对象以及底层快照也会被删除。或者，将其设置为 `Retain`意味着 `VolumeSnapshotContent`并保留物理快照。

Kubernetes `VolumeSnapshot`对象

Kubernetes `VolumeSnapshot` object 是创建卷快照的请求。 PVC 代表用户对卷的请求，卷快照则是用户对现有 PVC 创建快照的请求。

当收到卷快照请求时， Trident会自动在后端创建卷快照，并通过创建唯一标识符来公开该快照。`VolumeSnapshotContent`目的。您可以从现有的 PVC 创建快照，并在创建新的 PVC 时将这些快照用作数据源。

> VolumeSnapshot 的生命周期与源 PVC 无关：即使源 PVC 被删除，快照仍然会存在。删除具有关联快照的 PVC 时， Trident会将此 PVC 的后备卷标记为“正在删除”状态，但不会将其完全删除。当所有关联的快照都被删除后，该卷也会被移除。

Kubernetes `VolumeSnapshotContent`对象

Kubernetes `VolumeSnapshotContent`该对象表示从已配置的卷中获取的快照。它类似于 `PersistentVolume`表示存储集群上已配置的快照。类似于 `PersistentVolumeClaim`和 `PersistentVolume`当创建快照时，对象会……`VolumeSnapshotContent`对象与……保持一对一的映射关系 `VolumeSnapshot`该对象请求创建快照。

这 `VolumeSnapshotContent`对象包含唯一标识快照的详细信息，例如： `snapshotHandle`。这 `snapshotHandle`是PV名称和名称的独特组合 `VolumeSnapshotContent`目的。

当收到快照请求时， Trident会在后端创建快照。快照创建完成后， Trident会进行配置。`VolumeSnapshotContent`对象，从而将快照暴露给 Kubernetes API。

> 通常情况下，你不需要管理 `VolumeSnapshotContent`目的。但也有例外情况，比如你想……"导入卷快照"在Trident之外创建。

Kubernetes `VolumeGroupSnapshotClass`对象

Kubernetes `VolumeGroupSnapshotClass`物体类似于 `VolumeSnapshotClass`。它们有助于定义多种存储类别，并被卷组快照引用，以将快照与所需的快照类别关联起来。每个卷组快照都与一个卷组快照类相关联。

一个 `VolumeGroupSnapshotClass`应由管理员定义以创建快照组。使用以下定义创建卷组快照类：

```
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
```

这 `driver`向 Kubernetes 指定请求卷组快照的权限。 `csi-group-snap-class`类由Trident处理。这 `deletionPolicy`指定当必须删除组快照时要执行的操作。什么时候 `deletionPolicy`设置为 `Delete`当删除快照时，卷组快照对象以及存储集群上的基础快照也会被删除。或者，将其设置为 `Retain`意味着 `VolumeGroupSnapshotContent`并保留物理快照。

Kubernetes `VolumeGroupSnapshot`对象

Kubernetes `VolumeGroupSnapshot`该对象是创建多个卷的快照的请求。就像 PVC 代表用户对卷的请求一样，卷组快照是用户对现有 PVC 创建快照的请求。

当收到卷组快照请求时， Trident会自动在后端管理卷的组快照创建，并通过创建唯一标识符来公开该快照。 `VolumeGroupSnapshotContent`目的。您可以从现有的 PVC 创建快照，并在创建新的 PVC 时将这些快照用作数据源。

> VolumeGroupSnapshot 的生命周期与源 PVC 无关：即使源 PVC 被删除，快照仍然会保留。删除具有关联快照的 PVC 时， Trident会将此 PVC 的后备卷标记为“正在删除”状态，但不会将其完全删除。当所有关联的快照都被删除时，卷组快照也会被删除。

Kubernetes `VolumeGroupSnapshotContent`对象

Kubernetes `VolumeGroupSnapshotContent`该对象表示从已配置的卷中获取的组快照。它类似于 `PersistentVolume`表示存储集群上已配置的快照。类似于 `PersistentVolumeClaim`和 `PersistentVolume`当创建快照时，对象会…… `VolumeSnapshotContent`对象与……保持一对一的映射关系 `VolumeSnapshot`该对象请求创建快照。

这 `VolumeGroupSnapshotContent`对象包含用于标识快照组的详细信息，例如： `volumeGroupSnapshotHandle`以及存储系统上存在的各个 volumeSnapshotHandles。

当收到快照请求时， Trident会在后端创建卷组快照。卷组快照创建完成后， Trident会进行配置。 `VolumeGroupSnapshotContent`对象，从而将快照暴露给 Kubernetes API。

Kubernetes `CustomResourceDefinition`对象

Kubernetes 自定义资源是 Kubernetes API 中的端点，由管理员定义，用于对类似对象进行分组。 Kubernetes 支持创建自定义资源来存储对象集合。您可以通过运行以下命令来获取这些资源定义。 `kubectl get crds` 。

Kubernetes 将自定义资源定义 (CRD) 及其关联的对象元数据存储在其元数据存储中。这样就无需为Trident单独开设商店了。

Trident的使用 `CustomResourceDefinition`用于保存Trident对象标识的对象，例如Trident后端、 Trident存储类和Trident卷。这些对象由Trident管理。此外，CSI 卷快照框架引入了一些定义卷快照所需的 CRD。

CRD 是 Kubernetes 的一种构造。上述资源的对象由Trident创建。举个简单的例子，当使用以下方式创建后端时 `tridentctl`相应的 `tridentbackends`创建 CRD 对象供 Kubernetes 使用。

关于Trident的CRD，需要记住以下几点：

- 安装Trident时，会创建一组 CRD，可以像使用任何其他资源类型一样使用这些 CRD。
- 使用以下方式卸载Trident时 `tridentctl uninstall`使用命令后， Trident pod 会被删除，但创建的 CRD 不会被清理。请参阅"卸载Trident"了解如何将Trident完全移除并从头开始重新配置。

Trident `StorageClass`对象

Trident为 Kubernetes 创建匹配的存储类 `StorageClass`指定对象 `csi.trident.netapp.io`在他们的供应领域。存储类名称与 Kubernetes 的名称匹配。 `StorageClass`它所代表的对象。

> 使用 Kubernetes 时，这些对象会在 Kubernetes 集群启动时自动创建。 `StorageClass`已注册使用Trident作为配置器的配置器。

存储类别包含对卷的一系列要求。 Trident会将这些要求与每个存储池中存在的属性进行匹配；如果匹配，则该存储池是使用该存储类别配置卷的有效目标。

您可以使用 REST API 创建存储类配置，直接定义存储类。但是，对于 Kubernetes 部署，我们期望在注册新的 Kubernetes 实例时创建它们。 `StorageClass`物体。

Trident后端对象

后端代表存储提供商， Trident在其上配置卷；单个Trident实例可以管理任意数量的后端。

> 这是您可以自行创建和管理的两种对象类型之一。另一个是 Kubernetes。 `StorageClass`目的。

有关如何构建这些对象的更多信息，请参阅："配置后端" 。

Trident `StoragePool`对象

存储池代表每个后端可用于配置的不同位置。对于ONTAP而言，这些对应于 SVM 中的聚合。对于NetApp HCI/SolidFire，这些对应于管理员指定的 QoS 频段。对于Cloud Volumes Service，这些对应于云提供商区域。每个存储池都有一组独特的存储属性，这些属性定义了其性能特征和数据保护特征。

与此处的其他对象不同，存储池候选对象始终会自动发现和管理。

Trident `Volume`对象

卷是配置的基本单元，包括后端端点（例如 NFS 共享）以及 iSCSI 和 FC LUN。在 Kubernetes 中，这些直接对应于 `PersistentVolumes`。创建卷时，请确保它具有存储类（决定卷的部署位置）和大小。

- 在 Kubernetes 中，这些对象是自动管理的。您可以查看这些信息，了解Trident 的部署情况。
- 删除带有关联快照的 PV 时，相应的Trident卷将更新为 正在删除 状态。要删除Trident卷，您应该删除该卷的快照。

卷配置定义了已配置卷应具有的属性。

属性	类型	必填项	描述
version	string	不可以	Trident API 版本（“1”）
name	string	可以	要创建的卷的名称
存储类	string	可以	配置卷时要使用的存储类
大小	string	可以	要配置的卷的大小（以字节为单位）
protocol	string	不可以	使用的协议类型：“文件”或“块”
内部名称	string	不可以	存储系统中对象的名称；由Trident生成
克隆源卷	string	不可以	ontap（nas、san）和 solidfire-*：要克隆的卷的名称
splitOnClone	string	不可以	ontap（nas，san）：将克隆体从其父体中分离出来
快照策略	string	不可以	ontap-*：要使用的快照策略
快照储备	string	不可以	ontap-*：为快照预留的卷百分比
出口政策	string	不可以	ontap-nas*：要使用的导出策略
快照目录	布尔值	不可以	ontap-nas*：快照目录是否可见
unix权限	string	不可以	ontap-nas*：初始 UNIX 权限
块大小	string	不可以	solidfire-*：块/扇区大小
文件系统	string	不可以	文件系统类型

Trident生成 `internalName`创建卷时。这包括两个步骤。首先，它会添加存储前缀（默认值）。`trident`或后端配置中的前缀）添加到卷名称，从而得到如下形式的名称 `<prefix>-<volume-name>`。然后它会对名称进行清理，替换后端不允许的字符。对于ONTAP后端，它会将连字符替换为下划线（因此，内部名称变为 `<prefix>_<volume-name>`）。对于 Element 后端，它会将下划线替换为连字符。

您可以使用卷配置通过 REST API 直接配置卷，但在 Kubernetes 部署中，我们预计大多数用户将使用标准的 Kubernetes 管理配置。 `PersistentVolumeClaim`方法。 Trident会在配置过程中自动创建此卷对象。

Trident `Snapshot`对象

快照是卷在特定时间点的副本，可用于配置新卷或恢复状态。在 Kubernetes 中，这些直接对应于 `VolumeSnapshotContent`物体。每个快照都与一个卷相关联，该卷是快照数据的来源。

每个 `Snapshot`对象包含以下属性：

属性	类型	必填项	描述
version	字符串	是	Trident API 版本（“1”）
name	字符串	是	Trident快照对象的名称
内部名称	字符串	是	存储系统上Trident快照对象的名称
volumeName	字符串	是	创建快照的持久卷的名称
volumeInternalName	字符串	是	存储系统上关联的Trident卷对象的名称

在 Kubernetes 中，这些对象是自动管理的。您可以查看这些信息，了解Trident 的部署情况。

当 Kubernetes `VolumeSnapshot`创建对象请求后， `Trident` 的工作原理是在后端存储系统上创建快照对象。这 `internalName`此快照对象是通过组合前缀生成的。 `snapshot-`和 `UID`的 `VolumeSnapshot`对象（例如， `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。 `volumeName`和 `volumeInternalName`通过获取支持卷的详细信息来填充。

Trident `ResourceQuota`目的

Trident守护进程消耗一个 `system-node-critical`优先级等级——Kubernetes 中可用的最高优先级等级——以确保Trident能够在节点优雅关闭期间识别和清理卷，并允许Trident daemonset pod 在资源压力高的集群中抢占优先级较低的工作负载。

为了实现这一目标， Trident采用了一种 `ResourceQuota`确保Trident守护进程集上的“system-node-critical”优先级类得到满足。在部署和创建守护进程集之前， Trident会查找 `ResourceQuota`对象，如果未发现，则应用它。

如果您需要对默认资源配额和优先级类别进行更多控制，您可以生成一个 `custom.yaml`或配置 `ResourceQuota`使用 Helm Chart 的对象。

以下是一个 `ResourceQuota` 对象优先考虑Trident守护进程集的示例。

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values:
          - system-node-critical
```

有关资源配额的更多信息，请参阅"Kubernetes：资源配额"。

清理 `**ResourceQuota**`如果安装失败

在极少数情况下，如果安装失败， `ResourceQuota`对象已创建，首次尝试"卸载"然后重新安装。

如果这样不行，就手动删除。 `ResourceQuota`目的。

消除 `ResourceQuota`

如果您希望自行控制资源分配，您可以移除Trident。 `ResourceQuota`使用以下命令对象：

```
kubectl delete quota trident-csi -n trident
```

Pod 安全标准 (PSS) 和安全上下文约束 (SCC)

Kubernetes Pod 安全标准 (PSS) 和 Pod 安全策略 (PSP) 定义了权限级别并限制了 Pod 的行为。 OpenShift 安全上下文约束 (SCC) 类似地定义了 OpenShift Kubernetes Engine 特有的 pod 限制。为了实现这种自定义功能， Trident会在安装过程中启用某些权限。以下各节详细介绍了Trident设置的权限。

PSS 取代了 Pod 安全策略 (PSP)。 PSP 在 Kubernetes v1.21 中已被弃用，并将于 v1.25 中移除。更多信息，请参阅"Kubernetes：安全性"。

必需的 **Kubernetes** 安全上下文和相关字段

权限	描述
特权	CSI 要求挂载点是双向的，这意味着Trident节点 pod 必须运行特权容器。更多信息，请参阅"Kubernetes：挂载传播"。

权限	描述
主机网络	iSCSI守护进程需要它。 `iscsiadm`管理 iSCSI 挂载点，并使用主机网络与 iSCSI 守护进程通信。
主机 IPC	NFS 使用进程间通信 (IPC) 与 NFSD 进行通信。
主机 PID	开始需要 `rpc-statd`适用于 NFS。 Trident会查询主机进程以确定是否 `rpc-statd`在挂载 NFS 卷之前运行。

功能	这 `SYS_ADMIN`该功能作为特权容器的默认功能的一部分提供。例如，`Docker` 为特权容器设置了以下功能：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

赛康普	在特权容器中，Seccomp 配置文件始终为“Unconfined”；因此，它无法在Trident中启用。
SELinux	在 OpenShift 上，特权容器运行在 `spc_t`（“超级特权容器”）域，而非特权容器则运行在……域中。`container_t`领域。在 `containerd`， 和 `container-selinux`安装完成后，所有容器都在运行。`spc_t`域，这实际上禁用了 SELinux。因此， Trident不会增加 `seLinuxOptions`到容器中。
DAC	特权容器必须以 root 用户身份运行。非特权容器以 root 用户身份运行，以访问 CSI 所需的 Unix 套接字。

舱体安全标准 (PSS)

标签	描述	默认
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	允许将Trident控制器和节点添加到安装命名空间中。请勿更改命名空间标签。	`enforce: privileged enforce-version: <version of the current cluster or highest version of PSS tested.>`

更改命名空间标签可能会导致 Pod 无法调度，出现“创建时出错：…”或“警告：trident-csi-…”等错误。如果发生这种情况，请检查命名空间标签是否为 `privileged`已更改。如果是这样，请重新安装Trident。

Pod 安全策略 (PSP)

字段	描述	默认
`allowPrivilegeEscalation`	特权容器必须允许权限提升。	`true`
`allowedCSIDrivers`	Trident不使用内联 CSI 临时卷。	空
`allowedCapabilities`	非特权Trident容器不需要比默认集更多的功能，而特权容器将被授予所有可能的功能。	空

字段	描述	默认
allowedFlexVolumes	Trident不使用"FlexVolume驱动器"因此，它们不包含在允许的音量列表中。	空
allowedHostPaths	Trident节点 pod 会挂载节点的根文件系统，因此设置此列表没有任何好处。	空
allowedProcMountTypes	Trident不使用任何 ProcMountTypes。	空
allowedUnsafeSysctls	Trident不需要任何不安全措施 sysctls。	空
defaultAddCapabilities	特权容器无需添加任何功能。	空
defaultAllowPrivilegeEscalation	权限提升权限是在每个Trident pod中处理的。	false
forbiddenSysctls	不 `sysctls`允许。	空
fsGroup	Trident容器以root权限运行。	RunAsAny
hostIPC	挂载 NFS 卷需要主机 IPC 与主机通信 nfsd	true
hostNetwork	iscsiadm 需要主机网络才能与 iSCSI 守护进程通信。	true
hostPID	需要主机 PID 来进行检查 `rpc-statd`正在节点上运行。	true
hostPorts	Trident不使用任何主机端口。	空
privileged	Trident节点 pod 必须运行特权容器才能挂载卷。	true
readOnlyRootFilesystem	Trident节点 pod 必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点 pod 运行的是特权容器，不能放弃任何功能。	none
runAsGroup	Trident容器以root权限运行。	RunAsAny
runAsUser	Trident容器以root权限运行。	runAsAny
runtimeClass	Trident不使用 RuntimeClasses。	空
seLinux	Trident不设置 `seLinuxOptions`因为目前容器运行时和 Kubernetes 发行版在处理 SELinux 方面存在差异。	空
supplementalGroups	Trident容器以root权限运行。	RunAsAny
volumes	Trident pods 需要这些音量插件。	hostPath, projected, emptyDir

安全上下文约束 (SCC)

标签	描述	默认
allowHostDirVolumePlugin	Trident节点 pod 会挂载节点的根文件系统。	true
allowHostIPC	挂载 NFS 卷需要主机 IPC 与主机通信 nfsd。	true
allowHostNetwork	iscsiadm 需要主机网络才能与 iSCSI 守护进程通信。	true
allowHostPID	需要主机 PID 来进行检查 `rpc-statd`正在节点上运行。	true
allowHostPorts	Trident不使用任何主机端口。	false
allowPrivilegeEscalation	特权容器必须允许权限提升。	true
allowPrivilegedContainer	Trident节点 pod 必须运行特权容器才能挂载卷。	true
allowedUnsafeSysctls	Trident不需要任何不安全措施 sysctls。	none
allowedCapabilities	非特权Trident容器不需要比默认集更多的功能，而特权容器将被授予所有可能的功能。	空
defaultAddCapabilities	特权容器无需添加任何功能。	空
fsGroup	Trident容器以root权限运行。	RunAsAny
groups	此 SCC 专用于Trident ，并与其用户绑定。	空
readOnlyRootFilesystem	Trident节点 pod 必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点 pod 运行的是特权容器，不能放弃任何功能。	none
runAsUser	Trident容器以root权限运行。	RunAsAny
seLinuxContext	Trident不设置 `seLinuxOptions`因为目前容器运行时和 Kubernetes 发行版在处理 SELinux 方面存在差异。	空
seccompProfiles	特权容器始终以“非限制”模式运行。	空
supplementalGroups	Trident容器以root权限运行。	RunAsAny
users	提供了一个条目，用于将此 SCC 绑定到Trident命名空间中的Trident用户。	不适用
volumes	Trident pods 需要这些音量插件。	hostPath, downwardAPI, projected, emptyDir

法律声明

法律声明提供对版权声明、商标、专利等的访问。

版权

"https://www.netapp.com/company/legal/copyright/"

商标

NETAPP、NETAPP 徽标和NetApp商标页面上列出的标志是NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。

"https://www.netapp.com/company/legal/trademarks/"

专利

NetApp拥有的专利的最新列表可以在以下位置找到：

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

隐私政策

"https://www.netapp.com/company/legal/privacy-policy/"

开源

您可以在每个版本的通知文件中查看NetApp Trident软件中使用的第三方版权和许可信息。 https://github.com/NetApp/trident/ 。