Trident 25.10 文档

Trident

NetApp
March 04, 2026

目录

Trident 25.10 文档

发行说明

新增功能

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

> 安装程序 zip 文件中提供的 Linux `tridentctl` 二进制文件是经过测试并受支持的版本。请注意，zip 文件中 `macos` 部分提供的 `/extras` 二进制文件未经测试且不受支持。

25.10 中的新增功能

了解 Trident 和 Trident Protect 中的新增功能，包括增强功能、修复和弃用。

Trident

增强功能

- **Kubernetes**：
  - 除了 ONTAP-SAN（iSCSI 和 FC）之外，还增加了对用于 ONTAP-NAS NFS 和 ONTAP-SAN-Economy 驱动程序的 v1beta1 Volume Group Snapshot Kubernetes API 的 CSI Volume Group Snapshots 的支持。请参阅"处理卷组快照"。
  - 添加了对 ONTAP-NAS 和 ONTAP-NAS-Economy（两个 NAS 驱动程序中的 SMB 除外）以及 ONTAP-SAN 和 ONTAP-SAN-Economy 驱动程序的带强制卷分离的自动工作负载故障转移支持。请参阅 "使用 Trident 自动化有状态应用程序的故障转移"。
  - 增强的 Trident 节点并发性，为 FCP 卷的节点操作提供更高的可扩展性。
  - 添加了对 ONTAP NAS 驱动程序的 ONTAP AFX 支持。请参阅 "ONTAP NAS 配置选项和示例"。
  - 增加了对通过 TridentOrchestrator CR 和 Helm 图表值配置 Trident 容器的 CPU 和内存资源请求和限制的支持。("问题 #1000", "问题 #927", "问题 #853", "问题 #592", "问题 #110")。
  - 添加了对 ASAr2 个性的 FC 支持。请参阅 "ONTAP SAN 配置选项和示例"。
  - 添加了使用 HTTPS 而不是 HTTP 提供 Prometheus 指标的选项。请参阅 "监控 Trident"。
  - 在导入卷时添加了一个选项 `--no-rename`以保留原始名称，但让 Trident 管理卷的生命周期。请参阅"导入卷"。
  - Trident 部署现在以系统集群关键优先级类运行。
- 添加了 Trident 控制器通过 helm、operator 和 tridentctl ("问题 #858") 使用主机网络的选项。
- 为 ANF 驱动程序添加了手动 QoS 支持，使其在 Trident 25.10 中可投入生产；此实验性增强在 Trident 25.06 中引入。

实验性增强功能

> 不适用于生产环境。

- **[技术预览]**：除了 ONTAP-SAN 驱动程序的现有技术预览（统一 ONTAP 9 中的 iSCSI 和 FCP 协议）之外，还添加了对 ONTAP-NAS（仅限 NFS）和 ONTAP-SAN（统一 ONTAP 9 的 NVMe）并发的支持。

修复

- **Kubernetes**：
  - 通过将 Linux DaemonSet 标准化为 node-driver-registrar 以匹配 Windows DaemonSet 和容器映像命名，修复了 CSI node-driver-registrar 容器名称不一致的问题。
  - 修复了未正确升级旧版 qtree 导出策略的问题。
- **Openshift**：
  - 修复了由于 SCC 将 allowHostDirVolumePlugin 设置为 false ("问题 #950")，Trident 节点 pod 在 Openshift 中的 Windows 节点上无法启动的问题。
- 修复了未通过 Helm ("问题 #975") 设置 Kubernetes API QPS 的问题。
- 修复了无法在同一 Kubernetes 节点上挂载基于 NVMe 的 XFS 文件系统 PVC 快照的持久卷声明 (PVC) 的问题。
- 通过为每个后端添加唯一/共享子系统名称（例如，netappdvp_subsystem），修复了主机/Docker 在 NDVP 模式下重新启动后的 UUID 更改问题。
- 修复了 Trident 从 23.10 之前的版本升级到 24.10 及更高版本期间 iSCSI 卷的挂载错误，解决了"invalid SANType"问题。
- 修复了在未重新启动 Trident 控制器的情况下，Trident 后端状态未转换为在线/离线的问题。
- 修复了导致 PVC 调整大小缓慢的间歇性竞争条件。
- 修复了在卷克隆失败时快照未被清理的问题。
- 修复了内核更改其设备路径时取消暂存卷的故障。
- 修复了由于 LUKS 设备已关闭而无法取消暂存卷的问题。
- 修复了存储操作缓慢导致 ContextDeadline 错误的问题。
- Trident Operator 将等待可配置的 k8s-timeout 来检查 Trident 版本。

Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强由 NetApp ONTAP 存储系统和 NetApp Trident CSI 存储配置程序支持的有状态 Kubernetes 应用程序的功能和可用性。

增强功能

- 添加了用于控制计划和备份 CR 的快照 CR 超时的注释：
  - `protect.trident.netapp.io/snapshot-completion-timeout`
  - `protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout`
  - `protect.trident.netapp.io/volume-snapshots-created-timeout`

    请参阅 "支持的备份和计划注释"。
- 向计划 CR 添加了注释以配置 PVC 绑定超时，该超时将由备份 CR 使用：`protect.trident.netapp.io/pvc-bind-timeout-sec`。请参阅 "支持的备份和计划注释"。
- 通过新字段改进 `tridentctl-protect`备份和快照列表，以指示执行挂钩失败。

25.06.2 中的变化

Trident

修复

- **Kubernetes**：修复了从 Kubernetes 节点分离卷时发现不正确 iSCSI 设备的关键问题。

25.06.1 中的变化

Trident

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

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 设备导致分离卷失败的问题。
  - 修复了设备路径自创建以来发生更改时 iSCSI 卷取消暂存的问题。
  - 阻止跨存储类克隆卷。
- **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 ("问题 #1008") 克隆卷时的超时时间。

25.06 的变化

Trident

增强功能

- **Kubernetes**：
  - 为 ONTAP-SAN iSCSI 驱动程序添加了对带有 `v1beta1` Volume Group Snapshot Kubernetes API 的 CSI Volume Group Snapshot 的支持。请参阅"处理卷组快照"。

    > VolumeGroupSnapshot 是 Kubernetes 中带有 beta API 的 beta 功能。Kubernetes 1.32 是 VolumeGroupSnapshot 所需的最低版本。

  - 除了 iSCSI 之外，还增加了对 NVMe/TCP 的 ONTAP ASA r2 的支持。请参阅 "ONTAP SAN 配置选项和示例"。
  - 增加了对 ONTAP-NAS 和 ONTAP-NAS-Economy 卷的安全 SMB 支持。Active Directory 用户和组现在可以与 SMB 卷一起使用，以增强安全性。请参阅 "启用安全 SMB"。
  - 增强的 Trident 节点并发性，可在 iSCSI 卷的节点操作上实现更高的可扩展性。
  - 打开 LUKS 卷时添加 `--allow-discards`，以允许丢弃/TRIM 命令进行空间回收。
  - 增强了格式化 LUKS 加密卷时的性能。

- 针对失败但部分格式化的 LUKS 设备的增强型 LUKS 清理。
- 增强了 NVMe 卷连接和分离的 Trident 节点幂等性。
- 为 ONTAP-SAN-Economy 驱动程序的 Trident 卷配置添加了 `internalID`字段。
- 添加了对 NVMe 后端使用 SnapMirror 进行卷复制的支持。请参阅 "使用 SnapMirror 复制卷"。

实验性增强功能

> 不适用于生产环境。

- [Tech Preview] 通过 `--enable-concurrency` 功能标志启用并发 Trident 控制器操作。这允许控制器操作并行运行，从而提高繁忙或大型环境的性能。

> 此功能是实验性的，目前支持使用 ONTAP-SAN 驱动程序（iSCSI 和 FCP 协议）的有限并行工作流。

- [Tech Preview] 添加了 ANF 驱动程序的手动 QOS 支持。

修复

- **Kubernetes**：
  - 已修复 CSI NodeExpandVolume 中的一个问题，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。
  - 修复了无法清除 ONTAP-NAS 和 ONTAP-NAS-Economy 驱动程序的重复导出策略的问题。
  - 修复了未设置 `nfsMountOptions`时 GCNV 卷默认为 NFSv3 的问题；现在支持 NFSv3 和 NFSv4 协议。如果未提供 `nfsMountOptions`，将使用主机的默认 NFS 版本（NFSv3 或 NFSv4）。
  - 修复了使用 Kustomize ("问题 #831") 安装 Trident 时的部署问题。
  - 修复了从快照创建的 PVC 丢失的导出策略("问题 #1016")。
  - 修复了 ANF 卷大小未自动对齐为 1 GiB 增量的问题。
  - 修复了将 NFSv3 与 Bottlerocket 配合使用时的问题。
- 修复了 ONTAP-NAS-Economy 卷在调整大小失败的情况下扩展到 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**：将最低支持的 Kubernetes 更新为 v1.27。

Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强由 NetApp ONTAP 存储系统和 NetApp Trident CSI 存储配置程序支持的有状态 Kubernetes 应用程序的功能和可用性。

增强功能

- 改进了还原时间，提供了更频繁的完整备份选项。

- 使用 Group-Version-Kind (GVK) 过滤提高了应用程序定义和选择性恢复的粒度。
- 使用 AppMirrorRelationship (AMR) 与 NetApp SnapMirror 时的高效重新同步和反向复制，以避免完整的 PVC 复制。
- 添加了使用 EKS Pod Identity 创建 AppVault 存储桶的功能，无需为 EKS 集群的存储桶凭据指定密钥。
- 如果需要，添加了跳过还原命名空间中的标签和注释的功能。
- AppMirrorRelationship (AMR) 现在将检查源 PVC 扩展，并根据需要在目标 PVC 上执行适当的扩展。

修复

- 修复了将先前快照中的快照注释值应用于较新快照的错误。所有快照注释现在都已正确应用。
- 默认情况下，定义了数据移动器加密（Kopia / Restic）的密钥（如果未定义）。
- 为 S3 appvault 创建添加了改进的验证和错误消息。
- AppMirrorRelationship (AMR) 现在仅复制处于 Bound 状态的 PV，以避免尝试失败。
- 修复了在具有大量备份的 AppVault 上获取 AppVaultContent 时显示错误的问题。
- KubeVirt VMSnapshots 被排除在还原和故障转移操作之外，以避免故障。
- 修复了 Kopia 的问题，由于 Kopia 默认保留计划覆盖了用户在计划中设置的内容，因此提前删除了快照。

25.02.1 中的变更

Trident

修复

- **Kubernetes**：
  - 修复了 trident-operator 中使用非默认映像注册表时错误填充 sidecar 映像名称和版本的问题（"问题 #983"）。
  - 修复了在 ONTAP 故障转移回馈期间多路径会话无法恢复的问题（"问题 #961"）。

25.02 的变化

从 Trident 25.02 开始，What's New 摘要提供了有关 Trident 和 Trident Protect 版本的增强、修复和弃用的详细信息。

Trident

增强功能

- **Kubernetes**：
  - 增加了对 iSCSI 的 ONTAP ASA r2 支持。
  - 添加了对非优雅节点关闭场景期间 ONTAP-NAS 卷的强制分离的支持。新的 ONTAP-NAS 卷现在将利用由 Trident 管理的每卷导出策略。为现有卷提供升级路径，以便在取消发布时过渡到新的导出策略模型，而不会影响活动工作负载。
  - 已添加 cloneFromSnapshot 标注。

- 添加了对跨命名空间卷克隆的支持。
- 增强的 iSCSI 自我修复扫描修正，可按确切的主机、通道、目标和 LUN ID 启动重新扫描。
- 增加了对 Kubernetes 1.32 的支持。

- **OpenShift**：
  - 增加了对 ROSA 集群上 RHCOS 的自动 iSCSI 节点准备的支持。
  - 增加了对 ONTAP 驱动程序 OpenShift 虚拟化的支持。
- 在 ONTAP-SAN 驱动程序上添加了光纤通道支持。
- 添加了 NVMe LUKS 支持。
- 已切换到所有基础镜像的 scratch 镜像。
- 添加了 iSCSI 连接状态发现和日志记录，用于 iSCSI 会话应登录但未登录的情况（"问题 #961"）。
- 使用 google-cloud-netapp-volumes 驱动程序添加了对 SMB 卷的支持。
- 添加了允许 ONTAP 卷在删除时跳过恢复队列的支持。
- 添加了使用 SHA 而不是 tag 覆盖默认镜像的支持。
- 向 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少的节点 IP 地址("问题 #965")。
  - 修复了 ONTAP-NAS-Economy 的自动导出策略过早切换到每个卷策略的问题。
  - 修复了后端配置凭据，以支持所有可用的 AWS ARN 分区（"问题 #913"）。
  - 在 Trident 运算符中添加了禁用自动配置器对账的选项（"问题 #924"）。
  - 为 csi-resizer 容器添加了 securityContext ("问题 #976")。

Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强由 NetApp ONTAP 存储系统和 NetApp Trident CSI 存储配置程序支持的有状态 Kubernetes 应用程序的功能和可用性。

增强功能

- 为 KubeVirt / OpenShift Virtualization VM 添加了备份和还原支持，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，并增强了使用 NetApp SnapMirror 和 Trident Protect 复制存储时的现有保护功能。
- 添加了为 Kubevirt 环境在应用程序级别控制冻结行为的功能。
- 添加了对配置 AutoSupport 代理连接的支持。
- 添加了为数据移动器加密（Kopia / Restic）定义密钥的功能。
- 添加了手动运行执行挂钩的功能。
- 添加了在 Trident Protect 安装期间配置安全上下文约束 (SCC) 的功能。
- 添加了对在 Trident Protect 安装期间配置 nodeSelector 的支持。

- 添加了对 AppVault 对象的 HTTP / HTTPS 出口代理的支持。
- 扩展 ResourceFilter 以启用排除集群范围内的资源。
- 在 S3 AppVault 凭据中添加了对 AWS 会话令牌的支持。
- 增加了对快照前执行挂钩后资源收集的支持。

修复

- 改进了临时卷的管理以跳过 ONTAP 卷恢复队列。
- SCC 注释现已还原为原始值。
- 通过支持并行操作提高还原效率。
- 增强了对大型应用程序执行挂钩超时的支持。

24.10.1 中的变更

增强功能

- **Kubernetes**：增加了对 Kubernetes 1.32 的支持。
- 添加了 iSCSI 连接状态发现和日志记录，用于 iSCSI 会话应登录但未登录的情况（"问题 #961"）。

修复

- 修复了自动导出策略中缺少的节点 IP 地址("问题 #965")。
- 修复了 ONTAP-NAS-Economy 的自动导出策略过早切换到每个卷策略的问题。
- 更新了 Trident 和 Trident-ASUP 依赖项，以解决 CVE-2024-45337 和 CVE-2024-45310。
- 删除了 iSCSI 自我修复期间间歇性不健康的非 CHAP 门户的注销（"问题 #961"）。

24.10 中的变化

增强功能

- Google Cloud NetApp Volumes 驱动程序现已正式可用于 NFS 卷，并支持区域感知配置。
- GCP Workload Identity 将用作使用 GKE 的 Google Cloud NetApp Volumes 的云标识。
- 向 ONTAP-SAN 和 ONTAP-SAN-Economy 驱动程序添加了 `formatOptions`配置参数，以允许用户指定 LUN 格式选项。
- 将 Azure NetApp Files 最小卷大小减少到 50 GiB。Azure 新的最小尺寸预计将于 11 月正式推出。
- 添加了 `denyNewVolumePools` 配置参数，以将 ONTAP-NAS-Economy 和 ONTAP-SAN-Economy 驱动程序限制为预先存在的 FlexVol 池。
- 添加了对所有 ONTAP 驱动程序中从 SVM 添加、删除或重命名聚合的检测。
- 为 LUKS LUN 增加了 18 MiB 的开销，以确保报告的 PVC 尺寸可用。
- 改进了 ONTAP-SAN 和 ONTAP-SAN-Economy 节点阶段和取消阶段错误处理，以允许取消阶段在阶段失败后删除设备。
- 添加了一个自定义角色生成器，允许客户在 ONTAP 中为 Trident 创建最小化角色。

- 添加了用于故障排除的额外日志记录 `lsscsi`（"问题 #792"）。

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
  - 应用程序移动性

    "详细了解 Trident Protect"。
- 向安装程序添加了一个新标志 `--k8s-api-qps`，以设置 Trident 与 Kubernetes API 服务器通信时使用的 QPS 值。
- 为安装程序添加了 `--node-prep`标记，用于自动管理 Kubernetes 集群节点上的存储协议依赖关系。与 Amazon Linux 2023 iSCSI 存储协议的兼容性经过测试和验证
- 添加了对非优雅节点关闭场景期间 ONTAP-NAS-Economy 卷的强制分离的支持。
- 使用 `autoExportPolicy` 后端选项时，新的 ONTAP-NAS-Economy NFS 卷将使用每 qtree 导出策略。Qtree 仅在发布时映射到节点限制导出策略，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，而不会影响活动的工作负载。
- 增加了对 Kubernetes 1.31 的支持。

实验性增强功能

- 为 ONTAP-SAN 驱动程序上的光纤通道支持添加了技术预览。

修复

- **Kubernetes**：
  - 修复了阻止 Trident Helm 安装的 Rancher 准入 webhook（"问题 #839"）。
  - 修复了 helm chart values ("问题 #898") 中的 Affinity 键。
  - 已修复 tridentControllerPluginNodeSelector/tridentNodePluginNodeSelector 无法使用 "true" 值 ("问题 #899")。
  - 已删除在克隆期间创建的短暂快照 ("问题 #901")。
- 添加了对 Windows Server 2019 的支持。
- 修复了 Trident repo ("问题 #767") 中的 `go mod tidy` 问题。

弃用

- **Kubernetes**：
  - 将最低支持的 Kubernetes 更新为 1.25。
  - 已删除对 POD Security Policy 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 的变化

增强功能

- 重要提示： `limitVolumeSize` 参数现在限制了 ONTAP 经济驱动程序中的 qtree/LUN 大小。使用新的 `limitVolumePoolSize` 参数来控制这些驱动程序中的 FlexVol 大小。("问题 #341")。
- 添加了 iSCSI 自我修复功能，以便在使用已弃用的 igroup 时根据确切的 LUN ID 启动 SCSI 扫描（"问题 #883"）。
- 添加了对卷克隆和调整大小操作的支持，即使在后端处于挂起模式时也允许。
- 添加了将 Trident 控制器的用户配置日志设置传播到 Trident 节点 Pod 的功能。
- 在 Trident 中添加了对 ONTAP 版本 9.15.1 及更高版本默认使用 REST 而不是 ONTAPI (ZAPI) 的支持。
- 为新的永久卷在 ONTAP 存储后端上添加了对自定义卷名和元数据的支持。
- 增强了 `azure-netapp-files` (ANF) 驱动程序，以便在 NFS 挂载选项设置为使用 NFS 版本 4.x 时，默认情况下自动启用快照目录。
- 已添加对 NFS 卷的 Bottlerocket 支持。
- 添加了对 Google Cloud NetApp Volumes 的技术预览支持。

Kubernetes

- 增加了对 Kubernetes 1.30 的支持。
- 添加了 Trident DaemonSet 在启动时清理僵尸挂载和残留跟踪文件的功能（"问题 #883"）。
- 添加了 PVC 注释 `trident.netapp.io/luksEncryption`用于动态导入 LUKS 卷（"问题 #849"）。
- 为 ANF 驱动程序添加了拓扑感知。
- 添加了对 Windows Server 2022 节点的支持。

修复

- 修复了由于事务过时而导致 Trident 安装失败的问题。
- 修复了 tridentctl 忽略来自 Kubernetes ("问题 #892") 的警告消息。
- 将 Trident 控制器 `SecurityContextConstraint`优先级更改为 `0`（"问题 #887"）。
- ONTAP 驱动程序现在接受低于 20 MiB 的卷大小("问题[#885")。
- 修复了 Trident 以防止 ONTAP-SAN 驱动程序在调整大小操作期间缩小 FlexVol 卷。
- 已修复 NFS v4.1 的 ANF 卷导入失败。

24.02 的变化

增强功能

- 添加了对 Cloud Identity 的支持。
  - 带有 ANF 的 AKS - Azure Workload Identity 将用作云标识。
  - 具有 FSxN 的 EKS - AWS IAM 角色将用作云身份。
- 添加了从 EKS 控制台将 Trident 作为附加组件安装在 EKS 集群上的支持。
- 添加了配置和禁用 iSCSI 自我修复的功能("问题 #864")。
- 向 ONTAP 驱动程序添加了 Amazon FSx 个性，以启用与 AWS IAM 和 SecretsManager 的集成，并启用 Trident 删除带有备份的 FSx 卷（"问题 #453"）。

Kubernetes

- 增加了对 Kubernetes 1.29 的支持。

修复

- 修复了未启用 ACP 时的 ACP 警告消息（"问题 #866"）。
- 当克隆与快照关联时，在为 ONTAP 驱动程序删除快照期间执行克隆拆分之前添加了 10 秒的延迟。

弃用

- 从多平台图像清单中删除了 in-toto 证明框架。

23.10 中的变更

修复

- 如果新请求的大小小于 ontap-nas 和 ontap-nas-flexgroup 存储驱动程序的总卷大小，则修复卷扩展（"问题 #834"）。
- 修复了卷大小显示，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（"问题 #722"）。
- 修复了 ONTAP-NAS-Economy 的 FlexVol 名称转换。
- 修复了节点重新启动时 Windows 节点上的 Trident 初始化问题。

增强功能

Kubernetes

增加了对 Kubernetes 1.28 的支持。

Trident

- 添加了对将 Azure Managed Identities (AMI) 与 azure-netapp-files 存储驱动程序一起使用的支持。
- 为 ONTAP-SAN 驱动程序添加了对 NVMe over TCP 的支持。
- 添加了在后端由 user ("问题 #558") 设置为挂起状态时暂停配置卷的功能。

23.07.1 中的变化

Kubernetes: 修复了 daemonset 删除问题，以支持零停机升级（"问题 #740"）。

23.07 的变化

修复

Kubernetes

- 修复了 Trident 升级，以忽略停留在终止状态（"问题 #740"）的旧 pod。
- 向"transient-trident-version-pod"定义("问题 #795")添加了容忍。

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 Node Staging 操作期间识别并修复虚拟 iSCSI 设备。
- 修复了存储驱动程序代码中的错误处理 ("问题 #816")。
- 修复了在 use-rest=true 的情况下使用 ONTAP 驱动程序时调整配额的问题。
- 修复了在 ontap-san-economy 中创建 LUN 克隆的问题。
- 将发布信息字段从 `rawDevicePath` 恢复到 `devicePath`；添加逻辑以填充和恢复（在某些情况下）`devicePath` 字段。

增强功能

Kubernetes

- 添加了对导入预配置快照的支持。
- 最小化部署和 daemonset linux 权限（"问题 #817"）。

Trident

- 不再报告"在线"卷和快照的状态字段。
- 如果 ONTAP 后端处于离线状态，则更新后端状态（"问题 #801"，"#543"）。
- 在 ControllerVolumePublish 工作流程中始终检索和发布 LUN 序列号。
- 添加了额外的逻辑来验证 iSCSI 多路径设备序列号和大小。
- 对 iSCSI 卷的额外验证，以确保正确的多路径设备未暂存。

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

对组织和格式进行了许多改进。

弃用

Kubernetes

- 已删除对 v1beta1 快照的支持。
- 已删除对 pre-CSI 卷和存储类的支持。
- 将支持的最小 Kubernetes 更新为 1.22。

23.04 的变化

> 只有启用了非优雅节点关闭功能门的 Kubernetes 版本才支持 ONTAP-SAN-* 卷的强制卷分离。安装时必须使用 `--enable-force-detach` Trident 安装程序标志启用强制分离。

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 集群角色权限与捆绑包权限（"问题 #799"）同步的问题。
- 修复了在 RWX 模式下在多个节点上附加原始块卷的问题。
- 修复了 SMB 卷的 FlexGroup 克隆支持和卷导入。
- 修复了 Trident 控制器无法立即关闭的问题（"问题 #811"）。
- 添加了修复，以列出与使用 ontap-san-* 驱动程序配置的指定 LUN 关联的所有 igroup 名称。
- 添加了一个修复，以允许外部进程运行到完成。
- 修复了 s390 架构("问题 #537")的编译错误。
- 修复了卷挂载操作期间不正确的日志记录级别（"问题 #781"）。
- 已修复潜在类型断言错误 ("问题 #802")。

增强功能

- Kubernetes：
  - 增加了对 Kubernetes 1.27 的支持。
  - 添加了对导入 LUKS 卷的支持。
  - 添加了对 ReadWriteOncePod PVC 访问模式的支持。
  - 在非优雅节点关闭场景中，增加了对 ONTAP-SAN-* 卷的强制分离支持。
  - 所有 ONTAP-SAN-* 卷现在将使用每个节点 igroup。LUN 将仅映射到 igroup，同时主动发布到这些节点以改善我们的安全态势。当 Trident 确定在不影响活动工作负载的情况下这样做是安全的时，现有卷将被机会性地切换到新的 igroup 方案（"问题 #758"）。
  - 通过从 ONTAP-SAN-* 后端清理未使用的 Trident 管理的 igroup，提高了 Trident 安全性。
- 在 ontap-nas-economy 和 ontap-nas-flexgroup 存储驱动程序中添加了对使用 Amazon FSx 的 SMB 卷的支持。
- 通过 ontap-nas、ontap-nas-economy 和 ontap-nas-flexgroup 存储驱动程序添加了对 SMB 共享的支持。
- 添加了对 arm64 节点("问题 #732")的支持。

- 通过首先停用 API 服务器来改进 Trident 关闭程序（"问题 #811"）。
- 向 Makefile 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（"问题 #758"）时，将不再创建后端范围的 igroup。

23.01.1 中的变化

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 集群角色权限与捆绑包权限同步的问题"问题 #799"。
- 添加了一个修复，以允许外部进程运行到完成。
- 修复了在 RWX 模式下在多个节点上附加原始块卷的问题。
- 修复了 SMB 卷的 FlexGroup 克隆支持和卷导入。

23.01 的变化

Trident 现在支持 Kubernetes 1.27。请在升级 Kubernetes 之前升级 Trident。

修复

- Kubernetes：添加了排除 Pod Security Policy 创建的选项，以通过 Helm ("问题 #783，#794") 修复 Trident 安装。

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（"问题 #757"）。
- 添加了检测和修复主机节点上损坏或过时的 iSCSI 会话的自动化功能。
- 添加了对扩展 LUKS 加密卷的支持。
- Kubernetes：添加了对 LUKS 加密卷的凭据轮换支持。

Trident

- 在 ontap-nas 存储驱动程序中添加了对使用 Amazon FSx for NetApp ONTAP 的 SMB 卷的支持。
- 使用 SMB 卷时添加了对 NTFS 权限的支持。
- 增加了对具有 CVS 服务级别的 GCP 卷的存储池的支持。
- 在使用 ontap-nas-flexgroup 存储驱动创建 FlexGroups 时，新增了对可选使用 flexgroupAggregateList 的支持。
- 改进了管理多个 FlexVol 卷时 ontap-nas-economy 存储驱动程序的性能

- 已为所有 ONTAP NAS 存储驱动程序启用 dataLIF 更新。
- 更新了 Trident 部署和 DaemonSet 命名约定，以反映主机节点操作系统。

弃用

- Kubernetes：将最低支持的 Kubernetes 更新为 1.21。
- 配置 `ontap-san`或 `ontap-san-economy`驱动程序时不应再指定 DataLIF。

22.10 中的变化

在升级到 **Trident 22.10** 之前，您必须阅读以下重要信息。

有关 Trident 22.10 的重要信息

- Trident 现在支持 Kubernetes 1.25。在升级到 Kubernetes 1.25 之前，必须将 Trident 升级到 22.10。
- Trident 现在严格强制在 SAN 环境中使用多路径配置，推荐值为 multipath.conf 文件中的 `find_multipaths: no`。

  使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes`或 `find_multipaths: smart`值将导致挂载失败。自 `21.07` 版本发布以来，`Trident` 建议使用 `find_multipaths: no`。

修复

- 修复了在 22.07.0 升级期间使用 `credentials`字段创建的 ONTAP 后端无法联机的特定问题（"问题 #759"）。
- **Docker**：修复了导致 Docker 卷插件在某些环境（"问题 #548" 和 "问题 #760"）中无法启动的问题。
- 修复了特定于 ONTAP SAN 后端的 SLM 问题，以确保仅发布属于报告节点的 dataLIF 子集。
- 修复了连接卷时发生不必要的 iSCSI LUN 扫描的性能问题。
- 删除了 Trident iSCSI 工作流程中的粒度重试，以快速失败并减少外部重试间隔。
- 修复了在对应的多路径设备已刷新时刷新 iSCSI 设备时返回错误的问题。

增强功能

- Kubernetes：
  - 添加了对 Kubernetes 1.25 的支持。在升级到 Kubernetes 1.25 之前，您必须将 Trident 升级到 22.10。
  - 为 Trident Deployment 和 DaemonSet 添加了单独的 ServiceAccount、ClusterRole 和 ClusterRoleBinding，以便将来增强权限。
  - 已添加对 "跨命名空间卷共享" 的支持。
- 所有 Trident `ontap-*` 存储驱动程序现在都与 ONTAP REST API 配合使用。
- 添加了新的运算符 yaml (`bundle_post_1_25.yaml`）without a `PodSecurityPolicy` 以支持 Kubernetes 1.25。
- 已为"支持 LUKS 加密卷"和 `ontap-san`以及 `ontap-san-economy`存储驱动程序添加了支持。

- 添加了对 Windows Server 2019 节点的支持。
- 通过 `azure-netapp-files`存储驱动程序添加了"在 Windows 节点上支持 SMB 卷"。
- ONTAP 驱动程序的自动 MetroCluster 切换检测现已正式推出。

弃用

- **Kubernetes**：将最低支持的 Kubernetes 更新为 1.20。
- 删除了 Astra Data Store (ADS) 驱动程序。
- 删除了为 iSCSI 配置工作节点多路径时对 `yes`和 `smart`选项的支持 `find_multipaths`。

22.07 的变化

修复

Kubernetes

- 修复了使用 Helm 或 Trident Operator 配置 Trident 时处理节点选择器的布尔值和数字值的问题。("GitHub 问题 #700")
- 修复了处理来自非 CHAP 路径的错误的问题，以便 kubelet 在失败时重试。"GitHub 问题 #736")

增强功能

- 从 k8s.gcr.io 过渡到 registry.k8s.io 作为 CSI 映像的默认注册表
- ONTAP-SAN 卷现在将使用每个节点的 igroup，并且仅将 LUN 映射到 igroup，同时主动发布到这些节点，以改善我们的安全状况。当 Trident 确定在不影响活动工作负载的情况下这样做是安全的时，现有卷将被机会性地切换到新的 igroup 方案。
- 在 Trident 安装中包含了 ResourceQuota，以确保当默认情况下 PriorityClass 消耗受限时，Trident DaemonSet 能被调度。
- 向 Azure NetApp Files 驱动程序添加了对网络功能的支持。("GitHub 问题 #717")
- 向 ONTAP 驱动程序添加了技术预览自动 MetroCluster 切换检测。("GitHub 问题 #228")

弃用

- **Kubernetes**：将最低支持的 Kubernetes 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。

移除

- 已删除 AWS CVS 驱动程序（自 22.04 起已弃用）。
- Kubernetes
  - 已从节点 Pod 中删除不必要的 SYS_ADMIN 功能。
  - 将 nodeprep 减少到简单的主机信息和主动服务发现，以尽最大努力确认 NFS/iSCSI 服务在工作节点上可用。

文档

已添加一个新的"Pod 安全标准" (PSS) 部分，详细说明 Trident 在安装时启用的权限。

22.04 的变化

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

> 如果要从任何先前的 Trident 版本升级并使用 Azure NetApp Files，则 `location` 配置参数现在是必需的单例字段。

修复

- 改进了 iSCSI 启动程序名称的解析。("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")
- 已修复 Trident CRD 中的重复密钥声明。("GitHub 问题 #671")
- 修复了不准确的 CSI Snapshot 日志。("GitHub 问题 #629")
- 已修复在已删除节点上取消发布卷的问题。("GitHub 问题 #691")
- 添加了对块设备上文件系统不一致的处理。("GitHub 问题 #656")
- 修复了在安装过程中设置 `imageRegistry` 标志时拉取自动支持镜像的问题。("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

- 到 Trident 安全端点的入站连接现在至少需要 TLS 1.3。("GitHub 问题 #698")
- Trident 现在将 HSTS 标头添加到其安全端点的响应中。
- Trident 现在尝试自动启用 Azure NetApp Files unix 权限功能。
- **Kubernetes**：Trident daemonset 现在以 system-node-critical 优先级类运行。("GitHub 问题 #694")

移除

已删除 E-Series 驱动程序（自 20.07 起禁用）。

22.01.1 中的变化

修复

- 已修复在已删除节点上取消发布卷的问题。("GitHub 问题 #691")
- 修复了访问 ONTAP API 响应中聚合空间的 nil 字段时的死机问题。

22.01.0 中的变化

修复

- *Kubernetes：*增加大型集群的节点注册回退重试时间。
- 已修复 azure-netapp-files 驱动程序可能被多个同名资源混淆的问题。
- 如果使用括号指定，ONTAP SAN IPv6 DataLIF 现在可以工作。
- 修复了尝试导入已导入卷返回 EOF 使 PVC 处于待定状态的问题。("GitHub 问题 #489")
- 修复了在 SolidFire 卷上创建 > 32 个快照时 Trident 性能减慢的问题。
- 在 SSL 证书创建中用 SHA-256 替换了 SHA-1。
- 修复了 Azure NetApp Files 驱动程序，以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，以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：
  - 增加了对 Kubernetes 1.23 的支持。
  - 通过 Trident Operator 或 Helm 安装时，为 Trident pod 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")
- 新增对 Azure NetApp Files 卷的 'unixPermissions' 选项的支持。("GitHub 问题 #666")

弃用

Trident REST 接口只能在 127.0.0.1 或 [::1] 地址进行侦听和服务

21.10.1 中的变更

> v21.10.0 版本存在一个问题，当节点被删除然后添加回 Kubernetes 集群时，可以将 Trident 控制器置于 CrashLoopBackOff 状态。此问题已在 v21.10.1（GitHub 问题 669）中修复。

修复

- 修复了在 GCP CVS 后端导入卷时导致导入失败的潜在竞争条件。
- 修复了当节点被删除然后添加回 Kubernetes 集群时，Trident 控制器可能进入 CrashLoopBackOff 状态的问题（GitHub 问题 669）。
- 修复了在未指定 SVM 名称的情况下不再发现 SVM 的问题（GitHub 问题 612）。

21.10.0 中的变更

修复

- 修复了无法在与源卷相同的节点上挂载 XFS 卷克隆的问题（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）。
- Kubernetes 相关修补程序：

- 使用 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序创建快照时，返回卷的已用空间作为最小 restoreSize（GitHub 问题 645）。
- 修复了在调整卷大小后记录 `Failed to expand filesystem`错误的问题（GitHub 问题 560）。
- 修复了 Pod 可能陷入 `Terminating`状态的问题（GitHub 问题 572）。
- 修复了 `ontap-san-economy` FlexVol 可能充满快照 LUN 的情况（GitHub 问题 533）。
- 修复了不同映像的自定义 YAML 安装程序问题（GitHub 问题 613）。
- 修复了快照大小计算（GitHub 问题 611）。
- 修复了所有 Trident 安装程序可以将普通 Kubernetes 识别为 OpenShift 的问题（GitHub 问题 639）。
- 修复了 Trident 运算符在 Kubernetes API 服务器无法访问时停止协调的问题（GitHub 问题 599）。

增强功能

- 增加了对 GCP-CVS Performance 卷 `unixPermissions`选项的支持。
- 增加了对 GCP 中规模优化的 CVS 卷的支持，范围为 600 GiB 至 1 TiB。
- Kubernetes 相关增强功能：
  - 增加了对 Kubernetes 1.22 的支持。
  - 使 Trident 运算符和 Helm chart 能够与 Kubernetes 1.22 （GitHub 问题 628）配合使用。
  - 将操作员映像添加到 `tridentctl`映像命令（GitHub 问题 570）。

实验性增强功能

- 在 `ontap-san` 驱动程序中添加了对卷复制的支持。
- 为 `ontap-nas-flexgroup`、 `ontap-san` 和 `ontap-nas-economy` 驱动程序添加了 技术预览 REST 支持。

已知问题

已知问题可识别可能妨碍您成功使用产品的问题。

- 在将安装了 Trident 的 Kubernetes 集群从 1.24 升级到 1.25 或更高版本时，必须更新 values.yaml 以将 `excludePodSecurityPolicy`设置为 `true`或将 `--set excludePodSecurityPolicy=true`添加到 `helm upgrade` 命令中，然后才能升级集群。
- Trident 现在会对没有在其 StorageClass 中指定 `fsType` 的卷强制使用空白 `fsType` (`fsType=""`。在使用 Kubernetes 1.17 或更高版本时，Trident 支持为 NFS 卷提供空白 `fsType`。对于 iSCSI 卷，当使用 Security Context 强制 `fsGroup` 时，您需要在 StorageClass 上设置 `fsType`。
- 当跨多个 Trident 实例使用后端时，每个后端配置文件应具有不同的 `storagePrefix`值用于 ONTAP 后端或使用不同的 `TenantName`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 后端上创建现有卷会成功，因为 Trident 将卷创建视为幂等操作。如果 `storagePrefix` 或 `TenantName`没有差异，则可能存在在同一后端上创建的卷的名称冲突。
- 安装 Trident（使用 `tridentctl`或 Trident Operator）并使用 `tridentctl`来管理 Trident 时，应确保设置 `KUBECONFIG`环境变量。这对于指示 `tridentctl`应该针对的 Kubernetes 集群是必要的。使用多个 Kubernetes 环境时，应确保 `KUBECONFIG`文件来源准确。
- 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装载选项传递到卷。对于

RHEL/Red Hat Enterprise Linux CoreOS (RHCOS) 实例，需要 `discard` "挂载选项"；请确保在 [`StorageClass`] 中包含 discard mountOption 以支持在线块丢弃。

- 如果每个 Kubernetes 集群有多个 Trident 实例，Trident 无法与其他实例通信，也无法发现它们创建的其他卷，如果在一个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 集群只能有一个 Trident 实例。
- 如果在 Trident 离线时从 Kubernetes 中删除了基于 Trident 的 `StorageClass`对象，Trident 在重新上线时不会从其数据库中删除相应的存储类。您应该使用 `tridentctl`或 REST API 删除这些存储类。
- 如果用户在删除对应的 PVC 之前删除了 Trident 配置的 PV，Trident 不会自动删除后备卷。您应该通过 `tridentctl`或 REST API 删除卷。
- 除非聚合集对于每个配置请求是唯一的，否则 ONTAP 不能一次同时配置多个 FlexGroup。
- 在 IPv6 上使用 Trident 时，您应该在后端定义中的方括号内指定 `managementLIF`和 `dataLIF`。例如，`[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。

> 您无法在 ONTAP SAN 后端上指定 `dataLIF`。Trident 发现所有可用的 iSCSI LIF 并使用它们建立多路径会话。

- 如果将 `solidfire-san` 驱动程序与 OpenShift 4.5 一起使用，请确保底层工作节点使用 MD5 作为 CHAP 身份验证算法。Element 12.7 提供了安全的符合 FIPS 的 CHAP 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

如果您未运行 Trident 25.10，则可以根据 "Trident 支持生命周期" 获得先前版本的文档。

- "Trident 25.06"
- "Trident 25.02"
- "Trident 24.10"
- "Trident 24.06"
- "Trident 24.02"
- "Trident 23.10"
- "Trident 23.07"
- "Trident 23.04"
- "Trident 23.01"

NetApp Trident 支持 ONTAP ASA r2 存储系统

NetApp Trident 25.02 及更高版本支持 NetApp ASA r2 系统作为存储后端。有关详细信息，请参见 "ASA r2 系统"。

ASA r2 系统需要 `ontap-san`驱动程序。Trident 不支持 ASA r2 系统的 `ontap-san-economy`驱动程序。

当您在后端配置中指定 `ontap-san` 作为 `storageDriverName` 时，Trident 会自动检测 ASA r2 存储系统。

Trident 为具有 Trident protect 的 ASA r2 系统提供有限的数据保护。

支持的 SAN 协议取决于您的 Trident 版本：

- Trident 25.02 及更高版本支持 iSCSI。
- Trident 25.06 及更高版本除 iSCSI 外还支持 NVMe/TCP。

必须至少将一个聚合分配给 ONTAP 后端存储的 Storage Virtual Machine (SVM)。有关说明，请参阅 "为 ASA r2 系统中的 SVM 分配聚合"。

支持的操作

- 配置持久卷 (PV)
- 动态卷配置
- 创建和删除卷
- 克隆卷
- 扩展卷
- 管理存储类

不支持的操作

- LUKS 加密
- SnapMirror 卷复制
- 限制聚合使用
- 空间预留模式
- Snapshot
- 分层

有关详细信息，请参见 "ONTAP SAN 配置选项和示例" 。

已知问题

已知问题可识别可能妨碍您成功使用此产品版本的问题。

以下已知问题影响当前版本：

恢复大型文件的 **Restic** 备份可能会失败

从使用 Restic 创建的 Amazon S3 备份还原 30GB 或更大的文件时，还原操作可能会失败。作为一种解决方法，使用 Kopia 作为数据移动器来备份数据（Kopia 是备份的默认数据移动器）。有关说明，请参阅 "使用 Trident Protect 保护应用程序"。

开始使用

了解 Trident

了解 Trident

Trident 是一个完全支持的开源项目，由 NetApp 维护。它旨在使用行业标准接口（例如容器存储接口（CSI））帮助您满足容器化应用程序的持久性需求。

什么是 Trident?

Netapp Trident 支持在公共云或本地所有流行 NetApp 存储平台上使用和管理存储资源，包括本地 ONTAP 集群（AFF、FAS 和 ASA）、ONTAP Select、Cloud Volumes ONTAP、Element 软件（NetApp HCI、SolidFire ）、Azure NetApp Files 和 Amazon FSx for NetApp ONTAP。

Trident 是一个符合容器存储接口 (CSI) 的动态存储编排器，可与 "Kubernetes" 进行本地集成。Trident 在集群中的每个工作节点上作为单个控制器 Pod 和一个节点 Pod 运行。有关详细信息，请参阅 "Trident 架构"。

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 主机的存储资源调配和管理。有关详细信息，请参阅 "为 Docker 部署 Trident"。

如果这是您第一次使用 Kubernetes，则应熟悉"Kubernetes 概念和工具"。

Kubernetes 与 NetApp 产品集成

NetApp 存储产品组合集成了 Kubernetes 集群的许多方面，提供了高级数据管理功能，增强了 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" 是企业级 Azure 文件共享服务，由 NetApp 提供支持。您可以在 Azure 中以本机方式运行要求最苛刻的基于文件的工作负载，并提供您期望从 NetApp 获得的性能和丰富的数据管理。

Cloud Volumes ONTAP

"Cloud Volumes ONTAP" 是在云中运行 ONTAP 数据管理软件的纯软件存储设备。

Google Cloud NetApp Volumes

"Google Cloud NetApp Volumes" 是 Google Cloud 中的完全托管文件存储服务，可提供高性能、企业级文件存储。

Element 软件

"Element" 使存储管理员能够通过保证性能并实现简化和精简的存储占用空间来整合工作负载。

NetApp HCI

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

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" 是 NetApp 多协议的统一存储操作系统，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、混合或全硬盘配置，并提供许多不同的部署模式：本地 FAS、AFA 和 ASA 集群、ONTAP Select 和 Cloud Volumes ONTAP。Trident 支持这些 ONTAP 部署模式。

Trident 架构

Trident 作为单个 Controller Pod 以及集群中每个工作节点上的 Node Pod 运行。节点 Pod 必须运行在任何可能要装载 Trident 卷的主机上。

了解控制器 **pod** 和节点 **pod**

Trident 在 Kubernetes 集群上部署为单个 Trident 控制器 Pod 和一个或多个 Trident 节点 Pod，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 Storage 社区维护。

Kubernetes "节点选择器" 和 "容忍和污点" 用于限制 pod 在特定或首选节点上运行。您可以在 Trident 安装期间为控制器和节点 pod 配置节点选择器和容差。

- 控制器插件处理卷配置和管理，例如快照和调整大小。
- 节点插件处理将存储附加到节点。

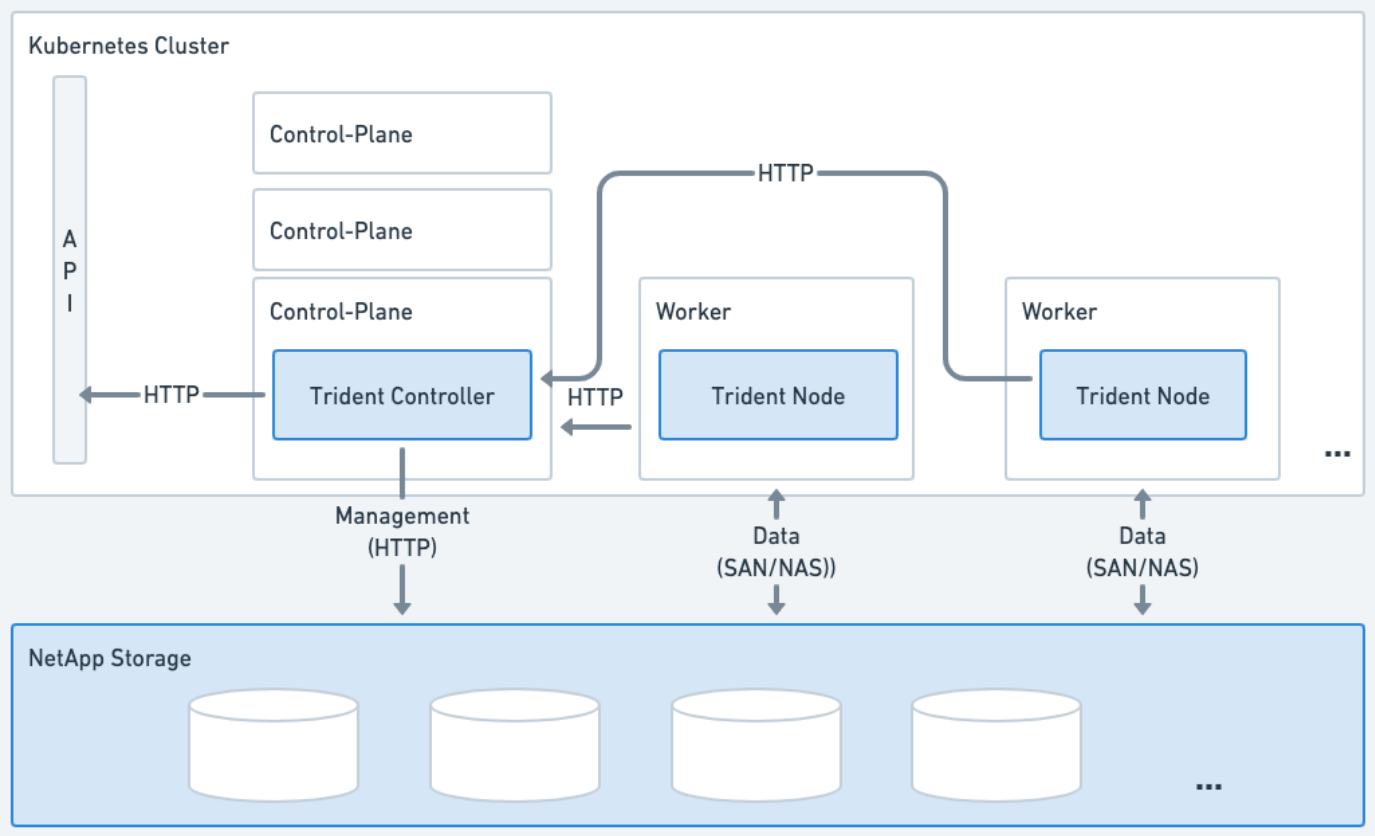


图 **1.** 在 **Kubernetes** 集群上部署的 **Trident**

Trident 控制器 **Pod**

Trident Controller Pod 是运行 CSI Controller 插件的单个 Pod。

- 负责配置和管理 NetApp 存储中的卷
- 通过 Kubernetes Deployment 管理
- 可以在控制平面或工作节点上运行，具体取决于安装参数。

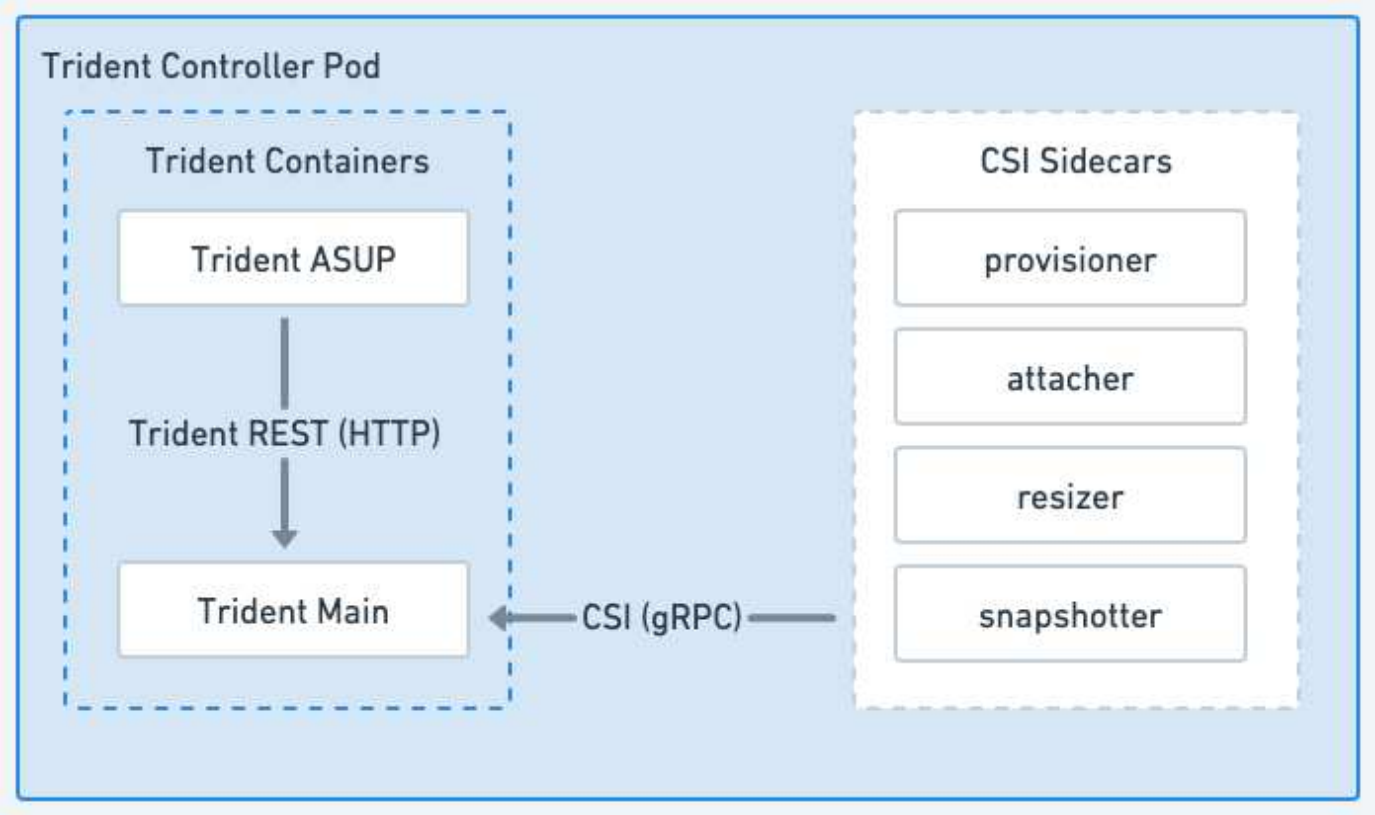


图 **2. Trident** 控制器 **Pod** 图

Trident 节点 Pod

Trident 节点 Pod 是运行 CSI 节点插件的特权 Pod。

- 负责挂载和卸载主机上运行的 Pod 的存储
- 通过 Kubernetes DaemonSet 管理
- 必须在将装载 NetApp 存储的任何节点上运行

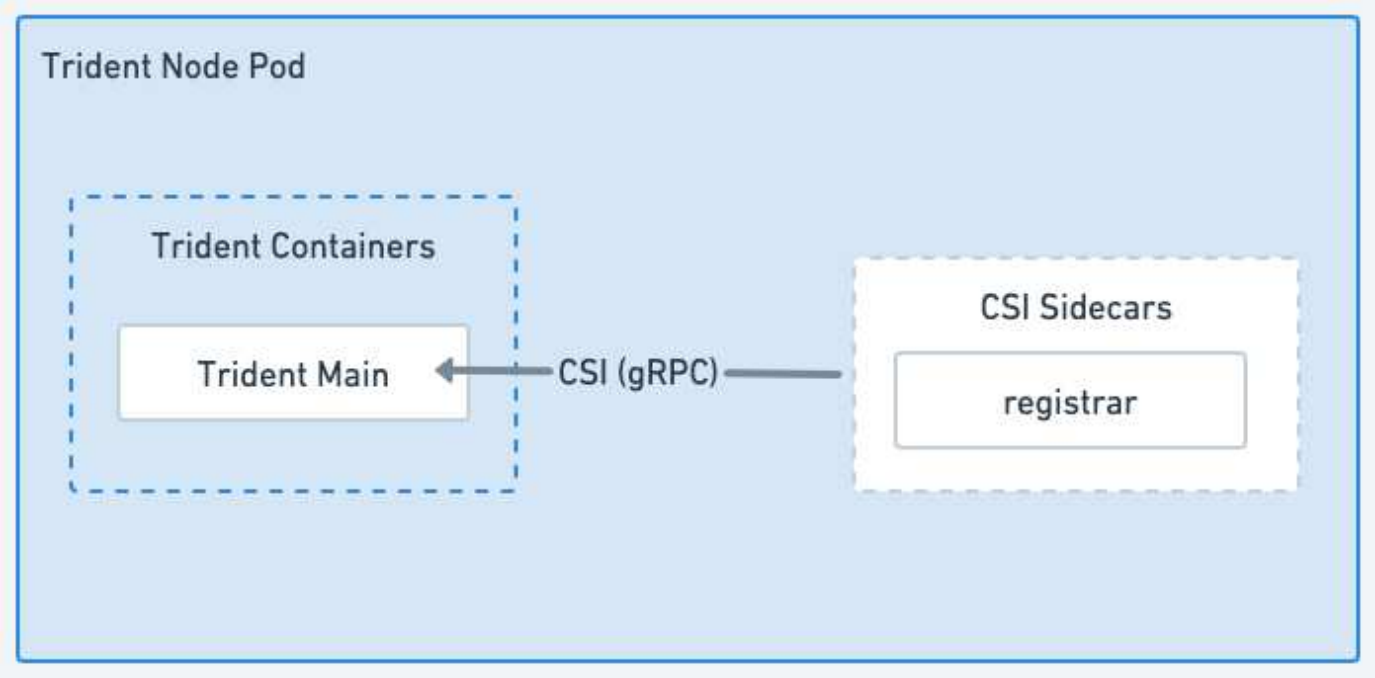


图 **3. Trident** 节点 **Pod** 图

支持的 **Kubernetes** 集群架构

以下 Kubernetes 架构支持 Trident：

Kubernetes 集群架构	支持	默认安装
单主机，计算	是	是
多个主节点、计算节点	是	是
Master、 `etcd`、compute	是	是
主控、基础架构、计算	是	是

概念

配置

Trident 中的置备有两个主要阶段。第一阶段将存储类与一组合适的后端存储池关联，并在置备之前作为必要的准备工作进行。第二阶段包括卷创建本身，并要求从与待处理卷的存储类关联的存储池中选择存储池。

存储类关联

将后端存储池与存储类关联依赖于存储类的请求属性及其 `storagePools`、 `additionalStoragePools`和 `excludeStoragePools`列表。在创建存储类时，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合适存储池集合中。此外，Trident 会将 `additionalStoragePools`列表中列出的所有存储池添加到该集合中，

即使它们的属性不满足存储类的所有或任何请求属性。你应该使用 `excludeStoragePools`列表来重写和删除用于存储类的存储池。每次添加新后端时，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，并删除已标记为排除的任何存储池。

创建卷

然后，Trident 使用存储类和存储池之间的关联来确定在何处配置卷。在创建卷时，Trident 首先获取该卷的存储类的存储池集，并且如果为该卷指定协议，则 Trident 将删除那些无法提供所请求的协议的存储池（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 随机化此结果集的顺序，以促进卷的均匀分布，然后进行迭代，尝试依次在每个存储池上配置卷。如果在其中一个上成功，则成功返回，记录过程中遇到的任何故障。Trident 返回故障*仅当*它在所请求的存储类和协议的*所有*可用存储池上都无法配置时。

卷快照

详细了解 Trident 如何为其驱动程序创建卷快照。

了解卷快照创建

- 对于 `ontap-nas`、 `ontap-san`和 `azure-netapp-files`驱动程序，每个永久卷 (PV) 映射到一个 FlexVol 卷。因此，卷快照将创建为 NetApp 快照。NetApp 快照技术比竞争对手的快照技术提供更多的稳定性、可扩展性、可恢复性和性能。这些快照副本在创建所需的时间和存储空间方面都非常高效。
- 对于 `ontap-nas-flexgroup`驱动程序，每个持久卷 (PV) 映射到 FlexGroup。因此，卷快照被创建为 NetApp FlexGroup 快照。NetApp 快照技术与竞争对手的快照技术相比，提供了更高的稳定性、可扩展性、可恢复性和性能。这些快照副本在创建所需的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，PV 的 VolumeSnapshots 是通过执行关联 LUN 的 FlexClones 来实现的。ONTAP FlexClone 技术几乎可以即时创建最大数据集的副本。副本与其父级共享数据块，除了元数据所需的存储空间外，不占用任何存储空间。
- 对于 `solidfire-san` 驱动程序，每个 PV 映射到在 NetApp Element 软件/NetApp HCI 群集上创建的 LUN。VolumeSnapshots 由底层 LUN 的 Element 快照表示。这些快照是时间点副本，仅占用少量系统资源和空间。
- 使用 `ontap-nas` 和 `ontap-san` 驱动程序时，ONTAP 快照是 FlexVol 的时间点副本，占用 FlexVol 本身的空间。这可能会导致卷中的可写空间量随着快照的创建/计划而随时间减少。解决这个问题的一个简单方法是通过 Kubernetes 调整大小来增加卷。另一个选项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。也可以删除不是通过 Kubernetes 创建的 ONTAP 快照。

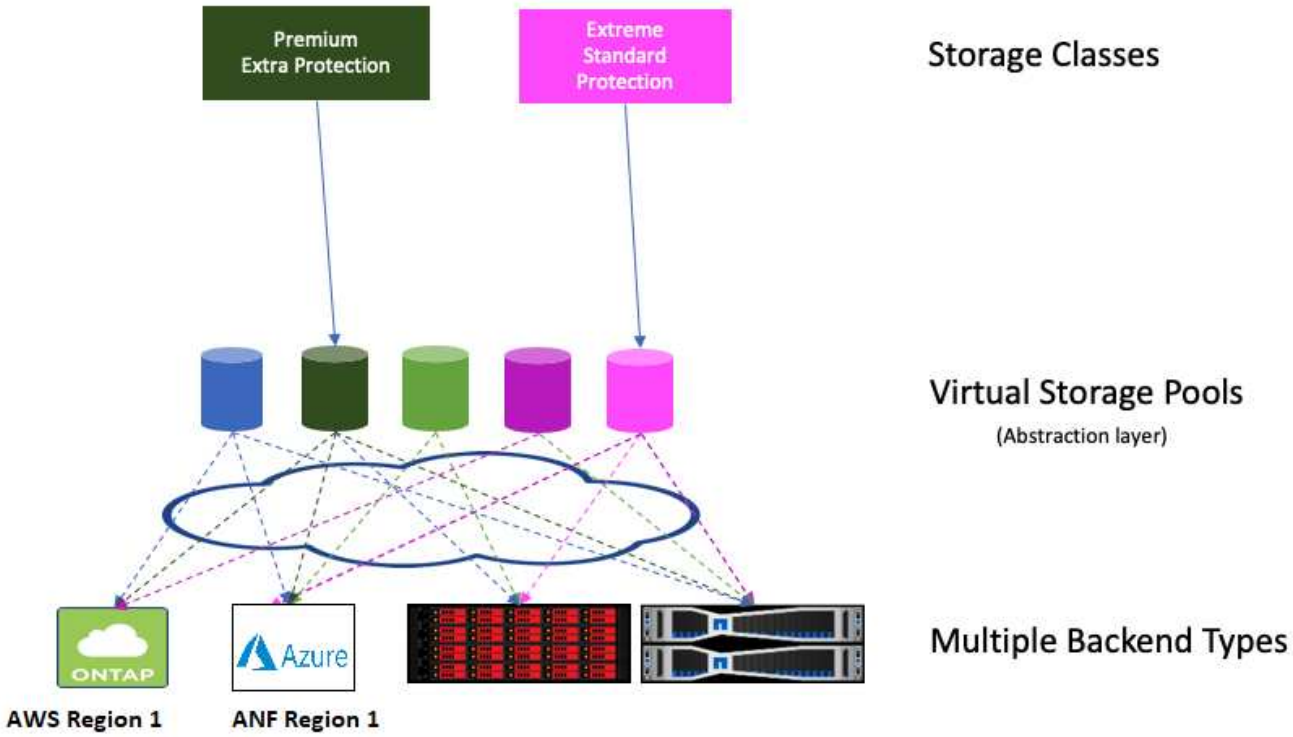
使用 Trident，您可以使用 VolumeSnapshots 从中创建新的 PV。从这些快照创建 PV 是通过对支持的 ONTAP 后端使用 FlexClone 技术来执行的。从快照创建 PV 时，后备卷是快照父卷的 FlexClone。 `solidfire-san` 驱动程序使用 Element 软件卷克隆从快照创建 PV。在这里，它从 Element 快照创建一个克隆。

虚拟池

虚拟池在 Trident 存储后端和 Kubernetes `StorageClasses` 之间提供了一个抽象层。它们允许管理员以通用的、与后端无关的方式为每个后端定义方面，例如位置、性能和保护，而无需 `StorageClass`指定使用哪种物理后端、后端池或后端类型来满足所需标准。

了解虚拟池

存储管理员可以在 JSON 或 YAML 定义文件中的任何 Trident 后端上定义虚拟池。



在虚拟池列表之外指定的任何方面对后端都是全局的，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）。

- 在定义虚拟池时，请勿尝试重新排列后端定义中现有虚拟池的顺序。
- 我们建议不要修改现有虚拟池的属性。您应该定义新的虚拟池来进行更改。

大多数方面都以特定于后端的术语指定。至关重要的是，方面值不会暴露在后端的驱动程序之外，并且不可用于匹配 `StorageClasses`。相反，管理员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之间是通用的。与方面一样，可以为每个池指定标签，也可以为后端指定全局标签。与具有预定义名称和值的方面不同，管理员有完全的自由裁量权来根据需要定义标签键和值。为方便起见，存储管理员可以为每个虚拟池定义标签，并按标签对卷进行分组。

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 `-`

通过引用选择器参数中的标签， `StorageClass`标识要使用的虚拟池。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=高级	匹配
!=	性能!=极致	不匹配
in	位置在（east、west）	位于值集合中
notin	性能缺陷（银牌、铜牌）	不在值集合中
<key>	保护	以任何值存在
!<key>	!保护	不存在

卷访问组

详细了解 Trident 的使用方式 "卷访问组"。

> 如果您使用的是 CHAP，请忽略此部分，建议使用 CHAP 来简化管理并避免下面描述的扩展限制。此外，如果您在 CSI 模式下使用 Trident，则可以忽略此部分。当 Trident 作为增强的 CSI 配置程序安装时，它会使用 CHAP。

了解卷访问组

Trident 可以使用卷访问组来控制对其提供的卷的访问。如果禁用 CHAP，它将期望找到名为 `trident`的访问组，除非您在配置中指定一个或多个访问组 ID。

虽然 Trident 将新卷与配置的访问组相关联，但它不会自行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，并且它们需要包含来自 Kubernetes 集群中每个节点的 iSCSI IQN，这些节点可能会装载该后端提供的卷。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，应使用多个访问组。每个访问组最多可以包含 64 个 IQN，每个卷可以属于四个访问组。配置最多四个访问组后，集群中大小最多为 256 个节点的任何节点都可以访问任何卷。有关卷访问组的最新限制，请参阅 "此处"。

如果要将配置从使用默认 `trident` 访问组的配置修改为也使用其他访问组的配置，请在列表中包含 `trident` 访问组的 ID。

Trident 快速入门

您可以通过几个步骤安装 Trident 并开始管理存储资源。在开始之前，请查看"Trident 要求"。

> 对于 Docker，请参阅 "适用于 Docker 的 Trident"。

1. 准备 **worker** 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"

2 安装 **Trident**

Trident 提供多种针对各种环境和组织进行优化的安装方法和模式。

"安装 Trident"

3 创建后端

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

"配置后端" 适用于您的存储系统

4 创建 **Kubernetes StorageClass**

Kubernetes StorageClass 对象将 Trident 指定为配置程序，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 配置程序的 Kubernetes 对象创建匹配的存储类。

"创建存储类"

5 预配卷

PersistentVolume (PV) 是由集群管理员在 Kubernetes 集群上配置的物理存储资源。*PersistentVolumeClaim* (PVC) 是访问集群上 PersistentVolume 的请求。

创建一个 PersistentVolume (PV) 和一个 PersistentVolumeClaim (PVC)，使用配置的 Kubernetes StorageClass 来请求对 PV 的访问。然后，您可以将 PV 挂载到 pod。

"预配卷"

下一步是什么?

现在，您可以添加其他后端、管理存储类、管理后端和执行卷操作。

要求

在安装 Trident 之前，您应该查看这些一般系统要求。特定后端可能有其他要求。

有关 **Trident** 的重要信息

您必须阅读以下有关 **Trident** 的重要信息。

有关 Trident 的重要信息

- Trident 现在支持 Kubernetes 1.34。在升级 Kubernetes 之前升级 Trident。
- Trident 严格执行在 SAN 环境中使用多路径配置，在 multipath.conf 文件中的建议值为 `find_multipaths: no`。

  使用非多路径配置或在 multipath.conf 文件中使用 `find_multipaths: yes`或 `find_multipaths: smart`值将导致挂载失败。自 `21.07` 版本发布以来，`Trident` 建议使用 `find_multipaths: no`。

支持的前端（编排器）

Trident 支持多个容器引擎和编排器，其中包括：

- Anthos On-Prem (VMware) 和 Anthos on bare metal 1.16
- Kubernetes 1.27 - 1.34
- OpenShift 4.12、4.14 - 4.20 （如果您计划使用 OpenShift 4.19 的 iSCSI 节点准备，则支持的最低 Trident 版本为 25.06.1。）

> Trident 继续支持与 "Red Hat Extended Update Support (EUS) 版本生命周期" 保持一致的旧 OpenShift 版本，即使它们依赖于不再正式支持上游的 Kubernetes 版本。在这种情况下安装 Trident 时，您可以安全地忽略有关 Kubernetes 版本的任何警告消息。

- Rancher Kubernetes Engine 2 (RKE2) v1.28.x - 1.34.x

> 虽然 *Trident* 支持 *Rancher Kubernetes Engine 2 (RKE2)* 版本 *1.27.x - 1.34.x*，但 *Trident* 目前仅在 *RKE2 v1.28.5+rke2r1* 上通过了认证。

Trident 还与许多其他完全托管和自我管理的 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 可以用作 "KubeVirt" 的存储提供商。

> 在将安装了 Trident 的 Kubernetes 集群从 1.25 升级到 1.26 或更高版本之前，请参阅 "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 支持 KubeVirt 和 OpenShift Virtualization

支持的存储驱动程序：

Trident 支持以下用于 KubeVirt 和 OpenShift Virtualization 的 ONTAP 驱动程序：

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

需要考虑的要点：

- 在 OpenShift Virtualization 环境中，更新存储类以包含 `fsType` 参数（例如： `fsType: "ext4"`）。如有需要，在 `dataVolumeTemplates`中使用 `volumeMode=Block`参数显式设置卷模式为 block，以通知 CDI 创建 Block 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 ）驱动程序仅受"volumeMode: Block"（原始设备）支持。对于这些驱动程序，无法使用 `fstype` 参数，因为卷是在原始设备模式下提供的。
- 对于需要 RWX 访问模式的实时迁移工作流，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)

功能要求

下表总结了此版本 Trident 及其支持的 Kubernetes 版本的可用功能。

功能	**Kubernetes** 版本	是否需要功能门?
Trident	1.27 - 1.34	否
卷 Snapshot	1.27 - 1.34	否
来自卷快照的 PVC	1.27 - 1.34	否
iSCSI PV 调整大小	1.27 - 1.34	否

功能	**Kubernetes** 版本	是否需要功能门?
ONTAP 双向 CHAP	1.27 - 1.34	否
动态导出策略	1.27 - 1.34	否
Trident Operator	1.27 - 1.34	否
CSI 拓扑	1.27 - 1.34	否

已测试的主机操作系统

虽然 Trident 不正式支持特定的操作系统，但已知以下操作系统可以正常工作：

- AMD64 和 ARM64 上的 OpenShift Container Platform 支持的 Red Hat Enterprise Linux CoreOS (RHCOS) 版本
- AMD64 和 ARM64 上的 Red Hat Enterprise Linux (RHEL) 8 或更高版本

  NVMe/TCP 需要 RHEL 9 或更高版本。

- AMD64 和 ARM64 上的 Ubuntu 22.04 LTS 或更高版本
- 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，因此将在任何 Linux worker 上运行。但是，这些 worker 需要能够使用标准 NFS 客户端或 iSCSI 启动程序挂载 Trident 提供的卷，具体取决于您使用的后端。

该 `tridentctl` 实用程序还可以在任何这些 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装载您为 Pod 配置的卷。若要准备工作节点，必须根据您的驱动程序选择安装 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，然后后端配置才能使用它。

"配置后端"

Trident 端口

Trident 需要访问特定端口进行通信。

"Trident 端口"

容器映像和相应的 **Kubernetes** 版本

对于气隙安装，以下列表是安装 Trident 所需的容器映像的参考。使用 `tridentctl images` 命令验证所需容器映像的列表。

Trident 25.10 所需的容器映像

Kubernetes 版本	容器映像
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:25.10.0 • docker.io/netapp/trident-autosupport:25.10 • registry.k8s.io/sig-storage/csi-provisioner:v5.3.0 • registry.k8s.io/sig-storage/csi-attacher:v4.10.0 • registry.k8s.io/sig-storage/csi-resizer:v1.14.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.3.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.15.0 • docker.io/netapp/trident-operator:25.10.0（可选）

安装 Trident

使用 Trident 操作员安装

使用 tridentctl 安装

使用 OpenShift 认证操作员安装

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装载您为 Pod 配置的卷。要准备工作节点，必须根据您的驱动程序选择安装 NFS、iSCSI、NVMe/TCP 或 FC 工具。

选择合适的工具

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

NFS 工具

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

iSCSI 工具

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

NVMe 工具

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

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

基于 **FC** 的 **SCSI** 工具

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

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

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

SMB 工具

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

节点服务发现

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

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

审阅事件

Trident 创建事件以供节点标识发现的服务。要查看这些事件，请运行：

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

查看已发现的服务

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

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

NFS 卷

使用操作系统的命令安装 NFS 工具。确保 NFS 服务在启动时启动。

RHEL 8+

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

Ubuntu

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

安装 NFS 工具后重新启动工作节点，以防止将卷附加到容器时出现故障。

iSCSI 卷

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

iSCSI 自我修复功能

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

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

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

Trident iSCSI 自我修复功能可以帮助防止：

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

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

- 缺少 iSCSI 会话
- 缺少 LUN

升级 **Trident** 前需要考虑的要点

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

安装 **iSCSI** 工具

使用操作系统命令安装 iSCSI 工具。

开始之前

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

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

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

RHEL 8+

1. 安装以下系统软件包：

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

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

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

3. 将扫描设置为手动：

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

4. 启用多路径：

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

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

5. 确保 `iscsid` 和 `multipathd` 正在运行：

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

6. 启用并启动 `iscsi`：

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

Ubuntu

1. 安装以下系统软件包：

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

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

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

3. 将扫描设置为手动：

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

4. 启用多路径：

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

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

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

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

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

配置或禁用 **iSCSI** 自我修复

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

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

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

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

Helm

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

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

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

tridentctl

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

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

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

NVMe/TCP 卷

使用操作系统命令安装 NVMe 工具。

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

RHEL 9

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

Ubuntu

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

验证安装

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

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

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

基于 FC 卷的 SCSI

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

前提条件

配置 FC 所需的网络和节点设置。

网络设置

1. 获取目标接口的 WWPN。有关详细信息，请参见 "network interface show"。
2. 获取启动器（主机）上接口的 WWPN。

   请参见相应的主机操作系统实用程序。

3. 使用主机和目标的 WWPN 在 FC 交换机上配置分区。

   有关信息，请参见相应的交换机供应商文档。

   有关详细信息，请参阅以下 ONTAP 文档：

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

安装 FC 工具

使用适用于您的操作系统的命令安装 FC 工具。

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

RHEL 8+

1. 安装以下系统软件包：

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

2. 启用多路径：

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

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

3. 请确保 `multipathd` 正在运行：

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

Ubuntu

1. 安装以下系统软件包：

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

2. 启用多路径：

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

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

3. 确保 `multipath-tools` 已启用并正在运行：

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

准备配置 SMB 卷

您可以使用 `ontap-nas` 驱动程序配置 SMB 卷。

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

`autoExportPolicy` 不支持 SMB 卷。

开始之前

在设置 SMB 卷之前，必须具有下列内容。

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

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

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

步骤

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

   Amazon FSx for ONTAP 需要 SMB 共享。

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

   a. 如有必要，请为共享创建目录路径结构。

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

   b. 创建与指定 SVM 关联的 SMB 共享：

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

   c. 验证是否已创建此共享：

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

有关详细信息，请参见 "创建 SMB 共享"。

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

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

配置和管理后端

配置后端

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

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

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

Azure NetApp Files

配置 **Azure NetApp Files** 后端

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 Azure Kubernetes Services (AKS) 集群的托管身份进行凭据管理。

Azure NetApp Files 驱动程序详细信息

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

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`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。如果请求较小的卷，Trident 会自动创建 50-GiB 的卷。
- Trident 支持安装到在 Windows 节点上运行的 pod 上的 SMB 卷。

AKS 的受管身份

Trident 支持 "托管标识" Azure Kubernetes Services 集群。要利用托管身份提供的简化凭据管理，您必须具备：

- 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 Trident，其中包括 `cloudProvider`以指定 `"Azure"`。

Trident 操作员

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

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

Helm

以下示例使用环境变量 `$CP`将 Trident 集 `cloudProvider`安装到 Azure：

```
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 并将 `cloudProvider` 标志设置为 `Azure`：

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

适用于 **AKS** 的云标识

云身份使 Kubernetes Pod 能够通过作为工作负载身份进行身份验证而不是提供显式 Azure 凭据来访问 Azure 资源。

要利用 Azure 中的云标识，必须具有：

- 使用 AKS 部署的 Kubernetes 集群
- 在 AKS Kubernetes 集群上配置的工作负载标识和 oidc-issuer
- 已安装 Trident，其中包括 `cloudProvider`以指定 `"Azure"`和 `cloudIdentity`指定工作负载身份

Trident 操作员

要使用 Trident 操作员安装 Trident，请编辑 `tridentorchestrator_cr.yaml`以将 `cloudProvider`设置为 `"Azure"`并将 `cloudIdentity`设置为 `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
```

Helm

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

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

以下示例安装 Trident，并使用环境变量 `$CP`将 `cloudProvider`设置为 `Azure`，同时使用环境变量 `$CI`设置 `cloudIdentity`：

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

<code>tridentctl</code>

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

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

以下示例安装 Trident 并将 `cloud-provider`标志设置为 `$CP`，并将 `cloud-identity`设置为 `$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

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

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

> - 在 AKS 集群上使用托管标识时，`subscriptionID`、 `tenantID`、 `clientID`、 `location` 和 `clientSecret` 是可选的。
> - `tenantID`、 `clientID` 和 `clientSecret` 在 AKS 集群上使用云标识时是可选的。

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

自定义贡献者角色

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

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

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

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

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

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

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

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

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

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

"Microsoft.Features/featureProviders/subscriptionFeatureRegistrat
```

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

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

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

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

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

SMB 卷的其他要求

要创建 SMB 卷，必须具有：

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

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

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

Azure NetApp Files 后端配置选项和示例

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

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

Azure NetApp Files 后端提供这些配置选项。

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

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

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

所需权限和资源

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

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

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

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

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

卷配置

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

参数	说明	默认
`exportRule`	新卷的导出规则。`exportRule` 必须是以 CIDR 表示法表示的 IPv4 地址或 IPv4 子网任意组合的逗号分隔列表。对于 SMB 卷将被忽略。	"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 SMB 卷忽略。	"（预览功能，需要在订阅中列入白名单）

示例配置

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

最小配置

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

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

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

AKS 的受管身份

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

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

适用于 **AKS** 的云标识

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

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

具有容量池筛选器的特定服务级别配置

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

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

具有手动 **QoS** 容量池的后端示例

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

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

高级配置

此后端配置进一步将卷放置范围缩小到单个子网，并修改了一些卷配置默认值。

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

虚拟池配置

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

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

支持的拓扑配置

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

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

存储类定义

以下 `StorageClass`定义参考了上述存储池。

使用 `parameter.selector` 字段的定义示例

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

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

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

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

SMB 卷的定义示例

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

默认命名空间的基本配置

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

每个命名空间使用不同的机密

```yaml
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

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

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

创建后端

创建后端配置文件后，运行以下命令：

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

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

```
tridentctl logs
```

在识别并更正配置文件的问题后，您可以再次运行 create 命令。

Google Cloud NetApp Volumes

配置 **Google Cloud NetApp Volumes** 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 后端附加 NFS 和 SMB 卷。

Google Cloud NetApp Volumes 驱动程序详细信息

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

驱动程序	协议	**volumeMode**	支持的访问模式	支持的文件系统
`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而不是提供显式 Google Cloud 凭据来访问 Google Cloud 资源。

要使用 Google Cloud 中的云身份，必须具备以下条件：

- 使用 GKE 部署的 Kubernetes 集群。
- 在 GKE 集群上配置的工作负载标识和在节点池上配置的 GKE MetaData 服务器。
- 具有 Google Cloud NetApp Volumes Admin (roles/netapp.admin) 角色或自定义角色的 GCP 服务帐户。
- Trident 已安装，包括指定 "GCP" 的 cloudProvider 和指定新 GCP 服务帐户的 cloudIdentity。下面给出了一个示例。

Trident 操作员

要使用 Trident 操作员安装 Trident，请编辑 `tridentorchestrator_cr.yaml`以将 `cloudProvider`设置为 `"GCP"`并将 `cloudIdentity`设置为 `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'
```

Helm

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，并使用环境变量 `cloudProvider`将其设置为 `GCP`，同时使用环境变量 `$CP`设置 `cloudIdentity: $ANNOTATION`

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

<code>tridentctl</code>

使用以下环境变量设置 **cloud provider** 和 **cloud identity** 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 并将 `cloud-provider`标志设置为 `$CP`，并将 `cloud-identity`设置为 `$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，需要确保满足以下要求。

NFS 卷的先决条件

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

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

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

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

Google Cloud NetApp Volumes 后端配置选项和示例

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

后端配置选项

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

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	`storageDriverName` 的值必须指定为"google-cloud-netapp-volumes"。
`backendName`	(可选) 存储后端的自定义名称	驱动程序名称 + "_" + API 密钥的一部分
`storagePools`	用于指定卷创建的存储池的可选参数。	
`projectNumber`	Google Cloud 帐户项目编号。该值位于 Google Cloud 门户主页上。	

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

卷配置选项

您可以在配置文件的 defaults 部分中控制默认卷配置。

参数	说明	默认
exportRule	新卷的导出规则。必须是任何 IPv4 地址组合的逗号分隔列表。	"0.0.0.0/0"
snapshotDir	访问 .snapshot 目录	NFSv4 为 "true"，NFSv3 为 "false"
snapshotReserve	为快照预留的卷百分比	" (接受默认值 0)
unixPermissions	新卷的 unix 权限（4 位八进制数字）。	""

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

最小配置

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

当您刚刚开始使用 Google Cloud NetApp Volumes 并尝试使用时，此配置非常理想，但在实践中，您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-----
    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
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-
project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

SMB 卷的配置

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

使用 **StoragePools** 过滤器的配置

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 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多个虚拟池。虚拟池在 `storage` 部分中定义。当您有多个支持不同服务级别的存储池，并且希望在 Kubernetes 中创建表示这些级别的存储类时，它们非常有用。虚拟池标签用于区分池。例如，在下面的示例中， `performance` 标签和 `serviceLevel` 类型用于区分虚拟池。

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

有关详细信息，请参阅"虚拟池"。

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: 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 便于根据区域和可用区为工作负载调配卷。此后端配置中的 `supportedTopologies`块用于为每个后端提供区域和区域的列表。此处指定的区域和区域值必须与每个 Kubernetes 集群节点上标签的区域和区域值匹配。这些区域和区域表示可以在存储类中提供的允许值列表。对于包含后端提供的区域和区域子集的存储类，Trident 会在上述区域和区域中创建卷。有关详细信息，请参阅"使用 CSI 拓扑"。

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

下一步是什么?

创建后端配置文件后，运行以下命令：

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

要验证是否已成功创建后端，请运行以下命令：

```
kubectl get tridentbackendconfig

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

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

```
tridentctl logs
```

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

存储类定义

以下是参考上述后端的基本 `StorageClass` 定义。

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

使用 `**parameter.selector**`字段的定义示例：

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

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

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

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

有关存储类的更多详细信息，请参阅 "创建存储类"。

SMB 卷的定义示例

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

默认命名空间的基本配置

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

每个命名空间使用不同的机密

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

每个卷使用不同的密钥

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

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

PVC 定义示例

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

要验证 PVC 是否已绑定，请运行以下命令：

```
kubectl get pvc gcnv-nfs-pvc

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

配置 NetApp HCI 或 SolidFire 后端

了解如何在 Trident 安装中创建和使用 Element 后端。

Element 驱动程序详细信息

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

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

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

驱动程序	协议	**VolumeMode**	支持的访问模式	支持的文件系统
`solidfire-san`	iSCSI	Filesystem	RWO、RWOP	`xfs`, `ext3`, `ext4`

开始之前

在创建 Element 后端之前，您需要执行下列操作。

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

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称	总是 "solidfire-san"
`backendName`	自定义名称或存储后端	"solidfire_" + 存储 (iSCSI) IP 地址
`Endpoint`	具有租户凭据的 SolidFire 集群的 MVIP	
`SVIP`	存储 (iSCSI) IP 地址和端口	
`labels`	要应用于卷的任意 JSON 格式标签集。	""
`TenantName`	要使用的租户名称（未找到时创建）	
`InitiatorIFace`	将 iSCSI 流量限制到特定主机接口	"default"
`UseCHAP`	使用 CHAP 对 iSCSI 进行身份验证。Trident 使用 CHAP。	true
`AccessGroups`	要使用的访问组 ID 列表	查找名为"trident"的访问组的 ID
`Types`	QoS 规范	
`limitVolumeSize`	如果请求的卷大小高于此值，则配置失败	" （默认情况下不强制执行）
`debugTraceFlags`	用于排除故障的调试标志。例如，{"api":false, "method":true}	空

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

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

此示例显示了一个使用 CHAP 身份验证并使用特定 QoS 保证对三种卷类型进行建模的后端文件。然后，您很可

能会使用 `IOPS` storage class 参数定义要使用其中每个的存储类。

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

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

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

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

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

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

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

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

第一个 StorageClass (`solidfire-gold-four` 将映射到第一个虚拟池。这是唯一提供黄金性能的池，具有

`Volume Type QoS` 为 Gold。最后一个 StorageClass (`solidfire-silver` 调用任何提供银色性能的存储池。Trident 将决定选择哪个虚拟池，并确保满足存储要求。

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

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

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

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

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

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

查找更多信息

- "卷访问组"

ONTAP SAN 驱动程序

ONTAP SAN 驱动程序概述

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

ONTAP SAN 驱动程序详细信息

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

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

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

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

用户权限

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

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

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

NVMe/TCP 的其他注意事项

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

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

Trident 不支持：

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

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

准备使用 **ONTAP SAN** 驱动程序配置后端

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

要求

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

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

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

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

对 **ONTAP** 后端进行身份验证

Trident 提供两种身份验证 ONTAP 后端的模式。

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

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

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

启用基于凭据的身份验证

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

示例后端定义如下所示：

YAML

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

JSON

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

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

启用基于证书的身份验证

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

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

典型的工作流程包括以下步骤。

步骤

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

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

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

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

3. 在 ONTAP 集群上安装客户端证书和密钥（来自步骤 1）。

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

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

4. 确认 ONTAP 安全登录角色支持 `cert`身份验证方法。

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

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

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

6. 使用 Base64 对证书、密钥和可信 CA 证书进行编码。

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

7. 使用从上一步获得的值创建后端。

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

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

更新身份验证方法或轮换凭据

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

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

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

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

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

为 **Trident** 创建自定义 **ONTAP** 角色

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

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

使用 **ONTAP CLI**

1. 使用以下命令创建新角色：

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

2. 为 Trident 用户创建用户名：

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

3. 将角色映射到用户：

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

使用 **System Manager**

在 ONTAP System Manager 中执行以下步骤：

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

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

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

有关详细信息，请参见以下页面：

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

使用双向 **CHAP** 验证连接

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

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

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

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

工作原理

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

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

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

轮换凭据并更新后端

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

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

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

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

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

ONTAP SAN 配置选项和示例

了解如何在 Trident 安装中创建和使用 ONTAP SAN 驱动程序。本节提供了将后端映射到 StorageClasses 的后端配置示例和详细信息。

"ASA r2 系统" 不同于其他 ONTAP 系统（ASA、AFF 和 FAS）的存储层实现。这些变化会影响某些标注参数的使用。"详细了解 ASA r2 系统与其他 ONTAP 系统之间的差异"。

ASA r2 系统仅支持 `ontap-san` 驱动程序（具有 iSCSI、NVMe/TCP 和 FC 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`时，Trident 会自动检测 ASA r2 或其他 ONTAP 系统。某些后端配置参数不适用于 ASA r2 系统，如下表所示。

有关后端配置选项，请参见下表：

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

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

参数	说明	默认
`lunsPerFlexvol`	每个 FlexVol 的最大 LUN 数，必须在 [50, 200] 范围内	`100`
`debugTraceFlags`	故障排除时使用的调试标志。例如，{"api":false, "method":true} 除非正在进行故障排除并需要详细的日志转储，否则不要使用。	`null`
`useREST`	用于使用 ONTAP REST API 的布尔参数。 `useREST`当设置为 `true`时，Trident 使用 ONTAP REST API 与后端通信；当设置为 `false` 时，Trident 使用 ONTAPI (ZAPI) 调用与后端通信。此功能需要 ONTAP 9.11.1 及更高版本。此外，所使用的 ONTAP 登录角色必须具有对 `ontapi`应用程序的访问权限。这通过预定义的 `vsadmin`和 `cluster-admin`角色来满足。从 Trident 24.06 版本和 ONTAP 9.15.1 或更高版本开始，`useREST`默认设置为 `true`；将 `useREST`更改为 `false`以使用 ONTAPI (ZAPI) 调用。 `useREST` 完全符合 NVMe/TCP。 仅 ONTAP REST API 支持 NVMe，ONTAPI (ZAPI) 不支持 NVMe。 如果指定，则对于 **ASA r2** 系统始终设置为 **`true`**。	`true` 适用于 ONTAP 9.15.1 或更高版本，否则 `false`。
`sanType`	用于选择 `iscsi` iSCSI、 `nvme` NVMe/TCP 或 `fcp` 光纤通道 (FC) 上的 SCSI。	`iscsi` 如果为空
`formatOptions`	使用 `formatOptions` 为 `mkfs` 命令指定命令行参数，该参数将在卷格式化时应用。这允许您根据您的偏好格式化卷。请务必指定与 mkfs 命令选项类似的 formatOptions，但不包括设备路径。例如："-E nodiscard" 支持 **`ontap-san`** 和 **`ontap-san-economy`** 驱动程序的 **iSCSI** 协议。*此外，使用 iSCSI 和 NVMe/TCP 协议时，支持 ASA r2 系统。*	
`limitVolumePoolSize`	在 ontap-san-economy 后端中使用 LUN 时可请求的最大 FlexVol 大小。	"（默认情况下不强制执行）

参数	说明	默认
`denyNewVolumePools`	限制 `ontap-san-economy`后端创建包含其 LUN 的新 FlexVol 卷。仅预先存在的 FlexVol 用于配置新的 PV。	

有关使用 **formatOptions** 的建议

Trident 建议使用以下选项来加快格式化过程：

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

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

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

步骤

1. 配置后端 SVM 的域名系统 (DNS) 设置：

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

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

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

3. 使用此命令创建 AD 用户或组以管理集群或 SVM

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

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

用于配置卷的后端配置选项

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

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

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

卷配置示例

以下是定义了默认值的示例：

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

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

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

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

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

当前，对现有卷使用新计算的唯一方法是调整大小。

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

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

ONTAP SAN 示例

这是使用 `ontap-san` 驱动程序的基本配置。

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

MetroCluster 示例

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

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

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

ONTAP SAN 经济示例

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

基于证书的身份验证示例

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

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

双向 **CHAP** 示例

这些示例创建一个后端，其中 `useCHAP` 设置为 `true`。

ONTAP SAN CHAP 示例

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

ONTAP SAN economy CHAP 示例

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

NVMe/TCP 示例

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

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

基于 FC 的 SCSI (FCP) 示例

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

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

带有 **nameTemplate** 的后端配置示例

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

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

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

具有虚拟池的后端示例

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

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

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

池覆盖了默认值。

ONTAP SAN 示例

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

ONTAP SAN 经济示例

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

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

NVMe/TCP 示例

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

将后端映射到 StorageClasses

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

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

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

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

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

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

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

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

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

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

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

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

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

Trident 将决定选择哪个虚拟池，并确保满足存储要求。

ONTAP NAS 驱动程序

ONTAP NAS 驱动程序概述

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

ONTAP NAS 驱动程序详细信息

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

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

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

用户权限

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

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

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

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

准备使用 **ONTAP NAS** 驱动程序配置后端

了解使用 ONTAP NAS 驱动程序配置 ONTAP 后端的要求、身份验证选项和导出策略。

从 25.10 版本开始，NetApp Trident 支持 "NetApp AFX 存储系统"。NetApp AFX 存储系统与其他 ONTAP 系统（ASA、AFF 和 FAS）在存储层的实现方面有所不同。

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

在 Trident 后端配置中，无需指定您的系统是 AFX。当您选择 `ontap-nas`作为 `storageDriverName`时，Trident 会自动检测 AFX 系统。

要求

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

对 **ONTAP** 后端进行身份验证

Trident 提供两种身份验证 ONTAP 后端的模式。

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

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

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

启用基于凭据的身份验证

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

示例后端定义如下所示：

YAML

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

JSON

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

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

启用基于证书的身份验证

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

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

典型的工作流程包括以下步骤。

步骤

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

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

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

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

3. 在 ONTAP 集群上安装客户端证书和密钥（来自步骤 1）。

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

4. 确认 ONTAP 安全登录角色支持 `cert`身份验证方法。

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

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

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

6. 使用 Base64 对证书、密钥和可信 CA 证书进行编码。

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

7. 使用从上一步获得的值创建后端。

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

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

更新身份验证方法或轮换凭据

您可以更新现有后端以使用不同的身份验证方法或轮换其凭据。这可以双向工作：可以将使用用户名/密码的后端更新为使用证书；可以将使用证书的后端更新为基于用户名/密码。为此，您必须删除现有的身份验证方法并添加新的身份验证方法。然后使用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。然后进行后端更新。轮换证书时，可以向用户添加多个证书。然后更新后端以使用新证书，之后可以从 ONTAP 集群中删除旧证书。

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

为 **Trident** 创建自定义 **ONTAP** 角色

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

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

使用 **ONTAP CLI**

1. 使用以下命令创建新角色：

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

2. 为 Trident 用户创建用户名：

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

3. 将角色映射到用户：

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

使用 **System Manager**

在 ONTAP System Manager 中执行以下步骤：

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

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

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

有关详细信息，请参见以下页面：

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

管理 NFS 导出策略

Trident 使用 NFS 导出策略来控制对其提供的卷的访问。

使用出口策略时，Trident 提供两种选择：

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

动态管理导出策略

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

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

示例

必须使用两个配置选项。以下是后端定义示例：

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

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

以下是使用上述示例说明此功能工作原理的解释：

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

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

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

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

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

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

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

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

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

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

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

准备配置 SMB 卷

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

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

`autoExportPolicy` 不支持 SMB 卷。

开始之前

在设置 SMB 卷之前，必须具有下列内容。

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

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

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

步骤

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

   Amazon FSx for ONTAP 需要 SMB 共享。

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

   a. 如有必要，请为共享创建目录路径结构。

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

   b. 创建与指定 SVM 关联的 SMB 共享：

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

c. 验证是否已创建此共享：

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

有关详细信息，请参见 "创建 SMB 共享"。

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

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

启用安全 SMB

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

需要记住的要点

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

步骤

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

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

2. 在存储类中添加批注。

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

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

1. 创建 PVC。

以下示例创建了 PVC：

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

ONTAP NAS 配置选项和示例

了解如何在 Trident 安装中创建和使用 ONTAP NAS 驱动程序。此部分提供了将后端映射到 StorageClasses 的后端配置示例和详细信息。

从 25.10 版本开始，NetApp Trident 支持 "NetApp AFX 存储系统"。NetApp AFX 存储系统与其他基于 ONTAP 的系统（ASA、AFF 和 FAS）在存储层的实现方面有所不同。

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

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

后端配置选项

有关后端配置选项，请参见下表：

参数	说明	默认
`version`		始终为 1
`storageDriverName`	存储驱动程序的名称 对于 NetApp AFX 系统，仅支持 `ontap-nas`。	`ontap-nas`， `ontap-nas-economy`，或 `ontap-nas-flexgroup`

参数	说明	默认
backendName	自定义名称或存储后端	驱动程序名称 + "_" + dataLIF
managementLIF	群集或 SVM 管理 LIF 的 IP 地址可以指定完全限定的域名 (FQDN)。如果使用 IPv6 标志安装了 Trident，可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。有关无缝 MetroCluster 切换，请参见MetroCluster 示例。	“10.0.0.1”，“[2001:1234:abcd::fefe]”
dataLIF	协议 LIF 的 IP 地址。NetApp 建议指定 dataLIF。如果未提供，Trident 会从 SVM 获取 dataLIF。您可以指定要用于 NFS 挂载操作的完全限定域名 (FQDN)，允许您创建循环 DNS 以跨多个 dataLIF 进行负载平衡。可以在初始设置后更改。请参阅。如果使用 IPv6 标志安装了 Trident，可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。*省略 MetroCluster。*请参阅MetroCluster 示例。	指定地址或源自 SVM，如果未指定（不推荐）
svm	要使用的 Storage Virtual Machine *对于 MetroCluster 请省略。*请参阅 MetroCluster 示例。	如果指定了 SVM managementLIF，则派生
autoExportPolicy	启用自动导出策略创建和更新 [Boolean]。使用 autoExportPolicy 和 autoExportCIDRs 选项，Trident 可以自动管理导出策略。	false
autoExportCIDRs	启用 `autoExportPolicy`时用于过滤 Kubernetes 节点 IP 的 CIDR 列表。使用 `autoExportPolicy`和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	["0.0.0.0/0", "::/0"]`
labels	要应用于卷的任意 JSON 格式标签集	""
clientCertificate	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端私钥的 Base64 编码值。用于基于证书的身份验证	""
trustedCACertificate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证	""
username	连接到集群/SVM 的用户名。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证 Trident"。	
password	连接到集群/SVM 的密码。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证 Trident"。	

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

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

用于配置卷的后端配置选项

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

参数	说明	默认
spaceAllocation	Qtree 的空间分配	"true"

参数	说明	默认
spaceReserve	空间预留模式；"none"（精简）或 "volume"（厚）	"无"
snapshotPolicy	要使用的 Snapshot 策略	"无"
qosPolicy	要为创建的卷分配的 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。为每个存储池/后端选择 qosPolicy 或 adaptiveQosPolicy 其中之一。ontap-nas-economy 不支持此功能。	""
snapshotReserve	为快照预留的卷百分比	"0" 如果 snapshotPolicy 为 "none"，否则为 "
splitOnClone	创建时从其父级拆分克隆	"false"
encryption	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 false。必须在群集上许可并启用 NVE 才能使用此选项。如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。有关更多信息，请参阅："Trident 如何与 NVE 和 NAE 配合使用"。	"false"
tieringPolicy	要使用"none"的分层策略	
unixPermissions	新卷的模式	NFS 卷为 "777"；SMB 卷为空（不适用）
snapshotDir	控制对 .snapshot 目录的访问	NFSv4 为 "true"，NFSv3 为 "false"
exportPolicy	要使用的导出策略	"default"
securityStyle	新卷的安全样式。NFS 支持 `mixed`和 `unix`安全样式。SMB 支持 `mixed`和 `ntfs`安全样式。	NFS 默认值为 unix。SMB 默认值为 ntfs。
nameTemplate	用于创建自定义卷名称的模板。	""

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

卷配置示例

以下是定义了默认值的示例：

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

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

计算结果如下所示：

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

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

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

最小配置示例

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

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

ONTAP NAS 经济示例

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

ONTAP NAS FlexGroup 示例

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

MetroCluster 示例

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

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

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

SMB 卷示例

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

基于证书的身份验证示例

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

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

自动导出策略示例

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

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

IPv6 地址示例

此示例显示了 `managementLIF`使用 IPv6 地址。

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

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

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

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

带有 **nameTemplate** 的后端配置示例

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

具有虚拟池的后端示例

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

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

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

ONTAP NAS 示例

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

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

ONTAP NAS FlexGroup 示例

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

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

ONTAP NAS 经济示例

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

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

将后端映射到 StorageClasses

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

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

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

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

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

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

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

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

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

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

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

Trident 将决定选择哪个虚拟池，并确保满足存储要求。

初始配置后更新 `dataLIF`

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

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

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

安全 **SMB** 示例

带有 **ontap-nas** 驱动程序的后端配置

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

使用 **ontap-nas-economy** 驱动程序的后端配置

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

使用存储池的后端配置

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

带有 **ontap-nas** 驱动程序的存储类示例

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

具有 **ontap-nas-economy** 驱动程序的存储类示例

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

单个 **AD** 用户的 **PVC** 示例

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

具有多个 **AD** 用户的 **PVC** 示例

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

Amazon FSx for NetApp ONTAP

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

"Amazon FSx for NetApp ONTAP" 是一项完全托管的 AWS 服务，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、性能和管理功能，同时利用在 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，类似于内部的 ONTAP 集群。在每个 SVM 中，您可以创建一个或多个卷，这些卷是在文件系统中存储文件和文件夹的数据容器。Amazon FSx for NetApp ONTAP 将作为云中的托管文件系统提供。新的文件系统类型称为 **NetApp ONTAP**。

使用 Trident 和 Amazon FSx for NetApp ONTAP，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

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

- 已安装 `kubectl`的现有 Amazon EKS 集群或自我管理的 Kubernetes 集群。
- 可从群集的工作节点访问的现有 Amazon FSx for NetApp ONTAP 文件系统和 Storage Virtual Machine (SVM)。
- 已为 "NFS 或 iSCSI" 准备好的工作节点。

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

注意事项

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

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

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

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

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

身份验证

Trident 提供了两种身份验证模式。

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

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

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

有关启用身份验证的详细信息，请参阅您的驱动程序类型的身份验证：

- "ONTAP NAS 身份验证"
- "ONTAP SAN 身份验证"

已测试的 **Amazon Machine Images (AMIs)**

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

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

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

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

执行测试使用：

- EKS 版本：1.32
- 安装方法：Helm 25.06 和作为 AWS 插件 25.06
- 对于 NAS，已测试 NFSv3 和 NFSv4.1。

- 对于 SAN，仅测试了 iSCSI，而未测试 NVMe-oF。

已执行的测试：

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

查找更多信息

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

创建 IAM 角色和 AWS Secret

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

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

创建 AWS Secrets Manager 密钥

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

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

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

创建 IAM 策略

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

以下示例使用 AWS CLI 创建 IAM 策略：

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

策略 JSON 示例：

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

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

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

Pod 身份

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

在 **EKS** 集群上安装 **Pod Identity**：

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

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

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

创建 **trust-relationship.json**：

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

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

trust-relationship.json 文件：

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

将角色策略附加到 **IAM** 角色：

将上一步中的角色策略附加到已创建的 IAM 角色：

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

创建 **pod** 标识关联：

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

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

服务帐户关联 **(IRSA)** 的 **IAM** 角色

使用 **AWS CLI**：

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

trust-relationship.json 文件：

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

更新 `trust-relationship.json` 文件中的以下值：

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

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

将 **IAM** 角色与 **IAM** 策略绑定：

创建角色后，使用此命令将策略（在上述步骤中创建的）附加到角色：

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

验证 **OICD** 提供者是否关联：

验证您的 OIDC 提供程序是否与您的集群关联。您可以使用以下命令验证它：

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

如果输出为空，请使用以下命令将 IAM OIDC 关联到您的群集：

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

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

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

安装 **Trident**

Trident 简化了 Kubernetes 中 Amazon FSx for NetApp ONTAP 存储管理，使您的开发人员和管理员能够专注于应用程序部署。

您可以使用以下方法之一安装 Trident：

- Helm
- EKS 附加项

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

通过 **helm** 安装 **Trident**

Pod 身份

1. 添加 Trident Helm 存储库：

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

2. 请使用以下示例安装 Trident：

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

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

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

服务账户关联 (IRSA)

1. 添加 Trident Helm 存储库：

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

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

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

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

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION    UPDATED
STATUS       CHART                          APP VERSION

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

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

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

通过 **EKS** 附加组件安装 **Trident**

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

前提条件

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

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

为 **AWS** 启用 **Trident** 加载项

管理控制台

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

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

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

```
{

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

}
```

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

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

   加载项安装完成后，您将看到已安装的加载项。

AWS CLI

1.创建 **`add-on.json`**文件：

对于 **Pod Identity**，请使用以下格式：

使用

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

对于 **IRSA** 身份验证，请使用以下格式：

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

将 `<role ARN>` 替换为上一步中创建的角色的 ARN。

2.安装 **Trident EKS** 附加组件。

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

eksctl

以下示例命令安装 Trident EKS 加载项：

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

更新 **Trident EKS** 附加组件

管理控制台

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

AWS CLI

以下示例更新 EKS 加载项：

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

eksctl

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

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

输出示例：

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

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

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

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

卸载/删除 **Trident EKS** 插件

您有两种删除 Amazon EKS 加载项的选项：

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

如果附加组件具有关联的 IAM 帐户，则不会删除该 IAM 帐户。

管理控制台

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

AWS CLI

将 `my-cluster` 替换为群集的名称，然后运行以下命令。

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

eksctl

以下命令卸载 Trident EKS 加载项：

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

ONTAP SAN 和 NAS 驱动程序集成

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

YAML

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

JSON

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

运行以下命令以创建和验证 Trident Backend Configuration (TBC)：

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

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

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

- 验证已成功创建 Trident 后端配置 (TBC)：

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

```
NAME                           BACKEND NAME          BACKEND UUID
PHASE   STATUS

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

FSx for ONTAP 驱动程序详细信息

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

- `ontap-san`：每个预配置的 PV 都是其自己的 Amazon FSx for NetApp ONTAP 卷中的 LUN。推荐用于块存储。
- `ontap-nas`：配置的每个 PV 都是完整的 Amazon FSx for NetApp ONTAP 卷。推荐用于 NFS 和 SMB。
- `ontap-san-economy`：每个配置的 PV 都是一个 LUN，每个 Amazon FSx for NetApp ONTAP 卷具有可配置数量的 LUN。
- `ontap-nas-economy`：配置的每个 PV 都是一个 qtree，每个 Amazon FSx for NetApp ONTAP 卷的 qtree 数量可配置。
- `ontap-nas-flexgroup`：设置的每个 PV 都是完整的 Amazon FSx for NetApp ONTAP FlexGroup 卷。

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

创建配置文件后，运行此命令在 EKS 中创建它：

```
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

有关后端配置选项，请参见下表：

参数	说明	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，也可以指定完全限定域名（FQDN）。如果 Trident 是使用 IPv6 标志安装的，则可以设置为使用 IPv6 地址。IPv6 地址必须用方括号括起来，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。如果你在 `fsxFilesystemID`下的 `aws` 字段中提供了 `managementLIF`，则无需再提供 `managementLIF`，因为 Trident 会从 AWS 检索 SVM 信息。因此，你必须为 SVM 下的用户（例如：vsadmin）提供凭据，并且该用户必须具有 `vsadmin`角色。	“10.0.0.1”，“[2001:1234:abcd::fefe]”

参数	说明	示例
`dataLIF`	协议 LIF 的 IP 地址。**ONTAP NAS** 驱动程序：NetApp 建议指定 dataLIF。如果未提供，Trident 会从 SVM 获取 dataLIF。您可以指定要用于 NFS 挂载操作的完全限定域名(FQDN)，允许您创建轮询 DNS 以跨多个 dataLIF 进行负载平衡。可以在初始设置后更改。请参阅。**ONTAP SAN** 驱动程序：不要为 iSCSI 指定。Trident 使用 ONTAP Selective LUN Map 来发现建立多路径会话所需的 iSCI LIF。如果明确定义了 dataLIF，则会生成警告。如果使用 IPv6 标志安装了 Trident，则可以设置为使用 IPv6 地址。IPv6 地址必须在方括号中定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。	
`autoExportPolicy`	启用自动导出策略创建和更新 [Boolean]。使用 `autoExportPolicy` 和 `autoExportCIDRs` 选项，Trident 可以自动管理导出策略。	`false`
`autoExportCIDRs`	启用 `autoExportPolicy`时用于过滤 Kubernetes 节点 IP 的 CIDR 列表。使用 `autoExportPolicy`和 `autoExportCIDRs`选项，Trident 可以自动管理导出策略。	"["0.0.0.0/0", "::/0"]"
`labels`	要应用于卷的任意 JSON 格式标签集	""
`clientCertificate`	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
`clientPrivateKey`	客户端私钥的 Base64 编码值。用于基于证书的身份验证	""
`trustedCACertificate`	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""
`username`	连接到集群或 SVM 的用户名。用于基于凭据的身份验证。例如，vsadmin。	
`password`	连接到集群或 SVM 的密码。用于基于凭据的身份验证。	
`svm`	要使用的 Storage Virtual Machine	如果指定了 SVM managementLIF，则派生。

参数	说明	示例
`storagePrefix`	在 SVM 中配置新卷时使用的前缀。创建后无法修改。要更新此参数，您需要创建一个新的后端。	`trident`
`limitAggregateUsage`	*请勿为 Amazon FSx for NetApp ONTAP 指定。*提供的 `fsxadmin` 和 `vsadmin`不包含检索聚合使用情况并使用 Trident 限制它所需的权限。	请勿使用。
`limitVolumeSize`	如果请求的卷大小高于此值，则设置失败。还限制其为 qtree 和 LUN 管理的卷的最大大小，并且该 `qtreesPerFlexvol`选项允许自定义每个 FlexVol 卷的最大 qtree 数	" （默认情况下不强制执行）
`lunsPerFlexvol`	每个 FlexVol volume 的最大 LUN 数必须在 [50, 200] 范围内。仅限 SAN。	“100”
`debugTraceFlags`	故障排除时使用的调试标志。例如，{"api":false, "method":true} 除非正在进行故障排除并需要详细的日志转储，否则不要使用 `debugTraceFlags`。	空
`nfsMountOptions`	NFS 挂载选项的逗号分隔列表。Kubernetes 持久卷的挂载选项通常在存储类中指定，但如果存储类中未指定挂载选项，则 Trident 将回退到使用存储后端配置文件中指定的挂载选项。如果存储类或配置文件中未指定挂载选项，Trident 将不会在关联的持久卷上设置任何挂载选项。	""
`nasType`	配置 NFS 或 SMB 卷创建。选项为 `nfs`、 `smb`或 `null`。*对于 SMB 卷，必须设置为 `smb`。*设置为 null 默认为 NFS 卷。	`nfs`
`qtreesPerFlexvol`	每个 FlexVol 卷的最大 Qtree 数，必须在 [50, 300] 范围内	`"200"`
`smbShare`	您可以指定以下选项之一：使用 Microsoft Management Console 或 ONTAP CLI 创建的 SMB 共享的名称，或允许 Trident 创建 SMB 共享的名称。Amazon FSx for ONTAP 后端需要此参数。	`smb-share`

参数	说明	示例
useREST	使用 ONTAP REST API 的布尔参数。当设置为 `true`时，Trident 将使用 ONTAP REST API 与后端进行通信。此功能需要 ONTAP 9.11.1 及更高版本。此外，所使用的 ONTAP 登录角色必须能够访问 `ontap`应用程序。这通过预定义的 `vsadmin`和 `cluster-admin`角色来满足。	false
aws	您可以在 AWS FSx for ONTAP 的配置文件中指定以下内容：- fsxFilesystemID：指定 AWS FSx 文件系统的 ID。- apiRegion：AWS API 区域名称。- apikey：AWS API 密钥。- secretKey：AWS 密钥。	"" "" ""
credentials	指定要存储在 AWS Secrets Manager 中的 FSx SVM 凭据。- name：机密的 Amazon 资源名称 (ARN)，其中包含 SVM 的凭据。- type：设置为 awsarn。有关详细信息，请参见"创建 AWS Secrets Manager 密钥"。	

用于配置卷的后端配置选项

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

参数	说明	默认
spaceAllocation	LUN 的空间分配	true
spaceReserve	空间预留模式；"none"（精简）或 "volume"（厚）	none
snapshotPolicy	要使用的 Snapshot 策略	none
qosPolicy	要为创建的卷分配的 QoS 策略组。为每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。在 Trident 中使用 QoS 策略组需要 ONTAP 9.8 或更高版本。您应该使用非共享 QoS 策略组，并确保该策略组单独应用于每个组成部分。共享 QoS 策略组强制执行所有工作负载总吞吐量的上限。	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。为每个存储池或后端选择 qosPolicy 或 adaptiveQosPolicy 中的一个。ontap-nas-economy 不支持此功能。	""

参数	说明	默认
snapshotReserve	为快照"0"保留的卷的百分比	如果 snapshotPolicy 是 none，else ""
splitOnClone	创建时从其父级拆分克隆	false
encryption	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 false。必须在群集上许可并启用 NVE 才能使用此选项。如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。有关更多信息，请参阅："Trident 如何与 NVE 和 NAE 配合使用"。	false
luksEncryption	启用 LUKS 加密。请参阅 "使用 Linux Unified Key Setup (LUKS)"。仅限 SAN。	""
tieringPolicy	要使用的分层策略 none	
unixPermissions	新卷的模式。对于 **SMB** 卷，请留空。	""
securityStyle	新卷的安全样式。NFS 支持 `mixed` 和 `unix`安全样式。SMB 支持 `mixed`和 `ntfs`安全样式。	NFS 默认值为 unix。SMB 默认值为 ntfs。

配置 **SMB** 卷

您可以使用 ontap-nas 驱动程序配置 SMB 卷。在完成ONTAP SAN 和 NAS 驱动程序集成之前，请完成以下步骤："准备配置 SMB 卷"。

配置存储类和 **PVC**

配置 Kubernetes StorageClass 对象并创建存储类，以指导 Trident 如何配置卷。创建一个 PersistentVolumeClaim (PVC)，该 PVC 使用配置的 Kubernetes StorageClass 来请求对 PV 的访问。然后，您可以将 PV 挂载到 pod 上。

创建存储类

配置 **Kubernetes StorageClass** 对象

https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass 对象"^] 对象将 Trident 标识为用于该类的配置程序，并指示 Trident 如何预配卷。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"
```

使用以下示例为使用 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，请将所需的 `mountOptions`添加到存储类：

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

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

创建存储类

步骤

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

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

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

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

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

创建 **PVC**

```
https://kubernetes.io/docs/concepts/storage/persistent-
volumes["_PersistentVolumeClaim_"^]（PVC）是访问集群上 PersistentVolume
的请求。
```

PVC 可以配置为请求特定大小或访问模式的存储。使用关联的 StorageClass，集群管理员可以控制超出 PersistentVolume 大小和访问模式的更多内容——例如性能或服务级别。

创建 PVC 后，您可以将卷安装在 Pod 中。

示例清单

PersistentVolumeClaim 示例清单

这些示例显示了基本的 PVC 配置选项。

带 **RWX** 访问的 **PVC**

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

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

PVC 使用 **iSCSI** 示例

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

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

创建 **PVC**

步骤

1. 创建 PVC。

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

2. 验证 PVC 状态。

```
kubectl get pvc
```

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

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

Trident 属性

这些参数确定应使用哪些 Trident 管理的存储池来配置给定类型的卷。

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

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

[1]：ONTAP Select 系统不支持

部署示例应用程序

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

步骤

1. 在 Pod 中挂载卷。

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

以下示例显示了将 PVC 连接到 pod 的基本配置：基本配置：

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

您可以使用 `kubectl get pod --watch` 监控进度。

2. 验证卷是否已装入 `/my/mount/path`。

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

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

现在您可以删除 Pod 了。Pod 应用程序将不再存在，但卷将保持不变。

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

在 **EKS** 集群上配置 **Trident EKS** 附加组件

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

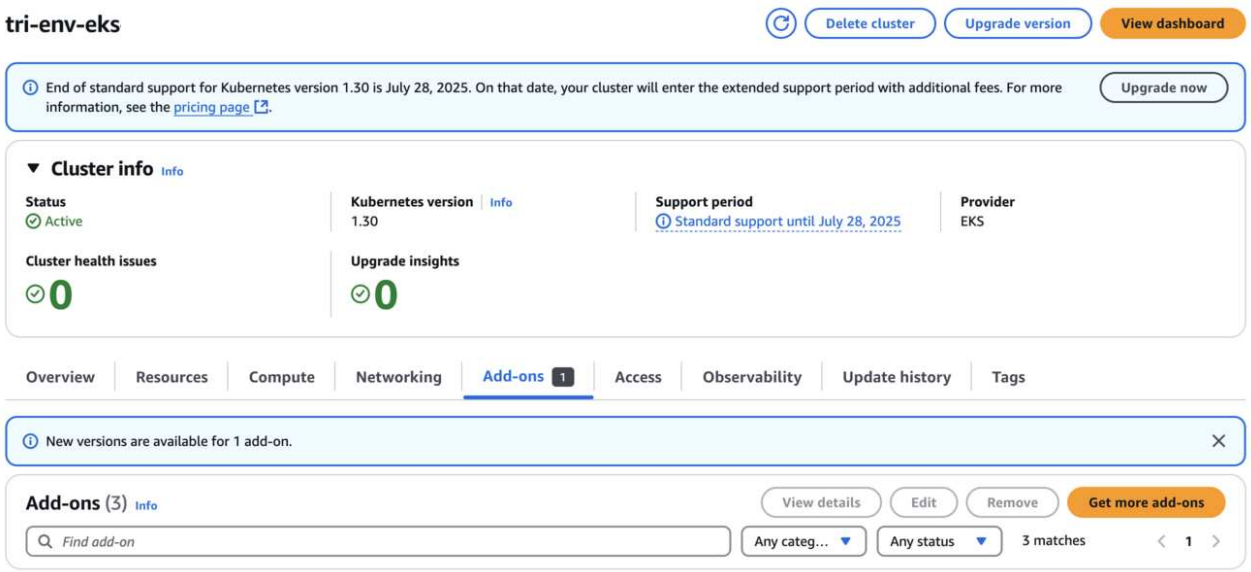
前提条件

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

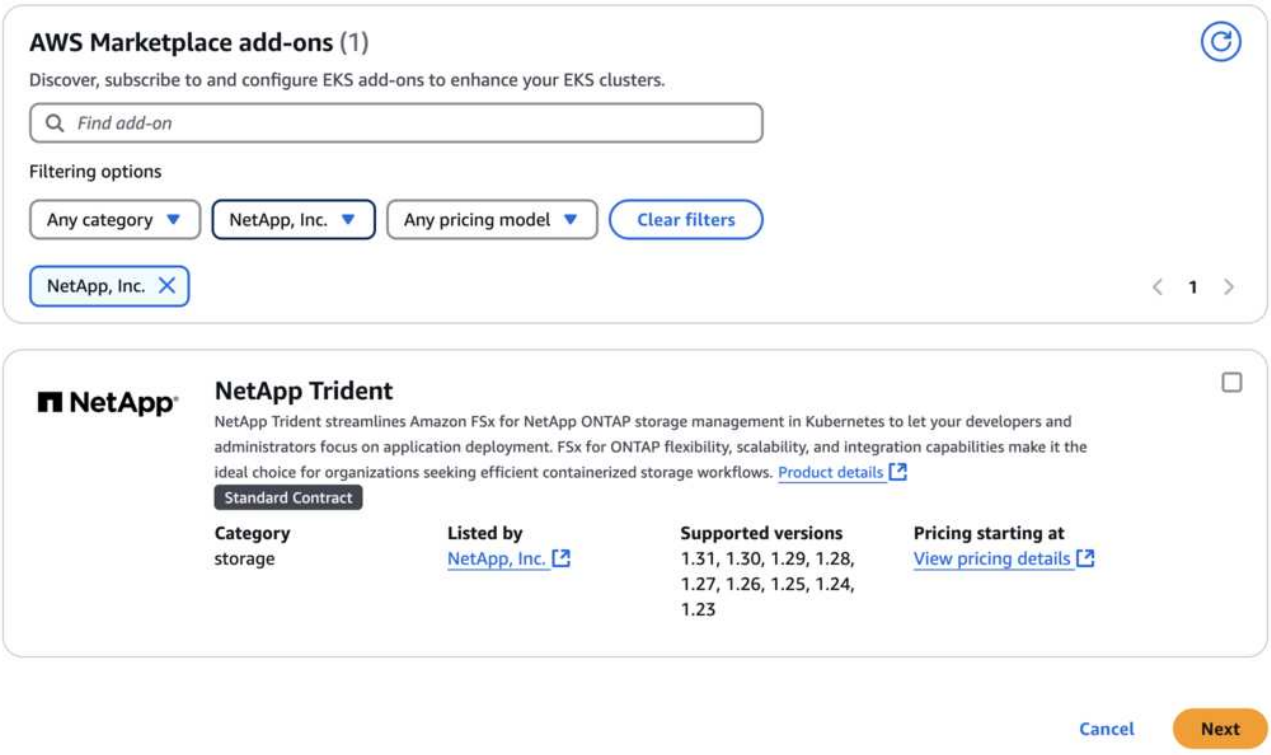
- 具有使用加载项权限的 Amazon EKS 群集帐户。请参阅 "Amazon EKS 附加组件"。
- AWS 对 AWS marketplace 的权限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI 类型：Amazon Linux 2 (AL2_x86_64) 或 Amazon Linux 2 Arm(AL2_ARM_64)
- 节点类型：AMD 或 ARM
- 现有 Amazon FSx for NetApp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

步骤

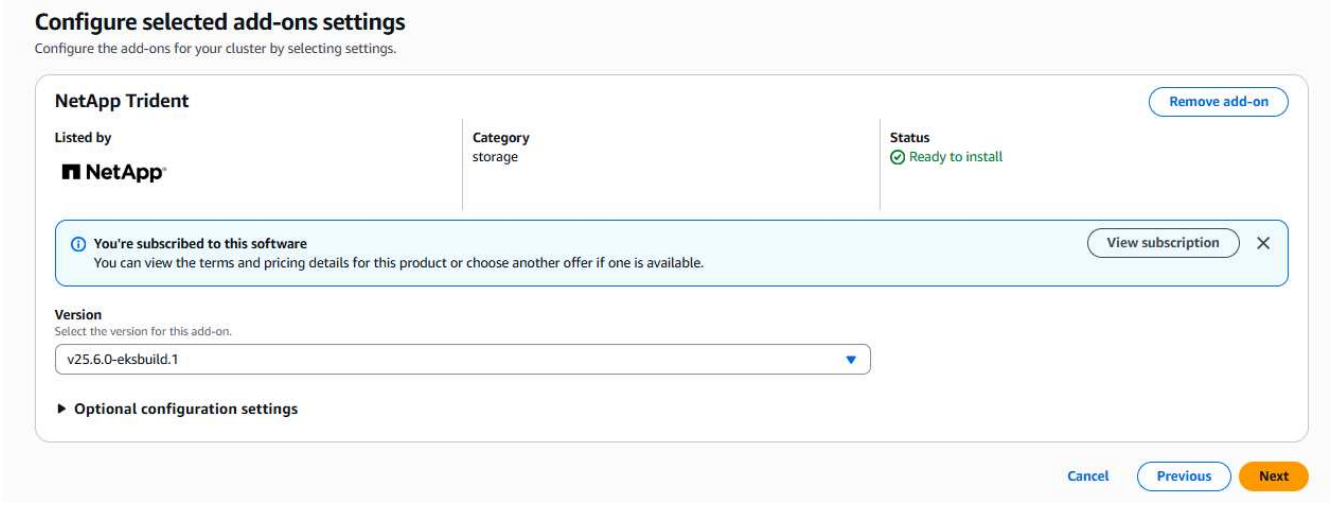
1. 请确保创建 IAM 角色和 AWS 密钥，以使 EKS Pod 能够访问 AWS 资源。有关说明，请参见"创建 IAM 角色和 AWS Secret"。
2. 在 EKS Kubernetes 集群上，导航到 **Add-ons** 选项卡。



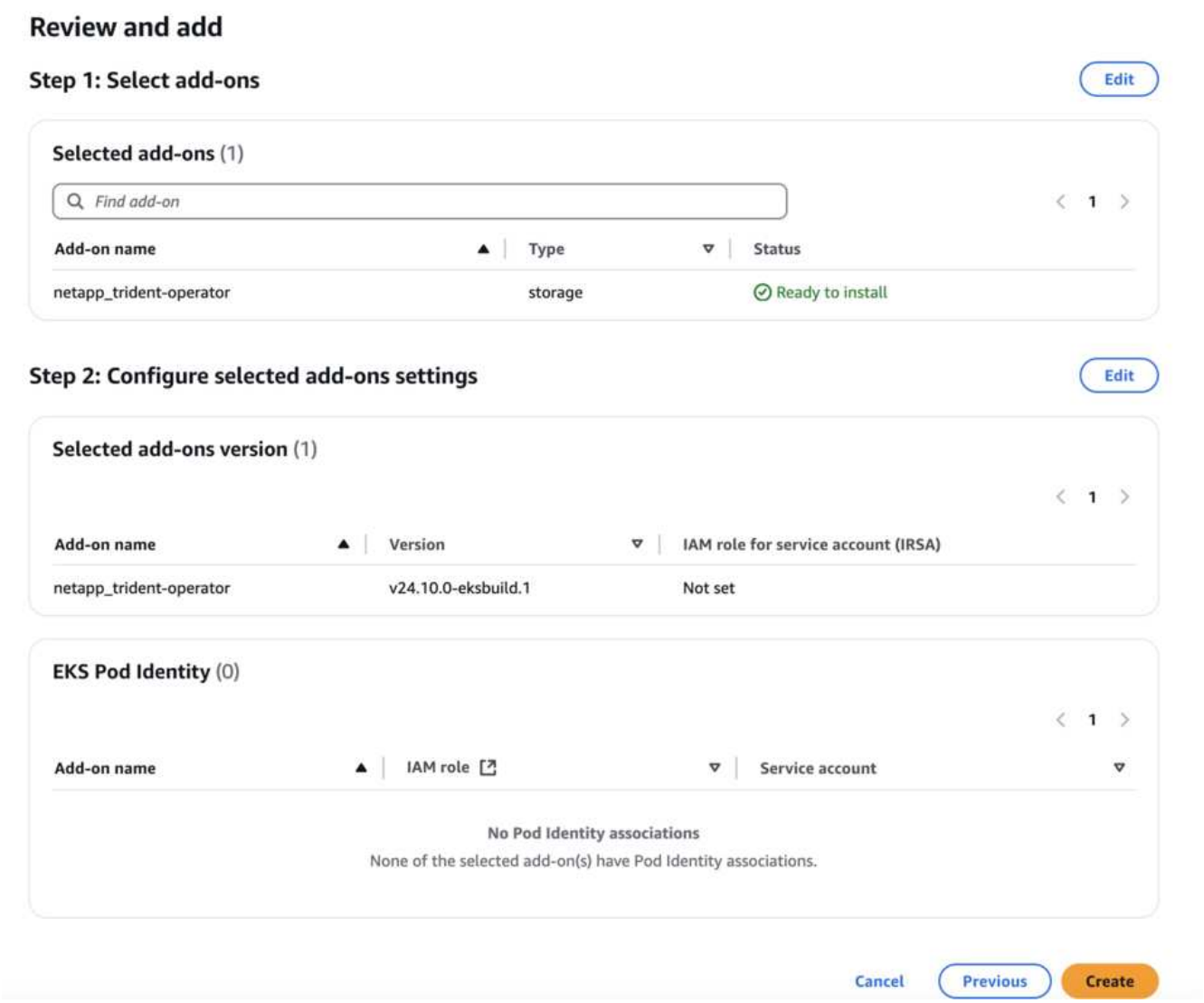
3. 转到 *AWS Marketplace 加载项*并选择 *storage* 类别。



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

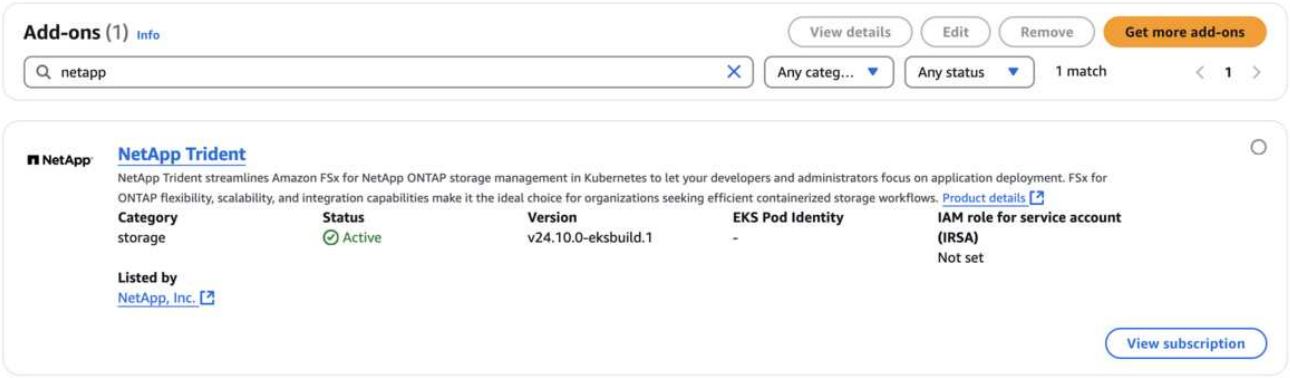


6. 配置所需的附加组件设置。



7. 如果使用 IRSA（服务帐户的 IAM 角色），请参阅其他配置步骤"此处"。
8. 选择 **Create**。

9. 验证加载项的状态是否为 *Active*。



10. 运行以下命令来验证集群上是否正确安装了 Trident：

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

11. 继续设置并配置存储后端。有关信息，请参见"配置存储后端"。

使用 **CLI** 安装/卸载 **Trident EKS** 附加组件

使用 **CLI** 安装 **NetApp Trident EKS** 附加组件：

以下示例命令安装 Trident EKS 加载项：

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

以下示例命令安装 Trident EKS 插件版本 25.6.1：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.1-eksbuild.1（带有专用版本）
```

以下示例命令安装 Trident EKS 插件版本 25.6.2：

```
eksctl create addon --cluster clusterName --name netapp_trident-operator
--version v25.6.2-eksbuild.1（带有专用版本）
```

使用 **CLI** 卸载 **NetApp Trident EKS** 附加组件：

以下命令卸载 Trident EKS 加载项：

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

使用 **kubectl** 创建后端

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

TridentBackendConfig

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

创建 TridentBackendConfig 对象后，会发生以下情况：

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

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

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

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

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

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

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

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

- credentials：此参数为必填字段，包含用于对存储系统/服务进行身份验证的凭据。这将设置为用户创建

的 Kubernetes Secret。凭据无法以纯文本形式传递，将导致错误。

- `deletionPolicy`：此字段定义删除 `TridentBackendConfig` 时应执行的操作。它可以采用以下两种可能的值之一：
  - `delete`：这将导致 `TridentBackendConfig` CR 和相关后端都被删除。这是默认值。
  - `retain`：删除 `TridentBackendConfig` CR 时，后端定义仍将存在，可以使用 `tridentctl` 进行管理。将删除策略设置为 `retain` 允许用户降级到较早版本（21.04 之前）并保留创建的后端。此字段的值可以在创建 `TridentBackendConfig` 后更新。

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

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

步骤概述

要使用 `kubectl` 创建新的后端，应执行以下操作：

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

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

步骤 **1**：创建 **Kubernetes Secret**

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

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

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

下表总结了每个存储平台的 Secret 中必须包含的字段：

存储平台 **Secret** 字段描述	密钥	字段说明
Azure NetApp Files	clientID	来自应用注册的客户端 ID
Element（NetApp HCI/SolidFire）	端点	具有租户凭据的 SolidFire 集群的 MVIP
ONTAP	用户名	连接到集群/SVM 的用户名。用于基于凭据的身份验证
ONTAP	password	连接到集群/SVM 的密码。用于基于凭据的身份验证
ONTAP	clientPrivateKey	客户端私钥的 Base64 编码值。用于基于证书的身份验证
ONTAP	chapUsername	入站用户名。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapInitiatorSecret	CHAP initiator secret。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`
ONTAP	chapTargetInitiatorSecret	CHAP 目标发起者密码。如果 useCHAP=true，则为必需。对于 `ontap-san`和 `ontap-san-economy`

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

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

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

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

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

第 3 步：验证 `TridentBackendConfig` CR 的状态

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

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

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

Phase 可以采用以下其中一个值：

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

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

(可选) 步骤 4：获取更多详细信息

您可以运行以下命令来获取有关后端的详细信息：

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

```
NAME                      BACKEND NAME         BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   8d24fce7-6f60-4d4a-8ef6-
bab2699e6ab8   Bound   Success   ontap-san        delete
```

此外，您还可以获取 `TridentBackendConfig` 的 YAML/JSON 转储。

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

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo` 包含了 `backendName` 和 `backendUUID`，这些是响应 `TridentBackendConfig` CR 创建的后端信息。 `lastOperationStatus` 字段表示 `TridentBackendConfig` CR 最后一次操作的状态，该操作可以由用户触发（例如，用户在 `spec` 中进行了更改）或由 Trident 触发（例如，在 Trident 重启期间）。它可以是 Success 或 Failed。 `phase` 表示 `TridentBackendConfig` CR 与后端之间关系的状态。在上面的示例中， `phase` 的值为 Bound，这意味着 `TridentBackendConfig` CR 已与后端关联。

您可以运行 `kubectl -n trident describe tbc <tbc-cr-name>` 命令以获取事件日志的详细信息。

您无法使用 `tridentctl`更新或删除包含关联 `TridentBackendConfig`对象的后端。要了解在 `tridentctl`和 `TridentBackendConfig`之间切换所涉及的步骤，"请参见此处"。

管理后端

使用 **kubectl** 执行后端管理

了解如何使用 `kubectl` 执行后端管理操作。

删除后端

通过删除 `TridentBackendConfig`，您指示 Trident 删除/保留后端（基于 `deletionPolicy`）。要删除后端，请确保将 `deletionPolicy`设置为 `delete`。要仅删除 `TridentBackendConfig`，请确保将 `deletionPolicy`设置为 retain。这可以确保后端仍然存在，并且可以通过使用 `tridentctl`进行管理。

运行以下命令：

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

Trident 不会删除正在使用的 Kubernetes Secrets `TridentBackendConfig`。Kubernetes 用户负责清理机密。删除机密时必须谨慎行事。仅当机密未被后端使用时，才应将其删除。

查看现有后端

运行以下命令：

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

您还可以运行 `tridentctl get backend -n trident`或 `tridentctl get backend -o yaml -n trident`以获取所有现有后端的列表。此列表还将包括使用 `tridentctl`创建的后端。

更新后端

更新后端可能有多种原因：

- 存储系统的凭据已更改。要更新凭据，必须更新 `TridentBackendConfig` 对象中使用的 Kubernetes Secret。Trident 将使用提供的最新凭据自动更新后端。运行以下命令以更新 Kubernetes Secret：

  ```
  kubectl apply -f <updated-secret-file.yaml> -n trident
  ```

- 需要更新参数（例如正在使用的 ONTAP SVM 的名称）。
  - 您可以使用以下命令直接通过 Kubernetes 更新 `TridentBackendConfig`对象：

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

  - 或者，您可以使用以下命令更改现有 `TridentBackendConfig` CR：

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

- 如果后端更新失败，则后端继续保持其最后已知的配置。您可以通过运行 `kubectl get tbc <tbc-name> -o yaml -n trident`或 `kubectl describe tbc <tbc-name> -n trident`来查看日志以确定原因。
- 确定并更正配置文件的问题后，您可以重新运行更新命令。

使用 **tridentctl** 执行后端管理

了解如何使用 `tridentctl`执行后端管理操作。

创建后端

创建 "后端配置文件" 后，运行以下命令：

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

如果后端创建失败，则表示后端配置有问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在识别并更正配置文件的问题后，只需再次运行此 `create` 命令即可。

删除后端

要从 Trident 删除后端，请执行以下操作：

1. 检索后端名称：

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

2. 删除后端：

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

如果 Trident 已从此后端配置仍然存在的卷和快照，则删除后端会阻止其配置新卷。后端将继续处于"Deleting"状态。

查看现有后端

要查看 Trident 知道的后端，请执行以下操作：

- 要获取摘要，请运行以下命令：

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

- 要获取所有详细信息，请运行以下命令：

```
tridentctl get backend -o json -n trident
```

更新后端

创建新的后端配置文件后，运行以下命令：

```
tridentctl update backend <backend-name> -f <backend-file> -n trident
```

如果后端更新失败，则说明后端配置有问题或您尝试的更新无效。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs -n trident
```

在识别并更正配置文件的问题后，只需再次运行此 `update` 命令即可。

标识使用后端的存储类

这是一个示例，说明您可以使用 `tridentctl` 为后端对象输出的 JSON 来回答这类问题。这使用了 `jq` 实用程序，您需要安装它。

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

这也适用于使用 `TridentBackendConfig` 创建的后端。

在后端管理选项之间移动

了解在 Trident 中管理后端的不同方法。

管理后端的选项

随着 `TridentBackendConfig` 的推出，管理员现在有两种独特的后端管理方式。这就提出了以下问题：

- 使用 `tridentctl`创建的后端可以通过 `TridentBackendConfig`进行管理吗?
- 使用 `TridentBackendConfig`创建的后端可以通过 `tridentctl`进行管理吗?

使用 tridentctl`管理后端 `TridentBackendConfig

本节介绍了管理通过 tridentctl 直接通过 Kubernetes 界面创建 TridentBackendConfig 对象而创建的后端所需的步骤。

这适用于以下情形：

- 已存在的后端，没有 TridentBackendConfig，因为它们是使用 tridentctl 创建的。
- 已使用 `tridentctl`创建新后端，但存在其他 `TridentBackendConfig`对象。

在这两种情况下，后端都将继续存在，Trident 调度卷并对其进行操作。管理员在此处有两种选择：

- 继续使用 tridentctl 来管理使用它创建的后端。
- 将使用 `tridentctl`创建的后端绑定到新的 `TridentBackendConfig`对象。这样做意味着后端将使用 `kubectl`而不是 `tridentctl`进行管理。

要使用 `kubectl`管理预先存在的后端，您需要创建一个 `TridentBackendConfig`绑定到现有后端。以下是其工作原理概述：

1. 创建 Kubernetes Secret。密钥包含 Trident 与存储集群/服务通信所需的凭据。
2. 创建 TridentBackendConfig 对象。其中包含有关存储群集/服务的详细信息，并引用了上一步中创建的密码。必须注意指定相同的配置参数（如 spec.backendName、 spec.storagePrefix、 spec.storageDriverName 等）。 spec.backendName 必须设置为现有后端的名称。

第 0 步：识别后端

要创建绑定到现有后端的 TridentBackendConfig，您需要获取后端配置。在此示例中，假设后端是使用以下 JSON 定义创建的：

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+---------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE   | VOLUMES |
+---------------------+----------------
+--------------------------------------+---------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+---------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
      }
    }
  ]
}
```

步骤 1：创建 **Kubernetes Secret**

创建一个包含后端凭据的 Secret，如以下示例所示：

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

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

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

步骤 2：创建 `TridentBackendConfig` **CR**

下一步是创建一个 `TridentBackendConfig` CR，该 CR 将自动绑定到预先存在的 `ontap-nas-backend`（如本示例所示）。确保满足以下要求：

- 在 `spec.backendName`中定义了相同的后端名称。
- 配置参数与原始后端相同。
- 虚拟池（如果存在）必须保持与原始后端相同的顺序。
- 凭据通过 Kubernetes Secret 提供，而不是以纯文本形式提供。

在此情况下， `TridentBackendConfig` 将如下所示：

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

```yaml
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

第 3 步：验证 `TridentBackendConfig` CR 的状态

创建 `TridentBackendConfig` 后，其阶段必须为 `Bound`。它还应反映与现有后端相同的后端名称和 UUID。

```
kubectl get tbc tbc-ontap-nas-backend -n trident
NAME                    BACKEND NAME          BACKEND UUID
PHASE   STATUS
tbc-ontap-nas-backend  ontap-nas-backend     52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7   Bound   Success

#confirm that no new backends were created (i.e., TridentBackendConfig did
not end up creating a new backend)
tridentctl get backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|          NAME       | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
+--------------------------------------+--------+---------+
```

现在，将使用 tbc-ontap-nas-backend TridentBackendConfig 对象来完全管理后端。

使用 TridentBackendConfig`管理后端 `tridentctl

`tridentctl` 可用于列出使用 `TridentBackendConfig` 创建的后端。此外，管理员还可以选择通过 `tridentctl` 完全管理此类后端，方法是删除 `TridentBackendConfig` 并确保将 `spec.deletionPolicy` 设置为 `retain`。

第 **0** 步：识别后端

例如，让我们假设以下后端是使用 TridentBackendConfig 创建的：

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

从输出中可以看出， `TridentBackendConfig` 已成功创建并绑定到后端 [观察后端的 UUID]。

步骤 **1**：确认 `deletionPolicy` 设置为 `retain`

让我们来看看 `deletionPolicy` 的价值。这需要设置为 `retain`。这可以确保在删除 `TridentBackendConfig` CR 时，后端定义仍然存在，并且可以使用 `tridentctl` 进行管理。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

除非 `deletionPolicy`设置为 `retain`，否则请勿继续下一步。

步骤 2：删除 `TridentBackendConfig` CR

最后一步是删除 `TridentBackendConfig` CR。确认 `deletionPolicy`设置为 `retain`后，您可以继续删除：

```
kubectl delete tbc backend-tbc-ontap-san -n trident
tridentbackendconfig.trident.netapp.io "backend-tbc-ontap-san" deleted

tridentctl get backend ontap-san-backend -n trident
+-------------------+----------------
+--------------------------------------+--------+---------+
|       NAME        | STORAGE DRIVER |                 UUID
| STATE  | VOLUMES |
+-------------------+----------------
+--------------------------------------+--------+---------+
| ontap-san-backend | ontap-san      | 81abcb27-ea63-49bb-b606-
0a5315ac5f82 | online |      33 |
+-------------------+----------------
+--------------------------------------+--------+---------+
```

删除 `TridentBackendConfig` 对象后，Trident 只需将其删除，而不会实际删除后端本身。

创建和管理存储类

创建存储类

配置 Kubernetes StorageClass 对象并创建存储类，以指导 Trident 如何配置卷。

配置 Kubernetes StorageClass 对象

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubernetes StorageClass 对象"^]` 将 `Trident` 识别为用于该类的预配程序，并指示 `Trident` 如何预配卷。例如：

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

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

创建存储类

创建 StorageClass 对象后，可以创建存储类。[存储类示例] 提供了一些可以使用或修改的基本示例。

步骤

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

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

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

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

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

```
./tridentctl -n trident get storageclass basic-csi -o json
```

```
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

存储类示例

Trident 可提供 "用于特定后端的简单存储类定义"。

或者，您可以编辑安装程序附带的 `sample-input/storage-class-csi.yaml.templ` 文件，并将 *`BACKEND_TYPE`* 替换为存储驱动程序名称。

```
./tridentctl -n trident get backend
+-------------+---------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+---------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+---------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

管理存储类

您可以查看现有存储类、设置默认存储类、标识存储类后端以及删除存储类。

查看现有存储类

- 要查看现有的 Kubernetes 存储类，请运行以下命令：

```
kubectl get storageclass
```

- 要查看 Kubernetes 存储类详细信息，请运行以下命令：

```
kubectl get storageclass <storage-class> -o json
```

- 要查看 Trident 的同步存储类，请运行以下命令：

```
tridentctl get storageclass
```

- 要查看 Trident 的同步存储类详细信息，请运行以下命令：

```
tridentctl get storageclass <storage-class> -o json
```

设置默认存储类

Kubernetes 1.6 增加了设置默认存储类的功能。如果用户未在 Persistent Volume Claim (PVC) 中指定存储类，则此存储类将用于配置 Persistent Volume。

- 通过在存储类定义中将注释 `storageclass.kubernetes.io/is-default-class`设置为 true 来定义默认存储类。根据规范，注释的任何其他值或缺失将被解释为 false。
- 您可以使用以下命令将现有存储类配置为默认存储类：

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

- 类似地，您可以使用以下命令删除默认存储类注释：

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

Trident 安装程序包中还有包含此注释的示例。

群集中一次只能有一个默认存储类。从技术上讲，Kubernetes 不会阻止您拥有多个存储类，但它会表现得好像根本没有默认存储类一样。

标识存储类的后端

这是一个示例，说明您可以使用 `tridentctl` 为 Trident 后端对象输出的 JSON 来回答的问题类型。这使用了 `jq` 实用程序，您可能需要先安装它。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

删除存储类

要从 Kubernetes 中删除存储类，请运行以下命令：

```
kubectl delete storageclass <storage-class>
```

`<storage-class>` 应替换为您的存储类。

通过此存储类创建的任何持久卷将保持不变，Trident 将继续管理它们。

Trident 为其创建的卷强制使用空白 `fsType`。对于 iSCSI 后端，建议在 StorageClass 中强制执行 `parameters.fsType`。您应该删除现有的 StorageClasses 并使用指定的 `parameters.fsType`重新创建它们。

配置和管理卷

预配卷

创建一个 PersistentVolumeClaim（PVC），该 PVC 使用配置的 Kubernetes StorageClass 来请求对 PV 的访问权限。然后，您可以将 PV 挂载到 pod 上。

概述

`https://kubernetes.io/docs/concepts/storage/persistent-volumes["_PersistentVolumeClaim_"^]`（PVC）是访问集群上 `PersistentVolume` 的请求。

PVC 可以配置为请求特定大小或访问模式的存储。使用关联的 StorageClass，集群管理员可以控制超出 PersistentVolume 大小和访问模式的更多内容——例如性能或服务级别。

创建 PVC 后，您可以将卷挂载到 pod 中。

创建 **PVC**

步骤

1. 创建 PVC。

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

2. 验证 PVC 状态。

```
kubectl get pvc
```

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

1. 在 Pod 中挂载卷。

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

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

2. 验证卷是否已装入 `/my/mount/path`。

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

3. 现在您可以删除 Pod 了。Pod 应用程序将不再存在，但卷将保持不变。

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

示例清单

PersistentVolumeClaim 示例清单

这些示例显示了基本的 PVC 配置选项。

带 RWO 访问的 PVC

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

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

带 NVMe/TCP 的 PVC

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

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

Pod 清单示例

这些示例显示了将 PVC 连接到 pod 的基本配置。

基本配置

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

基本 **NVMe/TCP** 配置

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

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

扩展卷

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

扩展 iSCSI 卷

您可以使用 CSI 置备程序扩展 iSCSI 永久卷 (PV)。

> ⓘ `ontap-san`、 `ontap-san-economy`、 `solidfire-san` 驱动程序支持 iSCSI 卷扩展，并且需要 Kubernetes 1.16 及更高版本。

步骤 1：配置 StorageClass 以支持卷扩展

编辑 StorageClass 定义以将 `allowVolumeExpansion` 字段设置为 `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的 StorageClass，请对其进行编辑以包含 `allowVolumeExpansion` 参数。

步骤 2：使用创建的 StorageClass 创建 PVC

编辑 PVC 定义并更新 `spec.resources.requests.storage`以反映新期望的尺寸，该尺寸必须大于原始尺寸。

```
cat pvc-ontapsan.yaml
```

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

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (PVC) 相关联。

```
kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc    ontap-san               10s
```

步骤 **3**：定义一个连接 **PVC** 的 **pod**

将 PV 连接到 pod 以调整其大小。调整 iSCSI PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 在扩展操作成功完成后更新 PVC 大小。

在此示例中，创建了一个使用 `san-pvc` 的 Pod。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

步骤 **4**：扩展 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage`更新为 2Gi。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

第 **5** 步：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证扩展是否正常工作：

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

扩展 FC 卷

您可以使用 CSI 配置程序扩展 FC 持久卷 (PV)。

> ⓘ `ontap-san` 驱动程序支持 FC 卷扩展，需要 Kubernetes 1.16 及更高版本。

步骤 1：配置 StorageClass 以支持卷扩展

编辑 StorageClass 定义以将 `allowVolumeExpansion` 字段设置为 `true`。

```
cat storageclass-ontapsan.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的 StorageClass，请对其进行编辑以包含 `allowVolumeExpansion` 参数。

步骤 2：使用创建的 **StorageClass** 创建 **PVC**

编辑 PVC 定义并更新 `spec.resources.requests.storage`以反映新期望的尺寸，该尺寸必须大于原始尺寸。

```
cat pvc-ontapsan.yaml
```

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

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (PVC) 相关联。

```
kubectl get pvc
NAME       STATUS    VOLUME                                     CAPACITY
ACCESS MODES    STORAGECLASS    AGE
san-pvc    Bound     pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO             ontap-san       8s

kubectl get pv
NAME                                       CAPACITY    ACCESS MODES
RECLAIM POLICY    STATUS    CLAIM              STORAGECLASS    REASON    AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi         RWO
Delete            Bound     default/san-pvc    ontap-san                 10s
```

步骤 3：定义一个连接 **PVC** 的 **pod**

将 PV 连接到 pod 以调整其大小。调整 FC PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 在扩展操作成功完成后更新 PVC 大小。

在此示例中，创建了一个使用 `san-pvc` 的 Pod。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-pod   1/1     Running   0          65s

 kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    ubuntu-pod
```

步骤 **4**：扩展 **PV**

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage`更新为 2Gi。

```
kubectl edit pvc san-pvc
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 # ...
```

第 **5** 步：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证扩展是否正常工作：

```
kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
tridentctl get volumes -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

扩展 **NFS** 卷

Trident 支持在 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`和 `azure-netapp-files`后端上配置的 NFS PV 的卷扩展。

步骤 **1**：配置 **StorageClass** 以支持卷扩展

要调整 NFS PV 的大小，管理员首先需要通过将 `allowVolumeExpansion`字段设置为 `true`来配置存储类以允许卷扩展：

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

如果您已经创建了一个没有此选项的存储类，则可以通过使用 `kubectl edit storageclass` 来允许卷扩展

，从而编辑现有存储类。

步骤 2：使用创建的 **StorageClass** 创建 **PVC**

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident 应为此 PVC 创建 20 MiB NFS PV：

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

步骤 3：扩展 **PV**

要将新创建的 20 MiB PV 调整为 1 GiB，请编辑 PVC 并设置 `spec.resources.requests.storage`为 1 GiB：

```
kubectl edit pvc ontapnas20mb
```

```
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
# ...
```

步骤 **4**：验证扩展

您可以通过检查 PVC、PV 和 Trident 卷的大小来验证调整大小是否正确：

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的持久卷声明 (PVC) 将现有存储卷导入为 Kubernetes PV。

概述和注意事项

您可以将卷导入 Trident 以：

- 容器化应用程序并重用其现有数据集
- 将数据集的克隆用于临时应用程序
- 重建失败的 Kubernetes 集群
- 在灾难恢复期间迁移应用程序数据

注意事项

在导入卷之前，请查看以下注意事项。

- Trident 只能导入 RW（读写）类型的 ONTAP 卷。DP（数据保护）类型卷是 SnapMirror 目标卷。在将卷导入 Trident 之前，您应该中断镜像关系。

- 我们建议导入没有活动连接的卷。要导入正在使用的卷，请克隆该卷，然后执行导入。

这对于块卷尤其重要，因为 Kubernetes 不会意识到以前的连接，并且可以轻松地将活动卷附加到 pod。这可能会导致数据损坏。

- 虽然 `StorageClass`必须在 PVC 上指定，但 Trident 在导入过程中不使用此参数。存储类用于在卷创建期间根据存储特性从可用池中进行选择。由于卷已存在，因此在导入过程中不需要选择池。因此，即使卷存在于与 PVC 中指定的存储类不匹配的后端或池中，导入也不会失败。
- 现有卷大小在 PVC 中确定和设置。卷由存储驱动程序导入后，PV 将使用 ClaimRef 创建到 PVC。
  - 回收策略最初在 PV 中设置为 `retain`。Kubernetes 成功绑定 PVC 和 PV 后，回收策略将更新为与 Storage Class 的回收策略匹配。
  - 如果存储类的回收策略为 `delete`，则在删除 PV 时将删除存储卷。
- 默认情况下，Trident 管理 PVC 并重命名后端上的 FlexVol 卷和 LUN。您可以传递 `--no-manage` 标记以导入非托管卷，并传递 `--no-rename` 标记以保留卷名称。
  - `--no-manage`* - 如果您使用 `--no-manage` 标志，Trident 不会在对象生命周期中对 PVC 或 PV 执行任何附加操作。删除 PV 时不会删除存储卷，也会忽略其他操作，如卷克隆和卷大小调整。
  - `--no-rename`* - 如果使用 `--no-rename` 标志，Trident 保留现有卷名，同时导入卷并管理卷的生命周期。只有 `ontap-nas`、 `ontap-san`（包括 ASA r2 系统）和 `ontap-san-economy` 驱动程序才支持此选项。

如果要将 Kubernetes 用于容器化工作负载，但要管理 Kubernetes 之外的存储卷的生命周期，则这些选项非常有用。

- 在 PVC 和 PV 上添加了一个注释，该注释具有双重目的，用于指示已导入该卷以及是否已管理 PVC 和 PV。不应修改或删除此注释。

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的 PVC 来导入卷。

如果使用 PVC 注释，则无需下载或使用 `tridentctl`来导入卷。

使用 **tridentctl**

步骤

1. 创建一个 PVC 文件（例如， `pvc.yaml`），该文件将用于创建 PVC。PVC 文件应包括 `name`、`namespace`、 `accessModes`和 `storageClassName`。或者，您可以在 PVC 定义中指定 `unixPermissions`。

   以下是最小规格的示例：

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

   仅包括所需的参数。PV 名称或卷大小等附加参数可能导致导入命令失败。

2. 使用 `tridentctl import`命令可指定包含卷的 Trident 后端的名称以及唯一标识存储上卷的名称（例如：ONTAP FlexVol、Element Volume）。指定 PVC 文件的路径需要 `-f`参数。

   ```
   tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
   ```

使用 **PVC** 注释

步骤

1. 使用所需的 Trident 导入注释创建 PVC YAML 文件（例如， `pvc.yaml`）。PVC 文件应包括：

   - `name` 和 `namespace` 在元数据中
   - `accessModes`、 `resources.requests.storage` 和 `storageClassName` 在规格中
   - 标注：
     - `trident.netapp.io/importOriginalName`: 后端上的卷名称
     - `trident.netapp.io/importBackendUUID`：卷存在的后端 UUID
     - `trident.netapp.io/notManaged` (*Optional*)：为非托管卷设置为 `"true"`。默认值为 `"false"`。

     下面是导入托管卷的示例规范：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. 将 PVC YAML 文件应用到 Kubernetes 集群：

```
kubectl apply -f <pvc-file>.yaml
```

Trident 将自动导入卷并将其绑定到 PVC。

示例

查看以下受支持驱动程序的卷导入示例。

ONTAP NAS 和 ONTAP NAS FlexGroup

Trident 支持使用 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序进行卷导入。

- Trident 不支持使用 `ontap-nas-economy` 驱动程序进行卷导入。
- `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名。

使用 `ontap-nas` 驱动程序创建的每个卷都是 ONTAP 集群上的一个 FlexVol 卷。使用 `ontap-nas` 驱动程序导入 FlexVol 卷的工作原理相同。ONTAP 集群上已存在的 FlexVol 卷可以作为 `ontap-nas` PVC 导入。同样，FlexGroup 卷可以作为 `ontap-nas-flexgroup` PVC 导入。

使用 **tridentctl** 的 **ONTAP NAS** 示例

以下示例演示如何使用 `tridentctl` 导入托管卷和非托管卷。

托管卷

以下示例导入名为 `managed_volume` 的卷到名为 `ontap_nas` 的后端：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

非受管卷

使用 `--no-manage`参数时， Trident 不会重命名卷。

以下示例在 `ontap_nas` 后端导入 `unmanaged_volume`：

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

使用 **PVC** 注释的 **ONTAP NAS** 示例

以下示例演示如何使用 PVC 注释导入托管和非托管卷。

托管卷

以下示例从后端 `81abcb27-ea63-49bb-b606-0a5315ac5f21`导入名为 `ontap_volume1`的 1GiB `ontap-nas` 卷，使用 PVC 注释设置了 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-
0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

非受管卷

以下示例使用 PVC 注释从后端 `34abcb27-ea63-49bb-b606-0a5315ac5f34`导入名为 `ontap-volume2`的 1Gi `ontap-nas` 卷，并设置 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <umanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-
0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

ONTAP SAN

Trident 支持使用 `ontap-san`（iSCSI、NVMe/TCP 和 FC）和 `ontap-san-economy`驱动程序进行卷导入。

Trident 可以导入包含单个 LUN 的 ONTAP SAN FlexVol 卷。这与 `ontap-san`驱动程序一致，该驱动程序为每个 PVC 创建一个 FlexVol 卷，并在 FlexVol 卷中创建一个 LUN。Trident 导入 FlexVol 卷并将其与 PVC 定义相关联。Trident 可以导入包含多个 LUN 的 `ontap-san-economy`卷。

以下示例显示了如何导入托管卷和非托管卷：

托管卷

对于托管卷，Trident 会将 FlexVol 卷重命名为 `pvc-<uuid>` 格式，并将 FlexVol 卷中的 LUN 重命名为 `lun0`。

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

```
tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

非受管卷

以下示例在 `ontap_san` 后端导入 `unmanaged_example_volume`：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

如果您的 LUN 映射到与 Kubernetes 节点 IQN 共享 IQN 的 igroup，如以下示例所示，您将收到错误： `LUN already mapped to initiator(s) in this group`。您需要删除启动器或取消映射 LUN 以导入卷。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

Element

Trident 支持 NetApp Element 软件和 NetApp HCI 卷导入，使用 `solidfire-san`驱动程序。

> Element 驱动程序支持重复的卷名。但是，如果存在重复的卷名，Trident 返回错误。作为一种解决方法，请克隆该卷，提供唯一的卷名，然后导入克隆的卷。

以下示例将在后端 `element_default`导入 `element-managed`卷。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |              BACKEND UUID              | STATE  | MANAGED |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident 支持使用 `azure-netapp-files`驱动程序进行卷导入。

> 若要导入 Azure NetApp Files 卷，请通过其卷路径识别卷。卷路径是卷的导出路径在 `:/`之后的部分。例如，如果装载路径为 `10.0.0.2:/importvol1`，则卷路径为 `importvol1`。

以下示例将在后端 `azurenetappfiles_40517`上导入一个 `azure-netapp-files`卷，卷路径为 `importvol1`。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+----------------------------------------+---------+--------------
+----------+----------------------------------------+--------+---------+
```

Google Cloud NetApp Volumes

Trident 支持使用 `google-cloud-netapp-volumes`驱动程序进行卷导入。

以下示例在后端 `backend-tbc-gcnv1` 上导入卷 `testvoleasiaeast1`。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                 |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

当两个卷位于同一区域时，以下示例导入 `google-cloud-netapp-volumes`卷：

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

自定义卷名和标签

使用 Trident，您可以为创建的卷分配有意义的名称和标签。这有助于您识别卷并轻松映射到相应的 Kubernetes 资源 (PVC)。您还可以在后端级别定义模板，用于创建自定义卷名和自定义标签；您创建、导入或克隆的任何卷都将遵守模板。

开始之前

可定制的卷名和标签支持：

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

限制

- 自定义卷名仅与 ONTAP 本地驱动程序兼容。
- 只有 `ontap-san`、 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序才支持自定义标签。
- 自定义卷名称不适用于现有卷。

可定制卷名的关键行为

- 如果由于名称模板中的语法无效而发生故障，则后端创建将失败。但是，如果模板应用失败，将根据现有的命名约定命名卷。
- 当卷使用后端配置中的名称模板命名时，存储前缀不适用。任何所需的前缀值都可以直接添加到模板中。

使用名称模板和标签的后端配置示例

可以在根和/或池级别定义自定义名称模板。

根级别示例

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

Pool 级别示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

命名模板示例

示例 **1**：

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

示例 **2**：

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

需要考虑的要点

1. 对于卷导入，仅当现有卷具有特定格式的标签时，才会更新标签。例如：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 对于托管卷导入，卷名遵循后端定义中根级别定义的名称模板。
3. Trident 不支持使用带有存储前缀的切片操作符。
4. 如果模板没有产生唯一的卷名，Trident 将附加几个随机字符以创建唯一的卷名。
5. 如果 NAS 经济卷的自定义名称长度超过 64 个字符，Trident 将根据现有的命名约定命名卷。对于所有其他 ONTAP 驱动程序，如果卷名超过名称限制，则卷创建过程将失败。

跨命名空间共享 **NFS** 卷

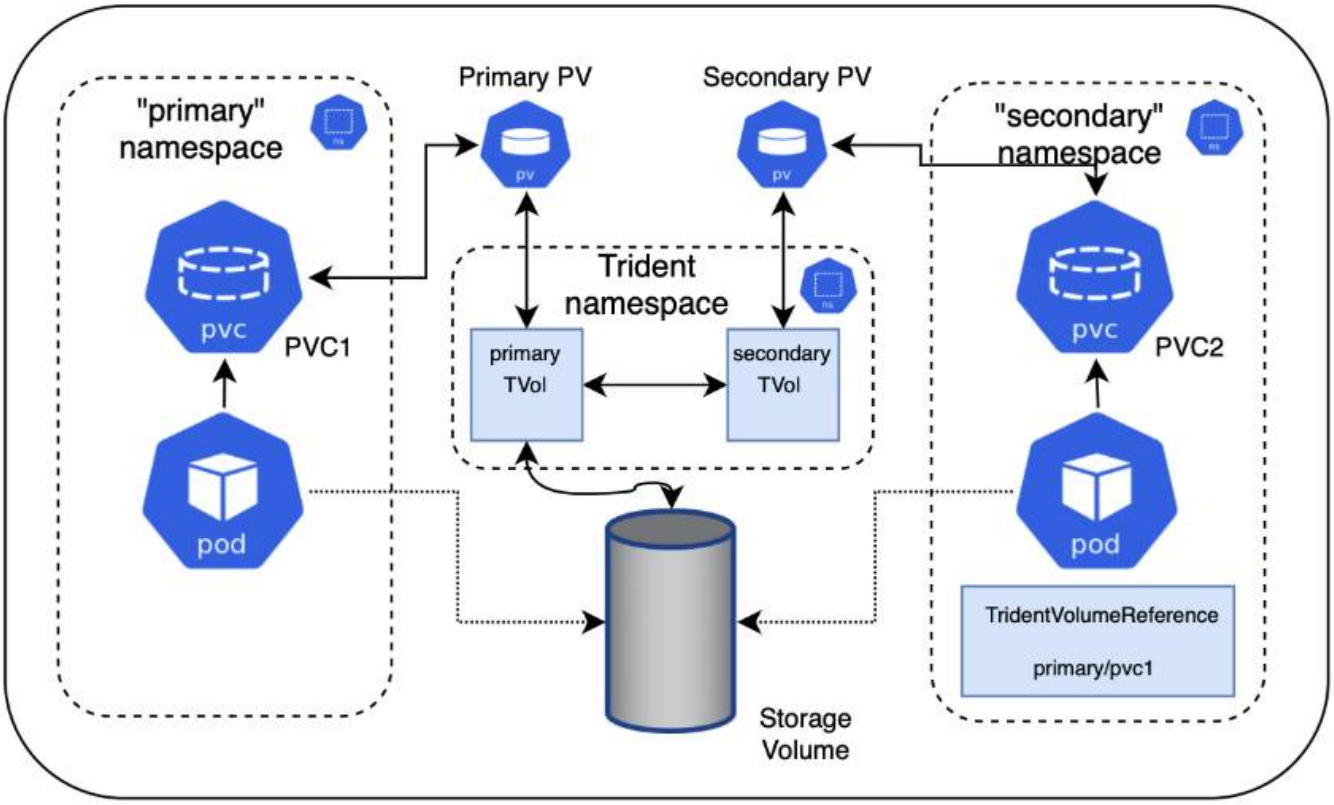
使用 Trident，您可以在主命名空间中创建卷，并在一个或多个辅助命名空间中共享它。

功能

TridentVolumeReference CR 允许您在一个或多个 Kubernetes 命名空间之间安全地共享 ReadWriteMany (RWX) NFS 卷。此 Kubernetes 原生解决方案具有以下优势：

- 多级访问控制，确保安全
- 适用于所有 Trident NFS 卷驱动程序
- 不依赖 tridentctl 或任何其他非本地 Kubernetes 功能

此图说明了两个 Kubernetes 命名空间之间的 NFS 卷共享。



快速入门

只需几个步骤即可设置 NFS 卷共享。

1 配置源 **PVC** 以共享卷

源命名空间所有者授予访问源 PVC 中的数据的权限。

2 授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 TridentVolumeReference CR 的权限。

3 在目标命名空间中创建 **TridentVolumeReference**

目标命名空间的所有者创建 TridentVolumeReference CR 以引用源 PVC。

4 在目标命名空间中创建从属 **PVC**

目标命名空间的所有者创建从属 PVC 以使用来自源 PVC 的数据源。

配置源和目标命名空间

为了确保安全性，跨命名空间共享需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。在每个步骤中指定用户角色。

步骤

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

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident 创建 PV 及其后端 NFS 存储卷。

- 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。例如，`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用 `*` 共享到所有命名空间。例如，`trident.netapp.io/shareToNamespace: *`
- 您可以随时更新 PVC 以包含 `shareToNamespace`注释。

2. *集群管理员：*确保已设置正确的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限。
3. 目标命名空间所有者： 在目标命名空间中创建引用源命名空间的 TridentVolumeReference CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 目标命名空间所有者： 在目标命名空间 (`namespace2`) 中创建一个 PVC (`pvc2`)，并使用 `shareFromPVC` 注解来指定源 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

目标 PVC 的大小必须小于或等于源 PVC。

结果

Trident 读取目标 PVC 上的 `shareFromPVC`注释，并将目标 PV 创建为没有自己的存储资源的从属卷，该卷指向源 PV 并共享源 PV 存储资源。目标 PVC 和 PV 显示为正常绑定。

删除共享卷

您可以删除跨多个命名空间共享的卷。Trident 将删除对源命名空间上卷的访问权限，并保留对共享该卷的其他命名空间的访问权限。当引用该卷的所有命名空间都被删除时，Trident 会删除该卷。

使用 `tridentctl get` 查询从属卷

使用 [`tridentctl` 实用程序，您可以运行 `get` 命令来获取从属卷。有关详细信息，请参见 `tridentctl` 命令和选项。

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

标记：

- `-h, --help`: 卷的帮助。
- `--parentOfSubordinate string`: 将查询限制为从属源卷。
- `--subordinateOf string`: 将查询限制为卷的下属。

限制

- Trident 无法阻止目标命名空间写入共享卷。您应该使用文件锁定或其他进程来防止覆盖共享卷数据。

- 您不能通过删除 `shareToNamespace`或 `shareFromNamespace`注释或删除 `TridentVolumeReference` CR 来撤销对源 PVC 的访问权限。要撤销访问权限，您必须删除从属 PVC。
- 无法在从属卷上执行快照、克隆和镜像。

了解更多信息

要了解有关跨命名空间卷访问的更多信息：

- 访问 "在命名空间之间共享卷：向跨命名空间的卷访问问好"。
- 观看 "NetAppTV" 上的演示。

跨命名空间克隆卷

使用 Trident，您可以使用来自同一 Kubernetes 集群内不同命名空间的现有卷或卷快照创建新卷。

前提条件

在克隆卷之前，请确保源和目标后端的类型相同并且具有相同的存储类。

只有 `ontap-san` 和 `ontap-nas` 存储驱动程序才支持跨命名空间克隆。不支持只读克隆。

快速入门

只需几步即可设置卷克隆。

1 配置源 **PVC** 以克隆卷

源命名空间所有者授予访问源 PVC 中的数据的权限。

2 授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 TridentVolumeReference CR 的权限。

3 在目标命名空间中创建 **TridentVolumeReference**

目标命名空间的所有者创建 TridentVolumeReference CR 以引用源 PVC。

4 在目标命名空间中创建克隆 **PVC**

目标命名空间的所有者创建 PVC 以从源命名空间克隆 PVC。

配置源和目标命名空间

为确保安全性，跨命名空间克隆卷需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。在

每个步骤中都指定了用户角色。

步骤

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

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident 创建 PV 及其后端存储卷。

- 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。例如，`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用 `*` 共享到所有命名空间。例如，`trident.netapp.io/cloneToNamespace: *`
- 您可以随时更新 PVC 以包含 `cloneToNamespace`注释。

2. 集群管理员： 确保已设置正确的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限(`namespace2`）。
3. 目标命名空间所有者： 在目标命名空间中创建引用源命名空间的 TridentVolumeReference CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. 目标命名空间所有者： 在目标命名空间 (`namespace2`) 中创建一个 PVC (`pvc2`)，使用 `cloneFromPVC` 或

cloneFromSnapshot，以及 cloneFromNamespace 注解来指定源 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

限制

- 对于使用 ontap-nas-economy 驱动程序配置的 PVC，不支持只读克隆。

使用 SnapMirror 复制卷

Trident 支持一个集群上的源卷与对等集群上的目标卷之间的镜像关系，用于复制数据以进行灾难恢复。 您可以使用命名空间的自定义资源定义 (CRD)，称为 Trident Mirror Relationship (TMR) 来执行以下操作：

- 创建卷 (PVC) 之间的镜像关系
- 删除卷之间的镜像关系
- 中断镜像关系
- 在灾难情况（故障转移）期间提升辅助卷
- 执行应用程序从集群到集群的无损转换（在计划的故障转移或迁移期间）

复制先决条件

在开始之前，请确保满足以下先决条件：

ONTAP 集群

- **Trident**：Trident 版本 22.10 或更高版本必须存在于使用 ONTAP 作为后端的源和目标 Kubernetes 集群上。
- 许可证：必须在源和目标 ONTAP 集群上启用使用数据保护捆绑包的 ONTAP SnapMirror 异步许可证。有关详细信息，请参见 "ONTAP 中的 SnapMirror 许可概述"。

  从 ONTAP 9.10.1 开始，所有许可证都以 NetApp 许可证文件 (NLF) 的形式交付，该文件是启用多个功能的

单个文件。有关详细信息，请参见 "ONTAP One 附带的许可证"。

> 仅支持 SnapMirror 异步保护。

对等

- 集群和 **SVM**：必须对等 ONTAP 存储后端。有关详细信息，请参见 "集群和 SVM 对等概述"。

> 确保在两个 ONTAP 集群之间的复制关系中使用的 SVM 名称是唯一的。

- **Trident** 和 **SVM**：对等远程 SVM 必须可用于目标集群上的 Trident。

支持的驱动程序

NetApp Trident 支持使用由以下驱动程序支持的存储类的 NetApp SnapMirror 技术进行卷复制： **`ontap-nas`: NFS** `ontap-san`: iSCSI **`ontap-san`: FC** `ontap-san`: NVMe/TCP（需要最低 ONTAP 版本 9.15.1）

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

创建镜像 **PVC**

按照以下步骤并使用 CRD 示例在主卷和二级卷之间创建镜像关系。

步骤

1. 在主 Kubernetes 集群上执行以下步骤：
   a. 使用 `trident.netapp.io/replication: true` 参数创建 StorageClass 对象。

      示例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 使用先前创建的 StorageClass 创建 PVC。

示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. 使用本地信息创建 MirrorRelationship CR。

示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident 获取卷的内部信息和卷的当前数据保护 (DP) 状态，然后填充 MirrorRelationship 的 status 字段。

d. 获取 TridentMirrorRelationship CR 以获取 PVC 的内部名称和 SVM。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. 请在辅助 Kubernetes 集群上执行以下步骤：

   a. 使用 trident.netapp.io/replication: true 参数创建 StorageClass。

   示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

   b. 创建包含目标和源信息的 MirrorRelationship CR。

   示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Trident 将创建一个具有已配置关系策略名称（或 ONTAP 默认值）的 SnapMirror 关系并对其进行初始化。

c. 使用先前创建的 StorageClass 创建 PVC 以充当辅助（SnapMirror 目标）。

示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Trident 将检查 TridentMirrorRelationship CRD，如果关系不存在，则无法创建卷。如果存在此关系，Trident 将确保将新 FlexVol 卷放置到与 MirrorRelationship 中定义的远程 SVM 对等的 SVM 上。

卷复制状态

Trident 镜像关系（TMR）是表示 PVC 之间复制关系的一端的 CRD。目标 TMR 有一个状态，它告诉 Trident 所需的状态是什么。目标 TMR 具有以下状态：

- **Established**：本地 PVC 是镜像关系的目标卷，这是一种新的关系。
- **Promoted**：本地 PVC 是 ReadWrite 且可挂载的，目前没有镜像关系生效。
- 已重新建立：本地 PVC 是镜像关系的目标卷，此前也处于该镜像关系中。
  - 如果目标卷与源卷之间曾经存在关系，则必须使用重新建立的状态，因为它会覆盖目标卷的内容。
  - 如果卷以前未与源建立关系，则重新建立状态将失败。

在计划外故障转移期间提升辅助 **PVC**

在辅助 Kubernetes 集群上执行以下步骤：

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

在计划的故障转移期间推广辅助 **PVC**

在计划的故障转移（迁移）期间，请执行以下步骤来升级辅助 PVC：

步骤

1. 在主 Kubernetes 集群上，创建 PVC 的快照，并等待创建快照。

2. 在主 Kubernetes 集群上，创建 SnapshotInfo CR 以获取内部详细信息。

   示例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. 在辅助 Kubernetes 集群上，将 *TridentMirrorRelationship* CR 的 *spec.state* 字段更新为 *promoted*，将 *spec.promotedSnapshotHandle* 更新为快照的 internalName。
4. 在辅助 Kubernetes 集群上，确认 TridentMirrorRelationship 的状态（status.state 字段）为 promoted。

故障转移后恢复镜像关系

在恢复镜像关系之前，请选择要作为新主要关系的一侧。

步骤

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

其他操作

Trident 支持主卷和二级卷上的以下操作：

将主要 **PVC** 复制到新的次要 **PVC**

请确保已有一个主要 PVC 和一个次要 PVC。

步骤

1. 从已建立的辅助（目标）集群中删除 PersistentVolumeClaim 和 TridentMirrorRelationship CRD。
2. 从主（源）集群中删除 TridentMirrorRelationship CRD。
3. 在要建立的新辅助（目标） PVC 的主（源）集群上创建新的 TridentMirrorRelationship CRD。

调整镜像、主 **PVC** 或辅助 **PVC** 的大小

PVC 可以正常调整大小，如果数据量超过当前大小，ONTAP 将自动扩展任何目标 flexvols。

从 **PVC** 中删除复制

要删除复制，请在当前辅助卷上执行以下操作之一：

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

删除 **PVC**（先前已镜像）

Trident 检查复制的 PVC，并在尝试删除卷之前释放复制关系。

删除 **TMR**

删除镜像关系一侧的 TMR 会导致剩余的 TMR 在 Trident 完成删除之前过渡到 *promoted* 状态。如果选择删除的 TMR 已经处于 *promoted* 状态，则不存在现有的镜像关系，TMR 将被删除，Trident 会将本地 PVC 提升为 *ReadWrite*。此删除将释放 ONTAP 中本地卷的 SnapMirror 元数据。如果将来在镜像关系中使用此卷，则在创建新的镜像关系时，必须使用具有 *established* 卷复制状态的新 TMR。

ONTAP 联机时更新镜像关系

镜像关系建立后可以随时更新。您可以使用 `state: promoted`或 `state: reestablished`字段更新关系。将目标卷升级到常规 ReadWrite 卷时，您可以使用 *promotedSnapshotHandle* 指定要将当前卷还原到的特定快照。

ONTAP 离线时更新镜像关系

您可以使用 CRD 执行 SnapMirror 更新，而无需 Trident 直接连接到 ONTAP 集群。请参阅以下 TridentActionMirrorUpdate 格式示例：

示例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` 反映了 TridentActionMirrorUpdate CRD 的状态。它可以取自 *Succeeded*、*In Progress* 或 *Failed* 的值。

使用 **CSI** 拓扑

Trident 可以通过使用 "CSI 拓扑功能"选择性地创建卷并将其附加到 Kubernetes 集群中的节点。

概述

使用 CSI 拓扑功能，可以根据区域和可用性区域将对卷的访问限制为节点的子集。如今，云提供商使 Kubernetes 管理员能够生成基于区域的节点。节点可以位于一个区域内的不同可用区，也可以位于不同的区域。为了便于在多区域架构中为工作负载配置卷，Trident 使用了 CSI 拓扑。

详细了解 CSI Topology 功能 "此处"。

Kubernetes 提供两种独特的卷绑定模式：

- 当 `VolumeBindingMode`设置为 `Immediate`时，Trident 会在没有任何拓扑感知的情况下创建卷。创建 PVC 时会处理卷绑定和动态配置。这是默认 `VolumeBindingMode`，适用于不强制执行拓扑约束的集群。创建持久卷时不依赖于请求 Pod 的调度要求。
- 将 `VolumeBindingMode` 设置为 `WaitForFirstConsumer` 时，PVC 的 Persistent Volume 的创建和绑定会被延迟，直到使用该 PVC 的 pod 被调度和创建。通过这种方式，创建的卷可以满足拓扑要求所实施的调度约束。

> The `WaitForFirstConsumer` 绑定模式不需要拓扑标签。这可以独立于 CSI 拓扑功能使用。

您需要什么

要使用 CSI 拓扑，您需要以下内容：

- 运行 "支持的 Kubernetes 版本" 的 Kubernetes 集群

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 集群中的节点应具有引入拓扑感知(`topology.kubernetes.io/region`和 `topology.kubernetes.io/zone`)的标签。在安装 Trident 之前，这些标签*应该存在于集群中的节点上*，以便 Trident 具有拓扑感知能力。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

步骤 **1**：创建可感知拓扑的后端

Trident 存储后端可以设计为基于可用性区域有选择地配置卷。每个后端都可以携带一个可选 `supportedTopologies`块，该块表示受支持的区域和地区列表。对于使用此类后端的 StorageClasses，只有在支持的地区/区域中调度的应用程序请求时，才会创建卷。

以下是后端定义示例：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

`supportedTopologies` 用于为每个后端提供区域和可用区的列表。这些区域和可用区表示可以在 StorageClass 中提供的允许值列表。对于包含后端提供的区域和可用区子集的 StorageClasses，Trident 会在后端上创建卷。

您也可以为每个存储池定义 `supportedTopologies`。请参见以下示例：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-backend-us-central1
managementLIF: 172.16.238.5
svm: nfs_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-a
  - topology.kubernetes.io/region: us-central1
    topology.kubernetes.io/zone: us-central1-b
storage:
  - labels:
      workload: production
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

在此示例中， `region` 和 `zone` 标签代表存储池的位置。 `topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone` 指示可以从哪里使用存储池。

步骤 2：定义拓扑感知的 **StorageClasses**

根据提供给集群中节点的拓扑标签，StorageClasses 可以定义为包含拓扑信息。这将确定用作 PVC 请求候选项的存储池，以及可以使用 Trident 配置的卷的节点子集。

请参见以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - us-east1-a
        - us-east1-b
    - key: topology.kubernetes.io/region
      values:
        - us-east1
parameters:
  fsType: ext4
```

在上面提供的 StorageClass 定义中， `volumeBindingMode` 设置为 `WaitForFirstConsumer`。使用此 StorageClass 请求的 PVC 在 pod 中引用之前不会被执行。并且， `allowedTopologies` 提供要使用的区域和地区。 `netapp-san-us-east1` StorageClass 在上面定义的 `san-backend-us-east1` 后端上创建 PVC。

步骤 **3**：创建和使用 **PVC**

创建 StorageClass 并将其映射到后端后，您现在可以创建 PVC。

请参见以下示例 `spec`：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

使用此清单创建 PVC 将导致以下结果：

```
kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

要使 Trident 创建卷并将其绑定到 PVC，请在 Pod 中使用 PVC。请参见以下示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

这个 podSpec 指示 Kubernetes 在 `us-east1` 区域中的节点上调度 pod，并从 `us-east1-a` 或 `us-east1-b` 可用区中的任何节点中进行选择。

请参阅以下输出：

```
kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

更新后端以包括 `supportedTopologies`

可以更新预先存在的后端，以包含 `supportedTopologies`使用 `tridentctl backend update`的列表。这不会影响已配置的卷，仅用于后续 PVC。

查找更多信息

- "管理容器的资源"
- "nodeSelector"
- "亲和性和反亲和性"
- "污点和容忍"

使用快照

持久卷 (PV) 的 Kubernetes 卷快照支持卷的时间点副本。您可以创建使用 Trident 创建的卷的快照，导入在 Trident 外部创建的快照，从现有快照创建新卷，并从快照中恢复卷数据。

概述

`ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san`、 `azure-netapp-files` 和 `google-cloud-netapp-volumes` 驱动程序支持卷快照。

开始之前

必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，请参阅 [部署卷快照控制器]。

如果在 GKE 环境中创建按需卷快照，请勿创建快照控制器。GKE 使用内置的隐藏快照控制器。

创建卷快照

步骤

1. 创建一个 `VolumeSnapshotClass`。有关详细信息，请参见 "VolumeSnapshotClass"。
   - `driver` 指向 Trident CSI 驱动程序。
   - `deletionPolicy` 可以为 `Delete`或 `Retain`。当设置为 `Retain`时，即使删除 `VolumeSnapshot` 对象，也会保留存储集群上的底层物理快照。

   示例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   deletionPolicy: Delete
   ```

2. 创建现有 PVC 的快照。

   示例

   - 此示例创建现有 PVC 的快照。

   ```
   cat snap.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: pvc1-snap
   spec:
     volumeSnapshotClassName: csi-snapclass
     source:
       persistentVolumeClaimName: pvc1
   ```

   - 此示例为名为 `pvc1`的 `PVC` 创建卷快照对象，快照的名称设置为 `pvc1-snap`。VolumeSnapshot 类似于 PVC，并与表示实际快照的 `VolumeSnapshotContent`对象相关联。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

- 您可以通过描述 `VolumeSnapshotContent`对象来识别 `pvc1-snap` VolumeSnapshot。`Snapshot Content Name`标识为此快照提供服务的 VolumeSnapshotContent 对象。 `Ready To Use`参数表示快照可用于创建新的 PVC。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

从卷快照创建 **PVC**

您可以使用 `dataSource`创建 PVC，使用名为 `<pvc-name>`的 VolumeSnapshot 作为数据源。创建 PVC 后，它可以连接到 pod 并像任何其他 PVC 一样使用。

> PVC 将在与源卷相同的后端创建。请参阅 "KB：无法在备用后端从 Trident PVC 快照创建 PVC"。

以下示例使用 `pvc1-snap` 作为数据源创建 PVC。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

导入卷快照

Trident 支持 "Kubernetes 预置 Snapshot 流程" 使集群管理员能够创建 `VolumeSnapshotContent` 对象并导入在 Trident 外部创建的快照。

开始之前

Trident 必须已创建或导入快照的父卷。

步骤

1. 集群管理员： 创建引用后端快照的 `VolumeSnapshotContent` 对象。这将启动 Trident 中的快照工作流。
   - 将 `annotations` 中的后端快照的名称指定为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 在 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` 中指定 `snapshotHandle`。这是外部快照程序在 `ListSnapshots` 调用中提供给 Trident 的唯一信息。

     由于 CR 命名限制， `<volumeSnapshotContentName>` 无法始终匹配后端快照名称。

     示例

     以下示例创建一个 `VolumeSnapshotContent` 对象，该对象引用后端快照 `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. 集群管理员： 创建引用 `VolumeSnapshot`对象的 `VolumeSnapshotContent` CR。这将请求在指定命名空间中使用 `VolumeSnapshot`的访问权限。

   示例

   以下示例创建了一个 `VolumeSnapshot`名为 `import-snap`的 CR，并引用了名为 `import-snap-content`的 `VolumeSnapshotContent`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. 内部处理（无需执行任何操作）： 外部快照器识别新创建的 `VolumeSnapshotContent`并运行 `ListSnapshots`调用。Trident 创建 `TridentSnapshot`。
   - 外部快照器将 `VolumeSnapshotContent` 设置为 `readyToUse`，将 `VolumeSnapshot` 设置为 `true`。
   - Trident 返回 `readyToUse=true`。
4. 任何用户： 创建 `PersistentVolumeClaim`以引用新 `VolumeSnapshot`，其中 `spec.dataSource`（或 `spec.dataSourceRef`）名称是 `VolumeSnapshot`名称。

   示例

以下示例创建了一个 PVC，引用名为 `VolumeSnapshot`的 `import-snap。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

使用快照恢复卷数据

默认情况下隐藏 snapshot 目录，以促进使用 `ontap-nas`和 `ontap-nas-economy`驱动程序配置的卷的最大兼容性。启用 `.snapshot`目录以直接从快照恢复数据。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 `TridentActionSnapshotRestore` (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

Trident 支持在 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`、 `ontap-nas-flexgroup`、 `azure-netapp-files`、 `google-cloud-netapp-volumes` 和 `solidfire-san` 驱动程序上还原快照。

开始之前

您必须具有绑定的 PVC 和可用的卷快照。

- 确认 PVC 状态已绑定。

```
kubectl get pvc
```

- 验证卷快照是否可以使用。

```
kubectl get vs
```

步骤

1. 创建 TASR CR。此示例为 PVC `pvc1` 和卷快照 `pvc1-snapshot`创建 CR。

   TASR CR 必须位于存在 PVC 和 VS 的命名空间中。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. 应用 CR 从快照中恢复。此示例从快照中还原 `pvc1`。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

结果

Trident 从快照中恢复数据。您可以验证快照还原状态：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- 在大多数情况下，出现故障时，Trident 不会自动重试此操作。您需要再次执行此操作。
- 没有管理员访问权的 Kubernetes 用户可能需要管理员授予权限才能在其应用程序命名空间中创建 TASR CR。

删除具有关联快照的 PV

删除具有关联快照的永久卷时，相应的 Trident 卷将更新为"删除状态"。删除卷快照以删除 Trident 卷。

部署卷快照控制器

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

步骤

1. 创建卷快照 CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

如有必要，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`到您的命名空间。

相关链接

- "卷快照"
- "VolumeSnapshotClass"

处理卷组快照

持久卷（PV）的 Kubernetes 卷组快照 NetApp Trident 提供了创建多个卷快照（一组卷快照）的功能。此卷组快照表示在同一时间点拍摄的多个卷的副本。

VolumeGroupSnapshot 是 Kubernetes 中带有 beta API 的 beta 功能。Kubernetes 1.32 是 VolumeGroupSnapshot 所需的最低版本。

创建卷组快照

以下存储驱动程序支持卷组快照：

- `ontap-san` 驱动程序 - 仅适用于 iSCSI 和 FC 协议，不适用于 NVMe/TCP 协议。
- `ontap-san-economy` - 仅适用于 iSCSI 协议。
- `ontap-nas`

NetApp ASA r2 或 AFX 存储系统不支持卷组快照。

开始之前

- 请确保您的 Kubernetes 版本为 K8s 1.32 或更高版本。
- 必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

  如果您的 Kubernetes 发行版不包括外部快照控制器和 CRD，请参阅 [部署卷快照控制器]。

  如果在 GKE 环境中创建按需卷组快照，请勿创建快照控制器。GKE 使用内置的隐藏快照控制器。

- 在快照控制器 YAML 中，将 `CSIVolumeGroupSnapshot` 功能门设置为 'true'，以确保启用卷组快照。
- 在创建卷组快照之前创建所需的卷组快照类。
- 确保所有 PVC/卷都在同一个 SVM 上，以便能够创建 VolumeGroupSnapshot。

步骤

- 在创建 VolumeGroupSnapshot 之前，先创建 VolumeGroupSnapshotClass。有关更多信息，请参阅"VolumeGroupSnapshotClass"。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- 使用现有存储类创建具有所需标签的 PVC，或将这些标签添加到现有 PVC。

  以下示例使用 `pvc1-group-snap` 作为数据源和标签 `consistentGroupSnapshot: groupA` 创建 PVC。根据您的要求定义标签键和值。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1
```

- 创建具有相同标签 (`consistentGroupSnapshot: groupA` 的 VolumeGroupSnapshot，该标签在 PVC 中指定。

  此示例创建卷组快照：

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA
```

使用组快照恢复卷数据

您可以使用作为 Volume Group Snapshot 一部分创建的单个快照来恢复单个 Persistent Volume。您无法将 Volume Group Snapshot 作为一个单位进行恢复。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 `TridentActionSnapshotRestore` (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

有关详细信息，请参见 "从快照就地还原卷"。

删除具有关联组快照的 **PV**

删除组卷快照时：

- 您可以整体删除 VolumeGroupSnapshots，而不是删除组中的单个快照。
- 如果在存在该 PersistentVolume 的快照时删除 PersistentVolumes，Trident 会将该卷移动到“deleting”状态，因为必须先删除快照，才能安全地删除该卷。
- 如果已使用分组快照创建克隆，然后要删除该组，则将开始克隆拆分操作，并且在完成拆分之前无法删除该组。

部署卷快照控制器

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

步骤

1. 创建卷快照 CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.
yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

如有必要，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`到您的命名空间。

相关链接

- "VolumeGroupSnapshotClass"
- "卷快照"

管理和监控 Trident

升级 Trident

升级 Trident

从 24.02 版本开始，Trident 遵循四个月的发布节奏，每个日历年发布三个主要版本。每个新版本都建立在先前版本的基础上，并提供新功能、性能增强、错误修复和改进。我们建议您每年至少升级一次，以利用 Trident 中的新功能。

升级前的注意事项

升级到最新版 Trident 时，请考虑以下事项：

- 在给定的 Kubernetes 集群中的所有命名空间中应仅安装一个 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。
- 升级时，请务必提供 Trident 使用的 `parameter.fsType`中的 `StorageClasses`。您可以删除和重新创建 `StorageClasses`而不会中断先前存在的卷。
  - 这是强制执行 "安全上下文" SAN 卷的要求。
  - sample input 目录包含示例，例如 `storage-class-basic.yaml.templ` 和 `storage-class-bronze-default.yaml`。
  - 有关详细信息，请参阅"已知问题"。

步骤 1：选择一个版本

Trident 版本遵循基于日期的 `YY.MM`命名约定，其中"YY"是年份的最后两位数字，"MM"是月份。点版本遵循 `YY.MM.X`约定，其中"X"是补丁级别。您将根据要升级的版本选择要升级到的版本。

- 您可以直接升级到安装版本的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 dot 版本）升级到 25.06。
- 如果您要从四发布窗口之外的版本进行升级，请执行多步骤升级。使用您要从"早期版本"升级的升级说明升级到适合四发布窗口的最新版本。例如，如果您运行的是 23.07 并希望升级到 25.06：
  a. 首次从 23.07 升级到 24.06。
  b. 然后从 24.06 升级到 25.06。

> 在 OpenShift Container Platform 上使用 Trident 操作员升级时，应升级到 Trident 21.01.1 或更高版本。随 21.01.0 发布的 Trident 操作员包含一个已在 21.01.1 中修复的已知问题。有关更多详细信息，请参阅 "在 GitHub 上的问题详细信息"。

步骤 2：确定原始安装方法

要确定您最初用于安装 Trident 的版本：

1. 使用 `kubectl get pods -n trident` 检查 pod。

- 如果没有操作员舱，则说明 Trident 是使用 `tridentctl` 安装的。
- 如果有 operator pod，Trident 是使用 Trident operator 手动或使用 Helm 安装的。

2. 如果有操作员 pod，请使用 `kubectl describe torc` 来确定是否使用 Helm 安装了 Trident。
   - 如果有 Helm 标签，则说明 Trident 是使用 Helm 安装的。
   - 如果没有 Helm 标签，则使用 Trident 操作员手动安装 Trident。

步骤 **3**：选择一种升级方法

一般来说，您应该使用与初始安装相同的方法进行升级，但您可以"在安装方法之间移动"。升级 Trident 有两种选择。

- "使用 Trident 操作员进行升级"

我们建议您在使用 operator 升级之前查看"了解 operator 升级工作流程"。

*

使用 operator 进行升级

了解 operator 升级工作流程

在使用 Trident 操作员升级 Trident 之前，您应该了解升级过程中发生的后台过程。这包括对 Trident 控制器、控制器 Pod 和节点 Pod 以及启用滚动更新的节点 DaemonSet 的更改。

Trident 操作员升级处理

安装和升级 Trident 的众多"使用 Trident 运算符的好处"之一是自动处理 Trident 和 Kubernetes 对象，而不会中断现有的已挂载卷。通过这种方式，Trident 可以支持零停机时间升级，或"滚动更新"。特别是，Trident 操作程序与 Kubernetes 集群进行通信以：

- 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。
  - 如果节点未更新，则不会阻止更新剩余节点。
  - 只有运行 Trident Node Pod 的节点才能挂载卷。

有关 Kubernetes 集群上的 Trident 架构的更多信息，请参见 "Trident 架构"。

Operator 升级工作流

使用 Trident 操作员启动升级时：

1. **Trident** 操作员：
   a. 检测当前安装的 Trident 版本（version *n*）。
   b. 更新所有 Kubernetes 对象，包括 CRD、RBAC 和 Trident SVC。

c. 删除版本 *n* 的 Trident Controller 部署。

d. 为版本 *n+1* 创建 Trident Controller 部署。

2. **Kubernetes** 为 *n+1* 创建 Trident Controller Pod。

3. **Trident** 操作员：

a. 删除 *n* 的 Trident 节点 DaemonSet。操作员不等待节点 Pod 终止。

b. 为 *n+1* 创建 Trident 节点守护进程集。

4. **Kubernetes** 在不运行 Trident Node Pod *n* 的节点上创建 Trident Node Pod。这可确保一个节点上不会有多于一个任何版本的 Trident Node Pod。

使用 **Trident** 操作员或 **Helm** 升级 **Trident** 安装

您可以使用 Trident 操作员手动或使用 Helm 升级 Trident。您可以从 Trident 操作员安装升级到另一个 Trident 操作员安装，或从 `tridentctl`安装升级到 Trident 操作员版本。在升级 Trident 操作员安装之前，请查看"选择升级方法"。

升级手动安装

您可以从集群范围内的 Trident 操作员安装升级到另一个集群范围内的 Trident 操作员安装。所有 Trident 版本都使用集群范围的操作符。

> 要从使用命名空间范围运算符（版本 20.07 至 20.10）安装的 Trident 进行升级，请使用 "您安装的版本" 版本的 Trident 升级说明。

关于此任务

Trident 提供了一个捆绑文件，可用于安装操作员并为您的 Kubernetes 版本创建关联的对象。

- 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

请确保使用的是正在运行 "受支持的 Kubernetes 版本" 的 Kubernetes 集群。

步骤

1. 验证您的 Trident 版本：

```
./tridentctl -n trident version
```

2. 使用要升级到的版本（例如 25.06）的注册表和图像路径以及正确的密码更新 `operator.yaml`、`tridentorchestrator_cr.yaml` 和 `post_1_25_bundle.yaml`。

3. 删除用于安装当前 Trident 实例的 Trident 操作员。例如，如果要从 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. 如果使用 `TridentOrchestrator` 属性自定义初始安装，则可以编辑 `TridentOrchestrator` 对象以修改安装参数。这可能包括为离线模式指定镜像的 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. 使用适合您环境的正确捆绑包 YAML 文件安装 Trident，其中 <*bundle.yaml*> 是 `bundle_pre_1_25.yaml`或 `bundle_post_1_25.yaml`，具体取决于您的 Kubernetes 版本。例如，如果要安装 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 Trident torc 以包括图像 25.06.0。

升级 **Helm** 安装

您可以升级 Trident Helm 安装。

> 在将安装了 Trident 的 Kubernetes 集群从 1.24 升级到 1.25 或更高版本时，必须更新 values.yaml 以将 `excludePodSecurityPolicy`设置为 `true`或将 `--set excludePodSecurityPolicy=true`添加到 `helm upgrade`命令中，然后才能升级集群。

如果您已经将 Kubernetes 集群从 1.24 升级到 1.25，而没有升级 Trident helm，则 helm 升级将失败。要完成 helm 升级，请执行以下步骤作为先决条件：

1. 从 https://github.com/helm/helm-mapkubeapis安装 helm-mapkubeapis 插件。
2. 在安装 Trident 的命名空间中对 Trident 版本执行试运行。这列出了将要清除的资源。

```
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. 如果您"使用 Helm 安装了 Trident"，您可以使用 `helm upgrade trident netapp-trident/trident-operator --version 100.2506.0`一步升级。如果您没有添加 Helm 存储库或无法使用它进行升级：
   a. 从 "GitHub 上的 *Assets* 部分" 下载最新的 Trident 版本。
   b. 请使用 `helm upgrade` 命令，其中 `trident-operator-25.10.0.tgz` 反映要升级到的版本。

```
helm upgrade <name> trident-operator-25.10.0.tgz
```

如果您在初始安装期间设置了自定义选项（例如为 Trident 和 CSI 镜像指定私有、镜像仓库），请使用 `helm upgrade`命令并附加 `--set`，以确保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. 运行 `helm list` 以确认图表和应用版本都已升级。运行 `tridentctl logs` 以查看所有调试消息。

从 `tridentctl` 安装升级到 **Trident** 操作员

您可以从 `tridentctl`安装升级到最新版本的 Trident 操作员。现有的后端和 PVC 将自动可用。

在切换安装方法之前，请查看 "在安装方法之间切换"。

步骤

1. 下载最新的 Trident 版本。

```
# Download the release required [25.10.0]
mkdir 25.10.0
cd 25.10.0
wget
https://github.com/NetApp/trident/releases/download/v25.10.0/trident-
installer-25.10.0.tar.gz
tar -xf trident-installer-25.10.0.tar.gz
cd trident-installer
```

2. 从清单中创建 `tridentorchestrator` 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作用域运算符。

```
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 `TridentOrchestrator` CR 以安装 Trident。

```
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:                v25.10.0
```

使用 tridentctl 升级

您可以使用 `tridentctl` 轻松升级现有 Trident 安装。

关于此任务

卸载和重新安装 Trident 相当于升级。卸载 Trident 时，不会删除 Trident 部署使用的 Persistent Volume Claim (PVC) 和 Persistent Volume (PV)。已配置的 PV 将在 Trident 脱机时保持可用，Trident 将在恢复联机后为在此期间创建的任何 PVC 配置卷。

开始之前

在使用 `tridentctl`进行升级之前，请先"选择升级方法"审核。

步骤

1. 在 `tridentctl`中运行卸载命令，以删除与 Trident 关联的所有资源，但 CRD 和相关对象除外。

```
./tridentctl uninstall -n <namespace>
```

2. 重新安装 Trident。请参见 "使用 tridentctl 安装 Trident"。

> 不要中断升级过程。确保安装程序运行完成。

使用 tridentctl 管理 Trident

`https://github.com/NetApp/trident/releases["Trident 安装包"^]` 包括 `tridentctl` 命令行实用程序，用于提供对 `Trident` 的简单访问。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

您可以运行 `tridentctl help`以获取 `tridentctl`的可用命令列表，或将 `--help`标志附加到任何命令以获取该特定命令的选项和标志列表。

```
tridentctl [command] [--optional-flag]
```

Trident `tridentctl` 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 中删除一个或多个资源。

get
从 Trident 获取一个或多个资源。

help
有关任何命令的帮助。

images
打印 Trident 需要的容器镜像表。

import
将现有资源导入到 Trident。

install
安装 Trident。

logs
打印 Trident 的日志。

send
从 Trident 发送资源。

uninstall
卸载 Trident。

update
修改 Trident 中的资源。

update backend state
暂时挂起后端操作。

upgrade
升级 Trident 中的资源。

version
打印 Trident 的版本。

全局标记

-d, --debug

调试输出。

-h, --help

`tridentctl` 的帮助。

-k, --kubeconfig string

指定在本地或从一个 Kubernetes 集群到另一个集群运行命令的 `KUBECONFIG`路径。

或者，您可以导出 `KUBECONFIG` 变量以指向特定的 Kubernetes 集群，并向该集群发出 `tridentctl` 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 之一（默认）。

-s, --server string

Trident REST 接口的地址/端口。

可以将 Trident REST 接口配置为仅在 127.0.0.1（用于 IPv4）或 [::1]（用于 IPv6）下侦听和服务。

命令选项和标志

create

使用 `create` 命令可将资源添加到 Trident。

```
tridentctl create [option]
```

选项

`backend`：向 Trident 添加后端。

删除

使用 `delete` 命令从 Trident 中删除一个或多个资源。

```
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。

获取

使用 `get` 命令从 Trident 获取一个或多个资源。

```
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 获取一个或多个卷。

标记

`-h, --help`: 卷的帮助。
`--parentOfSubordinate string`: 将查询限制为从属源卷。
`--subordinateOf string`: 将查询限制为卷的下属。

镜像

使用 `images`标志打印 Trident 需要的容器镜像表。

```
tridentctl images [flags]
```

标记

`-h, --help`: 图片帮助。
`-v, --k8s-version string`: Kubernetes 集群的语义版本。

导入卷

使用 `import volume` 命令可将现有卷导入 Trident。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

别名

`volume, v`

标记

`-f, --filename string`: YAML 或 JSON PVC 文件的路径。
`-h, --help`: 卷的帮助。
`--no-manage`: 仅创建 PV/PVC。不要假设卷生命周期管理。

安装

使用 `install` 标志安装 Trident。

```
tridentctl install [flags]
```

标记

`--autosupport-image string`：Autosupport Telemetry 的容器映像（默认为 "netapp/trident autosupport:<current-version>"）。
`--autosupport-proxy string`：用于发送 Autosupport Telemetry 的代理的地址/端口。
`--enable-node-prep`：尝试在节点上安装所需的程序包。
`--generate-custom-yaml`：在不安装任何内容的情况下生成 YAML 文件。
`-h， --help`：安装帮助。
`--http-request-timeout`：覆盖 Trident 控制器的 REST API 的 HTTP 请求超时（默认为 1m30s）。
`--image-registry string`：内部映像注册表的地址/端口。
`--k8s-timeout duration`：所有 Kubernetes 操作的超时（默认为 3m0s）。
`--kubelet-dir string`：kubelet 内部状态的主机位置（默认为 "/var/lib/kubelet"）。
`--log-format string`：Trident 日志记录格式（text、json）（默认为 "text"）。
`--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管理卷。当前，**`iscsi`** 是唯一受支持的值。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，确保其不存在（默认为 "trident"）。
`--pvc string`：Trident 使用的旧版 PVC 的名称，确保其不存在（默认为 "trident"）。
`--silence-autosupport`：不要自动将 autosupport 捆绑包发送到 NetApp（默认为 true）。
`--silent`：安装期间禁用大多数输出。
`--trident-image string`：要安装的 Trident 映像。
`--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`：使用安装目录中存在的任何现有 YAML 文件。
`--use-ipv6`：使用 IPv6 进行 Trident 通信。

logs

使用 `logs` 标志打印 Trident 的日志。

```
tridentctl logs [flags]
```

标记

`-a, --archive`: 除非另有指定，否则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 之一（默认为 "auto"）。
`--node string`: 从中收集节点 pod 日志的 Kubernetes 节点名称。
`-p, --previous`: 获取上一个容器实例的日志（如果存在）。
`--sidecars`: 获取 sidecar 容器的日志。

发送

使用 `send` 命令从 Trident 发送资源。

```
tridentctl send [option]
```

选项

`autosupport`：将 Autosupport 存档发送到 NetApp。

卸载

使用 `uninstall` 标志卸载 Trident。

```
tridentctl uninstall [flags]
```

标记

-h, --help：有关卸载的帮助。
--silent：在卸载期间禁用大多数输出。

更新

使用 update 命令可修改 Trident 中的资源。

```
tridentctl update [option]
```

选项

backend：更新 Trident 中的后端。

更新后端状态

使用 update backend state 命令可挂起或恢复后端操作。

```
tridentctl update backend state <backend-name> [flag]
```

需要考虑的要点

- 如果使用 TridentBackendConfig (tbc) 创建后端，则无法使用 backend.json 文件更新该后端。
- 如果 `userState`已在 tbc 中设置，则无法使用 `tridentctl update backend state <backend-name> --user-state suspended/normal`命令对其进行修改。
- 要在通过 tbc 设置 userState`后重新获得通过 tridentctl 设置的能力，必须从 tbc 中删除 `userState`字段。这可以使用 `kubectl edit tbc`命令来完成。删除 `userState`字段后，您可以使用 `tridentctl update backend state`命令来更改后端的 `userState。
- 使用 tridentctl update backend state`来更改 `userState。你也可以通过 TridentBackendConfig`或 `backend.json`文件来更新 `userState；这会触发后端的完全重新初始化，并且可能会耗费较多时间。

  标记

  -h, --help：后端状态的帮助。
  --user-state：设置为 suspended 以暂停后端操作。设置为 normal 以恢复后端操作。当设置为 suspended 时：

- AddVolume 和 Import Volume 已暂停。
- CloneVolume、 ResizeVolume、 PublishVolume、 UnPublishVolume、 CreateSnapshot、 GetSnapshot、 RestoreSnapshot、 DeleteSnapshot、 RemoveVolume、 GetVolumeExternal、 `ReconcileNodeAccess`仍然可用。

您还可以使用后端配置文件中的 userState`字段来更新后端状态 `TridentBackendConfig`或 `backend.json。有关更多信息，请参阅"管理后端的选项"和"使用 kubectl 执行后端管理"。

示例：

JSON

按照以下步骤使用 `userState` 文件更新 `backend.json`：

1. 编辑 `backend.json` 文件以包含值设置为 'suspended' 的 `userState` 字段。
2. 使用 `tridentctl update backend` 命令和更新的 `backend.json` 文件路径更新后端。

   示例: `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

您可以使用 `kubectl edit <tbc-name> -n <namespace>` 命令在应用 tbc 后对其进行编辑。以下示例使用 `userState: suspended` 选项将后端状态更新为挂起：

```
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` 标志打印 `tridentctl` 的版本和正在运行的 Trident 服务。

```
tridentctl version [flags]
```

标记

`--client`：仅客户端版本（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>"方案，并且二进制文件位于 PATH 环境变量列出的文件夹中，则 Tridentctl 会检测插件。所有检测到的插件都列在 tridentctl 帮助的插件部分中。或者，您也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（示例：`TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`）。如果使用该变量，tridentctl 仅在指定的文件夹中搜索。

监控 Trident

Trident 提供了一组 Prometheus 指标端点，可用于监控 Trident 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。您可以检查操作的成功程度，以及它是否可以按预期与后端进行通信。
- 检查后端使用情况信息并了解在后端上配置了多少卷以及占用的空间量等。
- 维护可用后端上配置的卷数量的映射。
- 跟踪性能。您可以查看 Trident 与后端通信并执行操作所需的时间。

默认情况下，Trident 的指标在目标端口 `8001`的 `/metrics`端点上公开。安装 Trident 时，这些指标*默认启用*。您也可以配置在端口 `8444`上通过 HTTPS 使用 Trident 指标。

您需要什么

- 已安装 Trident 的 Kubernetes 集群。
- Prometheus 实例。这可以是 "容器化 Prometheus 部署"，也可以选择将 Prometheus 作为 "本机应用程序" 运行。

步骤 1：定义 Prometheus 目标

您应该定义一个 Prometheus 目标，以收集指标并获取有关 Trident 管理的后端、其创建的卷等信息。请参阅 "Prometheus Operator 文档"。

步骤 2：创建 Prometheus ServiceMonitor

要使用 Trident 指标，您应该创建一个 Prometheus ServiceMonitor，用于监视 `trident-csi`服务并在 `metrics`端口上侦听。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 `trident-csi` 服务返回的指标，并特别查找服务的 `metrics` 端点。因此，Prometheus 现在被配置为理解 Trident 的指标。

除了可直接从 Trident 获得的指标外，kubelet 还通过自己的 `kubelet_volume_*`指标端点公开了许多指标。Kubelet 可以提供有关附加的卷及其处理的 Pod 和其他内部操作的信息。请参阅 "此处"。

通过 **HTTPS** 使用 **Trident** 指标

要通过 HTTPS（端口 8444）使用 Trident 指标，必须修改 ServiceMonitor 定义以包含 TLS 配置。您还需要将 `trident-csi` 密钥从 `trident` 命名空间复制到运行 Prometheus 的命名空间。您可以使用以下命令执行此操作：

```
kubectl get secret trident-csi -n trident -o yaml | sed 's/namespace:
trident/namespace: monitoring/' | kubectl apply -f -
```

HTTPS 指标的 ServiceMonitor 示例如下所示：

```yaml
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:
    - interval: 15s
      path: /metrics
      port: https-metrics
      scheme: https
      tlsConfig:
        ca:
          secret:
            key: caCert
            name: trident-csi
        cert:
          secret:
            key: clientCert
            name: trident-csi
        keySecret:
          key: clientKey
          name: trident-csi
        serverName: trident-csi
```

Trident 支持在所有安装方法中使用 HTTPS 指标：tridentctl、Helm chart 和 Operator：

- 如果使用 `tridentctl install` 命令，则可以传递 `--https-metrics` 标志以启用 HTTPS 指标。
- 如果使用 Helm 图表，可以设置 `httpsMetrics` 参数以启用 HTTPS 指标。
- 如果您正在使用 YAML 文件，您可以在 `trident-deployment.yaml` 文件中的 `trident-main` 容器添加 `--https_metrics` 标志。

步骤 3：使用 PromQL 查询 Trident 指标

PromQL 适用于创建返回时间序列或表格数据的表达式。

以下是可以使用的一些 PromQL 查询：

获取 **Trident** 运行状况信息

- 来自 **Trident** 的 **HTTP 2XX** 响应百分比

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- 通过状态代码从 **Trident** 获得的 **REST** 响应百分比

```
(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

- 平均卷大小

```
trident_volume_allocated_bytes/trident_volume_count
```

- 每个后端配置的总卷空间

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

获取单个卷使用量

只有当还收集了 kubelet 指标时才会启用此功能。

- 每个卷的已用空间百分比

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

了解 Trident AutoSupport 遥测

默认情况下，Trident 将 Prometheus 指标和基本后端信息按每日节奏发送到 NetApp。

- 要阻止 Trident 向 NetApp 发送 Prometheus 指标和基本后端信息，请在 Trident 安装期间传递 `--silence-autosupport` 标志。
- Trident 还可以通过 `tridentctl send autosupport` 按需将容器日志发送到 NetApp Support。您需要触发 Trident 上传其日志。在提交日志之前，您应接受 NetApp 的 "隐私政策"。
- 除非另有说明，否则 Trident 会从过去 24 小时内获取日志。
- 您可以使用 `--since`标志指定日志保留时间范围。例如： `tridentctl send autosupport --since=1h`。此信息通过与 Trident 一起安装的 `trident-autosupport`容器收集和发送。您可以在 "Trident AutoSupport"获取容器镜像。
- Trident AutoSupport 不会收集或传输个人身份信息 (PII) 或个人信息。它附带 "最终用户许可协议"，不适用于 Trident 容器图像本身。您可以了解有关 NetApp 对数据安全和信任承诺的更多信息 "此处"。

Trident 发送的示例负载如下所示：

```
---
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 消息将发送到 NetApp 的 AutoSupport 端点。如果要使用专用注册表存储容器图像，则可以使用 `--image-registry` 标志。
- 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`来创建 YAML 文件，并为 `trident-autosupport`容器在 `trident-deployment.yaml`中添加 `--proxy-url` 参数来实现。

禁用 Trident 指标

要禁用指标上报，您应生成自定义 YAML 文件（使用 `--generate-custom-yaml`参数），并编辑它们以移除 `--metrics`参数，使其不再被 `trident-main`容器调用。

卸载 Trident

您应该使用与安装 Trident 相同的方法来卸载 Trident。

关于此任务

- 如果您需要修复升级后发现的错误、依赖问题或未成功或未完成的升级，则应卸载 Trident 并使用特定说明重新安装早期版本"version"。这是_降级_到早期版本的唯一建议方法。
- 为了便于升级和重新安装，卸载 Trident 不会删除由 Trident 创建的 CRD 或相关对象。如果需要完全删除 Trident 及其所有数据，请参阅 "完全移除 Trident 和 CRD"。

开始之前

如果要停用 Kubernetes 集群，则必须在卸载之前删除所有使用 Trident 创建的卷的应用程序。这可确保 PVC 在删除之前未在 Kubernetes 节点上发布。

确定原始安装方法

您应该使用与安装 Trident 时相同的方法来卸载 Trident。在卸载之前，请验证您最初用于安装 Trident 的版本。

1. 使用 `kubectl get pods -n trident` 检查 pod。
   - 如果没有操作员舱，则说明 Trident 是使用 `tridentctl` 安装的。
   - 如果有 operator pod，Trident 是使用 Trident operator 手动或使用 Helm 安装的。
2. 如果有操作员 pod，请使用 `kubectl describe tproc trident` 来确定是否使用 Helm 安装了 Trident。
   - 如果有 Helm 标签，则说明 Trident 是使用 Helm 安装的。
   - 如果没有 Helm 标签，则使用 Trident 操作员手动安装 Trident。

卸载 Trident 操作员安装

您可以手动或使用 Helm 卸载 Trident 操作员安装。

卸载手动安装

如果您使用操作员安装了 Trident，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`标志设置为 `true`时，Trident 操作员卸载 Trident，但不删除 TridentOrchestrator 自身。如果要再次安装 Trident，您应该清理 TridentOrchestrator 并创建一个新的。

2. 删除 **`TridentOrchestrator`**：通过删除用于部署 Trident 的 `TridentOrchestrator` CR，您指示操作员卸载 Trident。操作员处理 `TridentOrchestrator`的删除并继续删除 Trident 部署和守护程序集，删除在安装过程中创建的 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 ，可以使用 `helm uninstall`将其卸载。

```
#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` 命令在 `tridentctl` 中删除与 Trident 关联的所有资源，但 CRD 和相关对象除外：

```
./tridentctl uninstall -n <namespace>
```

适用于 Docker 的 Trident

部署的前提条件

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。

验证要求

- 验证您的部署是否符合所有 "要求"。
- 验证您是否安装了受支持的 Docker 版本。如果您的 Docker 版本已过期， "安装或更新它"。

```
docker --version
```

- 验证协议先决条件是否已在主机上安装并配置。

NFS 工具

使用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 工具后重新启动工作节点，以防止将卷附加到容器时出现故障。

iSCSI 工具

使用操作系统命令安装 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. 检查 iscsi-initiator-utils 版本是否为 6.2.0.874-2.el7 或更高版本：

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

3. 将扫描设置为手动：

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

4. 启用多路径：

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

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

5. 确保 `iscsid` 和 `multipathd` 正在运行：

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

6. 启用并启动 `iscsi`：

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

Ubuntu

1. 安装以下系统软件包：

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

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

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

3. 将扫描设置为手动：

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

4. 启用多路径：

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

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

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

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

NVMe 工具

使用操作系统命令安装 NVMe 工具。

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

RHEL 9

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

Ubuntu

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

FC 工具

使用适用于您的操作系统的命令安装 FC 工具。

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

RHEL 8+

1. 安装以下系统软件包：

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

2. 启用多路径：

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

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

3. 请确保 `multipathd` 正在运行：

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

Ubuntu

1. 安装以下系统软件包：

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

2. 启用多路径：

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

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

3. 确保 `multipath-tools` 已启用并正在运行：

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

部署 Trident

Trident for Docker 为 NetApp 存储平台提供了与 Docker 生态系统的直接集成。它支持从存储平台到 Docker 主机的存储资源调配和管理，并具有将来添加其他平台的框架。

Trident 可以在同一主机上并发运行多个实例。这允许同时连接到多个存储系统和存储类型，并可以自定义用于 Docker 卷的存储。

您需要什么

请参见"部署的先决条件"。确保满足先决条件后，即可部署 Trident。

Docker 托管插件方法（版本 1.13/17.03 及更高版本）

开始之前

如果您在传统的守护进程方法中使用了 Docker 1.13/17.03 之前的 Trident，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. 停止所有运行的实例：

```
pkill /usr/local/bin/netappdvp
pkill /usr/local/bin/trident
```

2. 重新启动 Docker。

```
systemctl restart docker
```

3. 确保已安装 Docker Engine 17.03（新 1.13）或更高版本。

```
docker --version
```

如果您的版本已过期， "安装或更新您的安装"。

步骤

1. 创建配置文件并指定选项，如下所示：
   - `config`：默认文件名为 `config.json`，但是您可以通过指定 `config` 选项和文件名来使用您选择的任何名称。配置文件必须位于主机系统上的 `/etc/netappdvp` 目录中。
   - `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 true，则覆盖日志级别。
     i. 创建配置文件的位置：

```
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。将 `<version>` 替换为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 从已配置的系统中使用存储。

a. 创建名为"firstVolume"的卷：

```
docker volume create -d netapp --name firstVolume
```

b. 在容器启动时创建默认卷：

```
docker run --rm -it --volume-driver netapp --volume
secondVolume:/my_vol alpine ash
```

c. 删除卷 "firstVolume"：

```
docker volume rm firstVolume
```

传统方法（1.12 版或更早版本）

开始之前

1. 确保您拥有 Docker 1.10 或更高版本。

```
docker --version
```

如果您的版本已过期，请更新您的安装。

```
curl -fsSL https://get.docker.com/ | sh
```

或者 "按照您的发行版说明进行操作"。

2. 请确保为您的系统配置 NFS 和/或 iSCSI。

步骤

1. 安装并配置 NetApp Docker Volume 插件：

a. 下载并解压缩应用程序：

```
wget
https://github.com/NetApp/trident/releases/download/10.0/trident-
installer-25.10.0.tar.gz
tar zxf trident-installer-25.10.0.tar.gz
```

b. 移动到 bin 路径中的位置：

```
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. 创建配置文件的位置：

```
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 守护程序。

```
sudo trident --config=/etc/netappdvp/ontap-nas.json
```

除非指定，否则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. 启动容器时预配 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
```

在系统启动时启动 **Trident**

可以在 Git 存储库中的 `contrib/trident.service.example`找到基于 systemd 的系统的示例单元文件。要将文件与 RHEL 一起使用，请执行以下操作：

1. 将文件复制到正确的位置。

   如果有多个实例正在运行，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. 编辑文件，更改描述（第 2 行）以匹配驱动程序名称和配置文件路径（第 9 行）以反映您的环境。
3. 重新加载 systemd 以使其获取更改：

   ```
   systemctl daemon-reload
   ```

4. 启用服务。

   此名称因您在 `/usr/lib/systemd/system` 目录中为文件命名的名称而异。

   ```
   systemctl enable trident
   ```

5. 启动服务。

   ```
   systemctl start trident
   ```

6. 查看状态。

   ```
   systemctl status trident
   ```

> 每当您修改单元文件时，请运行 `systemctl daemon-reload` 命令以了解更改。

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。在升级过程中，会有一段短暂的时间，针对插件的 `docker volume`命令将不会成功，并且在插件再次运行之前，应用程序将无法挂载卷。在大多数情况下，这只是几秒钟的问题。

升级

执行以下步骤以升级 Docker 的 Trident。

步骤

1. 列出现有卷：

```
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。您应该直接从 `netapp/ndvp-plugin`镜像升级到 `netapp/trident-plugin`镜像。

4. 启用插件：

```
docker plugin enable netapp:latest
```

5. 验证插件是否已启用：

```
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       Trident - NetApp Docker Volume
Plugin   true
```

6. 验证卷是否可见：

```
docker volume ls
DRIVER              VOLUME NAME
netapp:latest       my_volume
```

如果要从旧版 Trident（20.10 之前）升级到 Trident 20.10 或更高版本，可能会遇到错误。有关详细信息，请参阅 "已知问题"。如果遇到错误，应先禁用插件，然后删除插件，然后通过传递额外的配置参数安装所需的 Trident 版本： `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

卸载

请执行以下步骤卸载 Docker 的 Trident。

步骤

1. 删除插件创建的任何卷。
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. 删除插件：

```
docker plugin rm netapp:latest
```

使用卷

您可以在需要时使用指定 Trident 驱动程序名称的标准 `docker volume`命令轻松创建、克隆和删除卷。

创建卷

- 使用默认名称的驱动程序创建卷：

```
docker volume create -d netapp --name firstVolume
```

- 使用特定 Trident 实例创建卷：

```
docker volume create -d ntap_bronze --name bronzeVolume
```

如果未指定任何 "options"，则使用驱动程序的默认值。

- 覆盖默认卷大小。请参见以下示例，使用驱动程序创建 20 GiB 的卷：

```
docker volume create -d netapp --name my_vol --opt size=20G
```

卷大小表示为包含整数值和可选单位的字符串（例如：10G、20GB、3TiB）。如果未指定单位，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）。速记单位使用 2 的幂（G = GiB、T = TiB、…）。

删除卷

- 像任何其他 Docker 卷一样删除该卷：

```
docker volume rm firstVolume
```

使用 `solidfire-san` 驱动程序时，上面的示例会删除并清除该卷。

执行以下步骤以升级 Docker 的 Trident。

克隆卷

使用 `ontap-nas`、 `ontap-san`和 `solidfire-san`存储驱动程序时，Trident 可以克隆卷。使用 `ontap-nas-flexgroup`或 `ontap-nas-economy`驱动程序时，不支持克隆。从现有卷创建新卷将导致创建新的快照。

- 检查卷以枚举快照：

```
docker volume inspect <volume_name>
```

- 从现有卷创建新卷。这将导致创建新的快照：

```
docker volume create -d <driver_name> --name <new_name> -o from
=<source_docker_volume>
```

- 从卷上的现有快照创建新卷。这不会创建新快照：

```
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

您可以通过使用 Trident 访问容器创建的外部块设备（或其克隆），*仅当*它们没有分区并且其文件系统受 Trident 支持时（例如： `ext4`格式的 `/dev/sdc1`将无法通过 Trident 访问）。

特定于驱动程序的卷选项

每个存储驱动程序都有一组不同的选项，您可以在卷创建时指定这些选项以自定义结果。请参阅下面适用于您配置的存储系统的选项。

在卷创建操作期间使用这些选项非常简单。在 CLI 操作期间使用 `-o`操作符提供选项和值。这些会覆盖 JSON 配置文件中的任何等效值。

ONTAP 卷选项

NFS、iSCSI 和 FC 的卷创建选项包括：

选项	说明
`size`	卷的大小，默认为 1 GiB。
`spaceReserve`	薄型或厚型配置卷，默认为薄型。有效值为 `none`（精简配置）和 `volume`（厚配置）。
`snapshotPolicy`	这将将快照策略设置为所需的值。默认值为 `none`，这意味着不会自动为卷创建快照。除非您的存储管理员修改，否则所有 ONTAP 系统上都存在名为"default"的策略，该策略会创建并保留六个每小时快照、两个每日快照和两个每周快照。可以通过浏览到卷中任何目录中的 `.snapshot`目录来恢复快照中保留的数据。
`snapshotReserve`	这将把快照预留设置为所需的百分比。默认情况下没有值，这意味着如果你选择了 snapshotPolicy ，ONTAP 会选择 snapshotReserve（通常为 5%）；如果 snapshotPolicy 为 none，则为 0%。你可以在配置文件中为所有 ONTAP 后端设置默认的 snapshotReserve 值，并且除了 ontap-nas-economy 以外，你可以将其作为所有 ONTAP 后端的卷创建选项。
`splitOnClone`	克隆卷时，这将导致 ONTAP 立即将克隆从其父级拆分出来。默认值为 `false`。克隆卷的某些用例最好在创建时立即将克隆从其父级拆分出来，因为不太可能有任何提高存储效率的机会。例如，克隆空数据库可以节省大量时间，但节省的存储空间很少，因此最好立即拆分克隆。
`encryption`	在新卷上启用 NetApp Volume Encryption (NVE)；默认值为 `false`。NVE 必须已在集群上获得许可并已启用，才能使用此选项。 如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。 有关详细信息，请参见："Trident 如何与 NVE 和 NAE 配合使用"。
`tieringPolicy`	设置要用于卷的分层策略。这决定了数据在变为非活动（冷）时是否移动到云层。

以下其他选项*仅适用于* NFS：

选项	说明
unixPermissions	这会控制卷本身的权限集。默认情况下，权限将设置为 `---rwxr-xr-x，或使用数字表示法 0755，root 将成为所有者。文本或数字格式都可以使用。
snapshotDir	将此设置为 true`将使 `.snapshot`目录对访问卷的客户端可见。默认值为 `false，这意味着默认情况下会禁用 `.snapshot`目录的可见性。某些镜像（例如官方 MySQL 镜像）在 `.snapshot`目录可见时无法按预期运行。
exportPolicy	设置要用于该卷的导出策略。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	说明
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	将此设置为 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。此选项还使 ONTAP 能够在主机删除数据时自动回收空间。

示例

请参见以下示例：

- 创建 10 GiB 卷：

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- 使用快照创建 100 GiB 卷：

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- 创建启用了 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，并且快照策略为 `none`，则 Trident 使用 0% 的快照保留。

- 创建没有快照策略和快照预留的卷：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- 创建没有快照策略且自定义快照保留为 10% 的卷：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
--opt snapshotReserve=10
```

- 创建具有快照策略和 10% 自定义快照预留的卷：

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy --opt snapshotReserve=10
```

- 使用快照策略创建卷，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 软件卷选项

Element 软件选项显示与卷关联的大小和服务质量 (QoS) 策略。创建卷时，与其关联的 QoS 策略使用 `-o type=service_level`命名法指定。

使用 Element 驱动程序定义 QoS 服务级别的第一步是至少创建一种类型，并指定与配置文件中的名称关联的最小、最大和突发 IOPS。

其他 Element 软件卷创建选项包括：

选项	说明
`size`	卷的大小，默认为 1 GiB 或配置项…"defaults": {"size": "5G"}。
`blocksize`	使用 512 或 4096，默认值为 512 或 config 条目 DefaultBlockSize。

示例

请参见以下具有 QoS 定义的配置文件示例：

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

在以上配置中，我们有三个策略定义：Bronze、Silver 和 Gold。这些名称是任意的。

- 创建 10 GiB Gold 卷：

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 创建 100 GiB Bronze 卷：

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

收集日志

您可以收集日志以获得故障排除方面的帮助。用于收集日志的方法因运行 Docker 插件的方式而异。

收集日志以进行故障排除

步骤

1. 如果使用推荐的托管插件方法（即使用 `docker plugin`命令）运行 Trident，请按如下方式查看：

```
docker plugin ls
```

```
ID                  NAME                DESCRIPTION
ENABLED
4fb97d2b956b        netapp:latest       nDVP - NetApp Docker Volume
Plugin    false
journalctl -u docker | grep 4fb97d2b956b
```

   标准日志记录级别应允许您诊断大多数问题。如果您发现这还不够，则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的插件：

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

   或者，在已安装插件时启用调试日志记录：

```
docker plugin disable <plugin>
```

```
docker plugin set <plugin> debug=true
```

```
docker plugin enable <plugin>
```

3. 如果您在主机上运行二进制文件本身，则可在主机的 `/var/log/netappdvp` 目录中找到日志。要启用调试日志记录，请在运行插件时指定 `-debug`。

常规故障排除提示

- 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。当发生这种情况时，您尝试安装或启用插件时可能会看到这样的消息：

```
Error response from daemon: dial unix /run/docker/plugins/<id>/netapp.sock:
connect: no such file or directory
```

这意味着插件无法启动。幸运的是，该插件具有全面的日志记录功能，可以帮助您诊断可能遇到的大多数问题。

- 如果在将 PV 安装到容器时出现问题，请确保 `rpcbind`已安装并运行。使用主机操作系统所需的包管理器，并检查 `rpcbind`是否正在运行。可以通过运行 `systemctl status rpcbind`或其等效项来检查 rpcbind 服务的状态。

管理多个 Trident 实例

当您希望同时提供多个存储配置时，需要多个 Trident 实例。多个实例的关键是使用带有容器化插件的 `--alias`选项或在主机上实例化 Trident 时使用 `--volume-driver`选项为它们提供不同的名称。

Docker 托管插件步骤（版本 1.13/17.03 或更高版本）

1. 启动指定别名和配置文件的第一个实例。

```
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. 创建将别名指定为驱动程序名称的卷。

   例如，对于黄金卷：

```
docker volume create -d gold --name ntapGold
```

   例如，对于银卷：

```
docker volume create -d silver --name ntapSilver
```

传统（1.12 版或更早版本）的步骤

1. 使用自定义驱动程序 ID 启动具有 NFS 配置的插件：

```
sudo trident --volume-driver=netapp-nas --config=/path/to/config
-nfs.json
```

2. 使用自定义驱动程序 ID 通过 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	说明	示例
`version`	配置文件版本号	`1`
`storageDriverName`	存储驱动程序的名称	`ontap-nas`, `ontap-san`, `ontap-nas-economy`, `ontap-nas-flexgroup`, `solidfire-san`
`storagePrefix`	卷名的可选前缀。默认值：`netappdvp_`。	`staging_`
`limitVolumeSize`	对卷大小的可选限制。默认值：" （未强制执行）	`10g`

不要对 Element 后端使用 `storagePrefix`（包括默认值）。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 tenantID 进行 Docker 卷映射，或者在可能使用任何名称 munging 的情况下使用 Docker 版本、驱动程序信息和 Docker 原始名称填充的属性数据。

默认选项可用于避免在创建的每个卷上指定它们。该 `size`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	说明	示例
`size`	新卷的可选默认大小。默认值： `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	说明	示例
`managementLIF`	ONTAP 管理 LIF 的 IP 地址。您可以指定完全限定的域名 (FQDN)。	`10.0.0.1`
`dataLIF`	协议 LIF 的 IP 地址。 **ONTAP NAS** 驱动程序：NetApp 建议指定 `dataLIF`。如果未提供，Trident 从 SVM 获取 dataLIF。您可以指定要用于 NFS 挂载操作的完全限定域名 (FQDN)，允许您创建循环 DNS 以跨多个 dataLIF 进行负载平衡。 **ONTAP SAN** 驱动程序：不为 iSCSI 或 FC 指定。Trident 使用 "ONTAP 选择性 LUN 映射" 来发现建立多路径会话所需的 iSCSI 或 FC LIF。如果明确定义了 `dataLIF`，则会生成警告。	`10.0.0.2`
`svm`	要使用的 Storage Virtual Machine（如果 Management LIF 是集群 LIF，则必需）	`svm_nfs`
`username`	连接到存储设备的用户名	`vsadmin`
`password`	连接到存储设备的密码	`secret`

选项	说明	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，如果使用率高于此百分比，则配置失败	`75%`
`nfsMountOptions`	NFS 挂载选项的细粒度控制；默认为 "-o nfsvers=3"。仅适用于 **`ontap-nas`**和 **`ontap-nas-economy`**驱动程序 "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的 `igroups`作为 `netappdvp`。 此值不能更改或省略。 仅适用于 **`ontap-san`** 驱动程序。	`netappdvp`
`limitVolumeSize`	可请求的最大卷大小。	`300g`
`qtreesPerFlexvol`	每个 FlexVol 的最大 qtrees，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。	`300`
`sanType`	*仅支持 `ontap-san` 驱动程序。*用于为 iSCSI 选择 `iscsi`，为 NVMe/TCP 选择 `nvme`，或为光纤通道 (FC) 上的 SCSI 选择 `fcp`。	`iscsi` 如果为空
`limitVolumePoolSize`	*仅支持 `ontap-san-economy`和 `ontap-san-economy`驱动程序。*限制 ONTAP ontap-nas-economy 和 ontap-SAN-economy 驱动程序中的 FlexVol 大小。	`300g`

默认选项可用于避免在创建的每个卷上指定它们：

选项	说明	示例
spaceReserve	空间预留模式； none （精简配置）或 volume （厚配置）	none
snapshotPolicy	要使用的 Snapshot 策略，默认为 none	none
snapshotReserve	Snapshot 预留百分比，默认为 "" 以接受 ONTAP 默认值	10
splitOnClone	创建时从其父级拆分克隆，默认为 false	false
encryption	在新卷上启用 NetApp Volume Encryption (NVE)；默认为 false。NVE 必须已在集群上获得许可并已启用，才能使用此选项。 如果在后端启用了 NAE，则在 Trident 中配置的任何卷都将启用 NAE。 有关详细信息，请参见："Trident 如何与 NVE 和 NAE 配合使用"。	true
unixPermissions	调配的 NFS 卷的 NAS 选项，默认为 777	777
snapshotDir	用于访问 `.snapshot`目录的 NAS 选项。	NFSv4 为 "true"，NFSv3 为 "false"
exportPolicy	要使用的 NFS 导出策略的 NAS 选项，默认为 default	default
securityStyle	用于访问配置的 NFS 卷的 NAS 选项。 NFS 支持 mixed`和 `unix`安全样式。默认值为 `unix。	unix
fileSystemType	选择文件系统类型的 SAN 选项，默认为 ext4	xfs
tieringPolicy	要使用的分层策略，默认值为 none。	none
skipRecoveryQueue	在卷删除过程中，绕过存储中的恢复队列并立即删除卷。	``

扩展选项

`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 FlexVols，集群最大容量为 12,000 个 FlexVol 卷。如果您的 Docker 卷要求符合该限制，则 `ontap-nas` 驱动程序是首选的 NAS 解决方案，因为 FlexVols 提供了额外的功能，例如 Docker 卷粒度快照和克隆。

如果您需要超出 FlexVol 限制的 Docker 卷，请选择 ontap-nas-economy 或 ontap-san-economy 驱动程序。

`ontap-nas-economy` 驱动程序在自动管理的 FlexVol 卷池中将 Docker 卷创建为 ONTAP Qtree。Qtree 提供了更大的扩展性，每个集群节点高达 100,000 个，每个集群高达 2,400,000 个，但牺牲了一些功能。 `ontap-nas-economy` 驱动程序不支持 Docker-volume-granular 快照或克隆。

Docker Swarm 当前不支持该 `ontap-nas-economy`驱动程序，因为 Docker Swarm 不跨多个节点编排卷创建。

`ontap-san-economy` 驱动程序在自动管理的 FlexVol 卷的共享池中将 Docker 卷创建为 ONTAP LUN。这样，每个 FlexVol 就不仅限于一个 LUN，还可以为 SAN 工作负载提供更好的可扩展性。根据存储阵列，ONTAP 支持每个集群最多 16384 个 LUN。因为卷是底层的 LUN，所以此驱动程序支持 Docker-volume-granular 快照和克隆。

选择 ontap-nas-flexgroup 驱动程序，以提升对单一卷的并行性，该卷可扩展到拥有数十亿文件的 PB 级别。FlexGroups 的一些理想用例包括 AI/ML/DL、大数据与分析、软件构建、流媒体、文件存储库等。Trident 在配置 FlexGroup 卷时，会使用分配给 SVM 的所有聚合体。Trident 中对 FlexGroup 的支持还需注意以下事项：

- 需要 ONTAP 版本 9.2 或更高版本。
- 截至本文撰写时，FlexGroups 仅支持 NFS v3。
- 建议为 SVM 启用 64 位 NFSv3 标识符。
- 最小推荐 FlexGroup 成员/卷大小为 100 GiB。
- FlexGroup 卷不支持克隆。

有关 FlexGroups 和适用于 FlexGroups 的工作负载的信息，请参见 "NetApp FlexGroup 卷最佳实践和实施指南"。

要在同一环境中获得高级功能和大规模扩展，您可以运行 Docker Volume Plugin 的多个实例，一个使用 ontap-nas，另一个使用 ontap-nas-economy。

Trident 的自定义 ONTAP 角色

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

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

使用 **ONTAP CLI**

1. 使用以下命令创建新角色：

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

2. 为 Trident 用户创建用户名：

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

在 ONTAP System Manager 中执行以下步骤：

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

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

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

有关详细信息，请参见以下页面：

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

ONTAP 配置文件示例

ONTAP-nas<code> 驱动程序的 NFS</code> 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"
  }
}
```

ontap-nas-flexgroup<code></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
}
```

用于 **<code>ontap-san</code> 驱动程序的基于 FC 的 SCSI 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，使用 Element 软件（NetApp HCI/SolidFire）时，还可以使用这些选项。

选项	说明	示例
`Endpoint`	https://<login>:<password>@<mvip>/json-rpc/<element-version>	https://admin:admin@192.168.160.3/json-rpc/8.0
`SVIP`	iSCSI IP 地址和端口	10.0.0.7:3260
`TenantName`	要使用的 SolidFire 租户（如果未找到，则创建）	`docker`

选项	说明	示例
`InitiatorIFace`	在将 iSCSI 流量限制为非默认接口时指定接口	`default`
`Types`	QoS 规范	请参见以下示例
`LegacyNamePrefix`	升级版 Trident 安装的前缀。如果您使用了 1.3.2 之前的 Trident 版本并对现有卷执行升级，则需要设置此值以访问通过卷名称方法映射的旧卷。	`netappdvp-`

此 `solidfire-san`驱动程序不支持 Docker Swarm。

示例 **Element** 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

查找有关在 Docker 中使用 Trident 的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，并显示不存在此文件或目录错误。

临时解决策

1. 禁用插件。

```
docker plugin disable -f netapp:latest
```

2. 删除插件。

```
docker plugin rm -f netapp:latest
```

3. 通过提供额外 `config`参数重新安装插件。

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

卷名长度必须至少为 **2** 个字符。

这是 Docker 客户端限制。客户端会将单个字符名称解释为 Windows 路径 "请参见错误 25773"。

Docker Swarm 的某些行为会阻止 **Trident** 支持每种存储和驱动程序组合。

- Docker Swarm 目前使用卷名而不是卷 ID 作为其唯一的卷标识符。
- 卷请求同时发送到 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`驱动程序的功能，它们是唯一能够在这些限制范围内运行的驱动程序。

其余的驱动程序会受到诸如竞争条件之类的问题的影响，这些问题可能会导致为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不同 ID 的功能。

NetApp 已经向 Docker 团队提供了反馈，但没有任何未来追索的迹象。

如果正在调配 **FlexGroup**，则当第二个 **FlexGroup** 与正在调配的 **FlexGroup** 具有一个或多个相同聚合时，**ONTAP** 不会调配第二个 **FlexGroup**。

最佳实践和建议

部署

部署 Trident 时，请使用此处列出的建议。

部署到专用命名空间

"命名空间" 提供不同应用程序之间的管理隔离，并且是资源共享的障碍。例如，来自一个命名空间的 PVC 不能从另一个命名空间使用。Trident 为 Kubernetes 集群中的所有命名空间提供 PV 资源，从而利用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。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 本身非常重要。

使用配额和范围限制来控制存储消耗

Kubernetes 有两个特性，当它们结合在一起时，提供了一个强大的机制来限制应用程序的资源消耗。 "存储配额机制"使管理员能够在每个命名空间的基础上实现全局和存储类特定的容量和对象计数消耗限制。此外，使用 "范围限制"确保在将请求转发给供应者之前，PVC 请求处于最小值和最大值之间。

这些值是在每个命名空间的基础上定义的，这意味着每个命名空间都应该定义符合其资源要求的值。有关 "如何利用配额"的信息，请参阅此处。

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，使应用程序（无论是否是容器化的）受益。

平台概述

Trident 与 ONTAP 和 Element 配合使用。没有一个平台比另一个平台更适合所有应用程序和场景，但是，在选择平台时应考虑应用程序和管理设备的团队的需求。

您应该遵循主机操作系统的基本最佳实践，以及您正在利用的协议。或者，您可能需要考虑将应用程序最佳实践（如果可用）与后端、存储类和 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

了解为 Trident 配置 ONTAP 和 Cloud Volumes ONTAP 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。应考虑并评估每种方法是否适合您的环境。

使用专用于 **Trident** 的 **SVM**

Storage Virtual Machine （ SVM ）在 ONTAP 系统上的租户之间提供隔离和管理隔离。将 SVM 专用于应用程序可以实现权限委派，并能够应用限制资源消耗的最佳实践。

有多种选择可用于管理 SVM：

- 在后端配置中提供集群管理界面，以及相应的凭据，并指定 SVM 名称。
- 使用 ONTAP System Manager 或 CLI 为 SVM 创建专用管理界面。
- 通过 NFS 数据接口共享管理角色。

在每种情况下，接口都应在 DNS 中，并且在配置 Trident 时应使用 DNS 名称。这有助于促进某些灾难恢复场景，例如，不使用网络身份保留的 SVM-DR。

对于 SVM，没有专用或共享管理 LIF 的偏好，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 应可通过 DNS 访问，以实现最大的灵活性，如果 "SVM-DR"与 Trident 结合使用。

限制最大卷计数

ONTAP 存储系统具有最大卷计数，该计数因软件版本和硬件平台而异。请参阅 "NetApp Hardware Universe"以确定您的特定平台和 ONTAP 版本的确切限制。当卷计数耗尽时，不仅 Trident 的配置操作会失败，所有存储请求的配置操作都会失败。

Trident 的 `ontap-nas`和 `ontap-san`驱动程序为创建的每个 Kubernetes 持久卷 (PV) 配置一个 FlexVolume。`ontap-nas-economy`驱动程序大约每 200 个 PV 创建一个 FlexVolume（可配置在 50 到 300 之间）。 `ontap-san-economy`驱动程序大约每 100 个 PV 创建一个 FlexVolume（可配置在 50 到 200 之间）。要防止 Trident 消耗存储系统上的所有可用卷，您应该在 SVM 上设置限制。您可以从命令行执行此操作：

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

`max-volumes` 的值根据您环境的几个特定标准而有所不同：

- ONTAP 集群中的现有卷数
- 您希望在 Trident 之外为其他应用程序配置的卷数量
- Kubernetes 应用程序预计消耗的持久卷数

该 `max-volumes`值是在 ONTAP 集群中的所有节点上配置的总卷，而不是在单个 ONTAP 节点上配置的总卷。因此，您可能会遇到某些情况，其中 ONTAP 集群节点可能比其他节点具有更多或更少 Trident 配置卷。

例如，双节点 ONTAP 集群最多可以托管 2000 个 FlexVol 卷。将最大卷计数设置为 1250 似乎非常合理。但是，如果仅 "聚合"从一个节点分配给 SVM，或者无法对从一个节点分配的聚合进行配置（例如，由于容量），则另一个节点将成为所有 Trident 配置卷的目标。这意味着在达到 `max-volumes`值之前，可能会达到该节点的卷限制，从而影响 Trident 和使用该节点的其他卷操作。您可以通过确保将集群中每个节点的聚合以相等数量分配给 **Trident** 使用的 **SVM** 来避免这种情况。

克隆卷

NetApp Trident 在使用 `ontap-nas`、 `ontap-san` 和 `solidfire-san` 存储驱动程序时支持克隆卷。使用 `ontap-nas-flexgroup` 或 `ontap-nas-economy` 驱动程序时，不支持克隆。从现有卷创建新卷将导致创建新的快照。

避免克隆与不同 StorageClass 关联的 PVC。在相同的 StorageClass 内执行克隆操作，以确保兼容性并防止意外行为。

限制 **Trident** 创建的卷的最大大小

要配置 Trident 可创建的卷的最大大小，请在您的 `limitVolumeSize`定义中使用 `backend.json`参数。

除了控制存储阵列的卷大小之外，还应利用 Kubernetes 功能。

限制 **Trident** 创建的 **FlexVols** 的最大大小

要为 ontap-san-economy 和 ontap-nas-economy 驱动程序用作池的 FlexVols 配置最大大小，请在 `limitVolumePoolSize` 参数中，在 `backend.json` 定义中使用。

配置 **Trident** 使用双向 **CHAP**

您可以在后端定义中指定 CHAP 启动程序和目标用户名和密码，并让 Trident 在 SVM 上启用 CHAP。使用后端配置中的 `useCHAP`参数，Trident 使用 CHAP 对 ONTAP 后端的 iSCSI 连接进行身份验证。

创建和使用 **SVM QoS** 策略

利用应用于 SVM 的 ONTAP QoS 策略，可限制 Trident 配置卷可消耗的 IOPS 数量。这有助于 "防止霸凌"或失控容器影响 Trident SVM 外部的工作负载。

您可以通过几个步骤为 SVM 创建 QoS 策略。有关最准确的信息，请参阅您的 ONTAP 版本文档。以下示例创建了一个 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 版本支持它，则可以考虑使用 QoS 最小值来保证容器化工作负载的吞吐量。自适应 QoS 与 SVM 级别策略不兼容。

专用于容器化工作负载的 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。如果有其他与 Kubernetes 部署无关的工作负载利用存储资源，则应注意确保这些工作负载不会受到意外不利影响。
- 在容器中运行的预期工作负载。如果具有高 IOPS 要求的工作负载将在容器中运行，则低 QoS 策略会导致不良体验。

重要的是要记住，在 SVM 级别分配的 QoS 策略会导致调配到 SVM 的所有卷共享相同的 IOPS 池。如果一个或少数容器化应用程序具有较高的 IOPS 要求，它可能会成为其他容器化工作负载的欺凌者。如果是这种情况，您可能需要考虑使用外部自动化来分配每个卷的 QoS 策略。

仅当您的 ONTAP 版本早于 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，并使用户能够为一个或多个工作负载定义吞吐量上限。有关 QoS 的详细信息，请参阅 "通过 QoS 保证吞吐量"。您可以在后端或存储池中指定 QoS 策略组，并将其应用于在该池或后端中创建的每个卷。

ONTAP 有两种 QoS 策略组：传统和自适应。传统策略组在 IOPS 中提供固定的最大（或更高版本中的最小值）吞吐量。自适应 QoS 自动将吞吐量扩展到工作负载大小，随着工作负载大小的变化而保持 IOPS 与 TBs|GBs 的比例。当您在大型部署中管理数百或数千个工作负载时，这提供了显著的优势。

创建 QoS 策略组时，请考虑以下事项：

- 您应在后端配置的 `defaults` 块中设置 `qosPolicy` 键。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，以便每个卷获得策略组指定的整个吞吐量。不支持共享策略组。

有关 QoS 策略组的详细信息，请参阅 "ONTAP 命令参考"。

限制存储资源访问 **Kubernetes** 集群成员

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。

请务必了解，命名空间是 Kubernetes 中资源的逻辑边界。假设相同命名空间中的资源可以共享，但重要的是，没有跨命名空间功能。这意味着，即使 PV 是全局对象，当绑定到 PVC 时，它们只能由位于同一命名空间中的 pod 访问。确保在适当的时候使用命名空间来提供分隔至关重要。

大多数组织在 Kubernetes 环境中对数据安全性的主要关注点是，容器中的进程可以访问挂载到主机的存储，但这些存储并非为该容器准备的。 "命名空间" 旨在防止这种类型的妥协。但是，有一个例外：特权容器。

特权容器是使用比正常情况下更多的主机级权限运行的容器。默认情况下，这些功能不会被拒绝，因此请确保使用 "pod 安全策略" 禁用此功能。

对于需要从 Kubernetes 和外部主机进行访问的卷，应以传统方式管理存储，PV 由管理员引入，而不是由 Trident 管理。这确保了只有当 Kubernetes 和外部主机都断开连接并且不再使用卷时，存储卷才会被销毁。此外，还可以应用自定义导出策略，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）的部署或其他无法安排用户应用程序的节点，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。

使用专用导出策略

您应该确保每个后端都存在一个导出策略，该策略仅允许访问 Kubernetes 集群中存在的节点。Trident 可以自动创建和管理导出策略。通过这种方式，Trident 将其配置的卷的访问限制为 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，并为其填充一个或多个处理每个节点访问请求的导出规则：

- 使用 `vserver export-policy create` ONTAP CLI 命令创建导出策略。
- 使用 `vserver export-policy rule create` ONTAP CLI 命令将规则添加到导出策略。

运行这些命令可以限制哪些 Kubernetes 节点可以访问这些数据。

为应用程序 **SVM** 禁用 `showmount`

该 `showmount`功能使 NFS 客户端能够查询 SVM 以获取可用 NFS 导出的列表。部署到 Kubernetes 集群的 pod 可以对发出 `showmount -e`命令，并接收可用挂载的列表，包括它无权访问的挂载。虽然这本身并不是安全威胁，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。

您应该使用 SVM 级 ONTAP CLI 命令禁用 `showmount`：

```
vserver nfs modify -vserver <svm_name> -showmount disabled
```

SolidFire 最佳实践

了解为 Trident 配置 SolidFire 存储的最佳实践。

创建 **SolidFire** 账户

每个 SolidFire 帐户代表一个唯一的卷所有者，并接收其自己的一组 Challenge-Handshake Authentication Protocol (CHAP) 凭据。您可以使用帐户名和相关 CHAP 凭据或通过卷访问组访问分配给帐户的卷。一个帐户最多可以分配两千个卷，但一个卷只能属于一个帐户。

创建 **QoS** 策略

如果要创建和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。可以通过设置定义 QoS 的三个可配置参数来确保每个卷的性能：最小 IOPS、最大 IOPS 和突发 IOPS。

下面是 4Kb 块大小可能的最小、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	卷的保证性能级别。	50	50	15000
最大 IOPS	性能不会超过此限制。	50	15000	200,000
突发 IOPS	短时间突发场景中允许的最大 IOPS。	50	15000	200,000

虽然最大 IOPS 和突发 IOPS 可以设置为高达 200,000，但卷的实际最大性能受到群集使用率和每个节点性能的限制。

块大小和带宽对 IOPS 数量有直接影响。随着块大小的增加，系统将带宽增加到处理较大块大小所需的级别。随着带宽的增加，系统能够实现的 IOPS 数量会减少。有关 QoS 和性能的更多信息，请参阅 "SolidFire 服务质量"。

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 协议向后端验证主机。卷访问组控制对其设置的卷的访问。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。

具有增强 CSI 配置程序的 Trident 支持使用 CHAP 身份验证。VAG 只能在传统的非 CSI 操作模式中使用。

只有基于帐户的访问控制才支持 CHAP 身份验证（验证启动器是预期的卷用户）。如果使用 CHAP 进行身份验证，则有两个选项可用：单向 CHAP 和双向 CHAP。单向 CHAP 使用 SolidFire 帐户名和启动器密码对卷访问进行身份验证。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启动器密码对主机进行身份验证，然后主机通过帐户名和目标密码对卷进行身份验证。

但是，如果无法启用 CHAP 并且需要 VAG，请创建访问组并将主机启动程序和卷添加到访问组。添加到访问组中的每个 IQN 都可以使用 CHAP 身份验证或不使用 CHAP 身份验证访问组中的每个卷。如果将 iSCSI 启动程序配置为使用 CHAP 身份验证，则使用基于帐户的访问控制。如果未将 iSCSI 启动程序配置为使用 CHAP 身份验证，则使用卷访问组访问控制。

在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"以查找最新版本。

ONTAP

- "NFS 最佳实践和实施指南"
- "SAN 管理" (用于 iSCSI)
- "适用于 RHEL 的 iSCSI Express 配置"

Element 软件

- "为 Linux 配置 SolidFire"

NetApp HCI

- "NetApp HCI 部署先决条件"
- "访问 NetApp 部署引擎"

应用程序最佳实践信息

- "ONTAP 上 MySQL 的最佳实践"
- "MySQL 在 SolidFire 上的最佳实践"
- "NetApp SolidFire 和 Cassandra"
- "Oracle 在 SolidFire 上的最佳实践"
- "PostgreSQL 在 SolidFire 上的最佳实践"

并非所有应用程序都有特定的指导方针，与您的 NetApp 团队合作并使用 "NetApp 库"查找最新文档非常重要。

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 的 OpenShift 服务部署。

驱动程序选择和部署

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 后端驱动程序根据使用的协议以及如何在存储系统上调配卷而有所不同。因此，在决定部署哪个驱动程序时，请仔细考虑。

在更高的级别上，如果您的应用程序具有需要共享存储的组件（多个 pod 访问相同的 PVC），则基于 NAS 的驱动程序将成为默认选择，而基于块的 iSCSI 驱动程序将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舒适度来选择协议。一般来说，对于大多数应用程序，它们之间几乎没有区别，因此决定通常取决于是否需要共享存储（其中需要同时访问多个 pod）。

可用的 ONTAP 后端驱动程序包括：

- `ontap-nas`: 配置的每个 PV 都是完整的 ONTAP FlexVolume。

- `ontap-nas-economy`：配置的每个 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 配置为完整的 ONTAP FlexGroup，并使用分配给 SVM 的所有聚合。
- `ontap-san`：配置的每个 PV 都是其自身 FlexVolume 内的 LUN。
- `ontap-san-economy`：调配的每个 PV 都是一个 LUN，每个 FlexVolume 具有可配置的 LUN 数（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提供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。如果需要该功能，则存储管理员必须在配置后应用某些功能。上标脚注区分了每个功能和驱动程序的功能。

ONTAP NAS 驱动程序	**Snapshot**	克隆	动态导出策略	多重连接	**QoS**	调整大小	复制
`ontap-nas`	是	是	是 [5]	是	是脚注:1[]	是	是脚注:1[]
`ontap-nas-economy`	NO [3]	NO [3]	是 [5]	是	NO [3]	是	NO [3]
`ontap-nas-flexgroup`	是脚注:1[]	否	是 [5]	是	是脚注:1[]	是	是脚注:1[]

Trident 为 ONTAP 提供 2 个 SAN 驱动程序，其功能如下所示。

ONTAP SAN 驱动程序	**Snapshot**	克隆	多重连接	双向 **CHAP**	**QoS**	调整大小	复制
`ontap-san`	是	是	是脚注:4[]	是	是脚注:1[]	是	是脚注:1[]
`ontap-san-economy`	是	是	是脚注:4[]	是	NO [3]	是	NO [3]

> 以上表格的脚注：是脚注：1[]：不由 Trident 管理 是脚注：2[]：由 Trident 管理，但不是 PV 粒度 否脚注：3[]：不由 Trident 管理且不是 PV 粒度 是脚注：4[]：支持原始块卷 是脚注：5[]：由 Trident 支持

非 PV 粒度的功能将应用于整个 FlexVolume，并且所有 PV（即共享 FlexVols 中的 qtree 或 LUN）都将共享一个共同的时间表。

如上表所示， `ontap-nas` 和 `ontap-nas-economy` 之间的大部分功能是相同的。但是，由于 `ontap-nas-economy` 驱动程序限制了按 PV 粒度控制计划的能力，这尤其会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，只有在使用 `ontap-nas`、 `ontap-san` 或 `ontap-san-economy` 驱动程序时才有可能。

> `solidfire-san` 驱动程序还能够克隆 PVC。

Cloud Volumes ONTAP 后端驱动程序

Cloud Volumes ONTAP 为各种用例提供数据控制以及企业级存储功能，包括为 NAS 和 SAN 协议（NFS 、SMB/CIFS 和 iSCSI）提供服务的文件共享和块级存储。Cloud Volume ONTAP 的兼容驱动程序为 `ontap-nas`、 `ontap-nas-economy`、 `ontap-san`和 `ontap-san-economy`。这些适用于 Azure 的 Cloud Volume ONTAP、GCP 的 Cloud Volume ONTAP。

Amazon FSx for ONTAP 后端驱动程序

Amazon FSx for NetApp ONTAP 可让您利用熟悉的 NetApp 功能、性能和管理功能，同时利用在 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 的兼容驱动程序是 `ontap-nas`、 `ontap-nas-economy`、 `ontap-nas-flexgroup`、 `ontap-san` 和 `ontap-san-economy`。

NetApp HCI/SolidFire 后端驱动程序

与 NetApp HCI/SolidFire 平台一起使用的 `solidfire-san`驱动程序可帮助管理员根据 QoS 限制为 Trident 配置 Element 后端。如果您想设计后端以在 Trident 配置的卷上设置特定的 QoS 限制，请使用后端文件中的 `type`参数。管理员还可以使用 `limitVolumeSize`参数限制可以在存储上创建的卷大小。目前， `solidfire-san`驱动程序不支持调整卷大小和卷复制等 Element 存储功能。这些操作应通过 Element Software Web UI 手动完成。

SolidFire 驱动程序	Snapshot	克隆	多重连接	CHAP	QoS	调整大小	复制
`solidfire-san`	是	是	是脚注:2[]	是	是	是	是脚注:1[]

> 脚注：是脚注：1[]：不由 Trident 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` 驱动程序来管理 "Azure NetApp Files" 服务。

有关此驱动程序及其配置方法的详细信息，请参见 "Azure NetApp Files 的 Trident 后端配置"。

Azure NetApp Files 驱动程序	Snapshot	克隆	多重连接	QoS	展开	复制
`azure-netapp-files`	是	是	是	是	是	是脚注:1[]

> 脚注：是脚注：1[]：不由 Trident 管理

存储类设计

需要配置和应用单个存储类来创建 Kubernetes Storage Class 对象。本节讨论如何为应用程序设计存储类。

特定后端利用率

筛选可以在特定存储类对象内使用，以确定要与该特定存储类一起使用的存储池或池集。可以在存储类中设置三组过滤器： `storagePools`、 `additionalStoragePools` 和/或 `excludeStoragePools`。

该 `storagePools` 参数有助于将存储限制为与任何指定属性匹配的一组池。该 `additionalStoragePools` 参数用于扩展 Trident 用于配置的池集以及由属性和 `storagePools` 参数选择的池集。您可以单独使用参数或同时使用两者，以确保选择了相应的存储池集。

`excludeStoragePools` 参数用于专门排除与属性匹配的已列出池集。

模拟 **QoS** 策略

如果要设计模拟服务质量策略的存储类，请创建一个具有 `media` 属性为 `hdd` 或 `ssd` 的存储类。根据存储类中提到的 `media` 属性，Trident 将选择适当的后端来提供 `hdd` 或 `ssd` 聚合以匹配介质属性，然后将卷的配置引导到特定的聚合。因此，我们可以创建一个存储类 PREMIUM，其 `media` 属性设置为 `ssd`，可以将其归类为 PREMIUM QoS 策略。我们可以创建另一个存储类 STANDARD，将介质属性设置为 `hdd`，可以将其归类为 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可以定义为 QoS 策略的 Element 设备。

根据特定功能使用后端

存储类可以设计为在启用了精简和厚配置、快照、克隆和加密等功能的特定后端上指导卷配置。若要指定要使用的存储，请创建指定适当后端并启用所需功能的存储类。

虚拟池

虚拟池可用于所有 Trident 后端。您可以使用 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创建可通过存储类引用的抽象级别，以提高后端卷的灵活性和高效放置。可以使用相同的服务类别定义不同的后端。此外，可以在同一后端创建多个存储池，但具有不同的特征。当存储类配置有带有特定标签的选择器时，Trident 会选择与所有选择器标签匹配的后端来放置卷。如果存储类选择器标签与多个存储池匹配，Trident 将选择其中一个来配置卷。

虚拟池设计

创建后端时，通常可以指定一组参数。管理员无法使用相同的存储凭据和不同的参数集创建另一个后端。随着虚拟池的引入，这一问题得到了缓解。虚拟池是在后端和 Kubernetes Storage Class 之间引入的一个级别抽象，以便管理员可以以与后端无关的方式定义参数以及可以通过 Kubernetes Storage Classes 作为选择器引用的标签。可以为所有支持的 NetApp 后端使用 Trident 定义虚拟池。该列表包括 SolidFire/NetApp HCI、ONTAP 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。还建议不要编辑/修改现有虚拟池的属性，而是定义新的虚拟池。

模拟不同的服务级别/**QoS**

可以为模拟服务类设计虚拟池。使用 Cloud Volume Service for Azure NetApp Files 的虚拟池实现，让我们检查如何设置不同的服务类。使用多个标签配置 Azure NetApp Files 后端，代表不同的性能级别。将 `servicelevel` 方面设置为适当的性能水平，并在每个标签下添加其他必需的方面。现在创建映射到不同虚拟池的不同 Kubernetes Storage Classes。使用 `parameters.selector`字段，每个 StorageClass 调用哪些虚拟池可用于托管卷。

分配特定的一组方面

可以从单个存储后端设计具有特定方面的多个虚拟池。为此，请使用多个标签配置后端，并在每个标签下设置所需的方面。现在，使用映射到不同虚拟池的 `parameters.selector` 字段创建不同的 Kubernetes 存储类。在后端调配的卷将具有所选虚拟池中定义的方面。

影响存储配置的 **PVC** 特性

在创建 PVC 时，超出请求存储类的某些参数可能会影响 Trident 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所选的托管存储请求的后端。

Trident 将尝试根据以下矩阵匹配与指定访问方法一起使用的存储协议。这与底层存储平台无关。

	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
iSCSI	是	是	是（Raw 块）
NFS	是	是	是

在未配置 NFS 后端的情况下，提交给 Trident 部署的 ReadWriteMany PVC 请求将导致未配置任何卷。为此，请求者应使用适合其应用程序的访问模式。

卷操作

修改持久卷

持久卷是 Kubernetes 中的不可变对象，但有两个例外。创建后，可以修改回收策略和大小。但是，这并不能阻止在 Kubernetes 之外修改卷的某些方面。这可能是理想的，以便为特定应用定制卷，确保容量不会意外消耗，或者只是出于任何原因将卷移动到不同的存储控制器。

目前，Kubernetes 树内配置程序不支持 NFS、iSCSI 或 FC PV 的卷大小调整操作。Trident 支持扩展 NFS、iSCSI 和 FC 卷。

创建后，便无法修改 PV 的连接详细信息。

按需创建卷快照

Trident 支持使用 CSI 框架创建按需卷快照和从快照创建 PVC。快照为维护数据的时间点副本提供了一种方便的方法，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 PersistentVolumes。要完成此操作，只需创建一个 PersistentVolumeClaim 并提及 `datasource`作为需要从中创建卷的必需快照。Trident 将通过使用快照上存在的数据创建卷来处理此 PVC。使用此功能，可以跨区域复制数据、创建测试环境、完全替换损坏或损坏的生产卷，或检索特定文件和目录并将其传输到另一个附加卷。

移动集群中的卷

存储管理员能够以不中断存储使用者的方式将卷在 ONTAP 集群中的聚合和控制器之间移动。此操作不会影响 Trident 或 Kubernetes 集群，只要目标聚合是 Trident 正在使用的 SVM 有权访问的聚合。重要的是，如果聚合已新添加到 SVM，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。

但是，Trident 不会自动支持跨后端移动卷。这包括同一集群中的 SVM 之间、集群之间或不同存储平台上的 SVM（即使该存储系统连接到 Trident）。

如果将卷复制到其他位置，则可以使用卷导入功能将当前卷导入 Trident。

扩展卷

Trident 支持调整 NFS、iSCSI 和 FC PV 的大小。这使用户能够通过 Kubernetes 层直接调整卷的大小。所有主要 NetApp 存储平台（包括 ONTAP 和 SolidFire/NetApp HCI 后端）都可以进行卷扩展。要允许稍后进行可能的扩展，请在与卷关联的 StorageClass 中将 `allowVolumeExpansion`设置为 `true`。每当永久卷需要调整大小时，将永久卷声明中的 `spec.resources.requests.storage`注释编辑为所需的卷大小。Trident 将自动调整存储集群上的卷大小。

将现有卷导入 **Kubernetes**

卷导入提供了将现有存储卷导入 Kubernetes 环境的功能。目前 `ontap-nas`、 `ontap-nas-flexgroup`、`solidfire-san`和 `azure-netapp-files`驱动程序支持此功能。此功能在将现有应用程序移植到 Kubernetes 或灾难恢复场景期间非常有用。

使用 ONTAP 和 `solidfire-san`驱动程序时，使用命令 `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`将现有卷导入 Kubernetes 以由 Trident 管理。import volume 命令中使用的 PVC YAML 或 JSON 文件指向将 Trident 标识为配置程序的存储类。使用 NetApp HCI/SolidFire 后端时，请确保卷名称是唯一的。如果卷名重复，请将该卷克隆为唯一的名称，以便卷导入功能可以区分它们。

如果使用 `azure-netapp-files` 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 以由 Trident 管理。这确保了唯一的卷引用。

执行上述命令时，Trident 将在后端找到卷并读取其大小。它将自动添加（并在必要时覆盖）配置的 PVC 的卷大小。然后，Trident 创建新的 PV，Kubernetes 将 PVC 绑定到 PV。

如果容器的部署需要特定的导入 PVC，则它将保持挂起状态，直到通过卷导入过程绑定 PVC/PV 对。在 PVC/PV 对绑定后，如果没有其他问题，容器应该启动。

Registry 服务

已将部署和管理注册表的存储记录在 "netapp.io" 的 "博客" 中。

日志记录服务

与其他 OpenShift 服务一样，日志记录服务使用 Ansible 部署，配置参数由提供给剧本的清单文件（又名主机）提供。将涵盖两种安装方法：在初始 OpenShift 安装期间部署日志记录和在 OpenShift 安装后部署日志记录。

> 从 Red Hat OpenShift 版本 3.9 开始，由于担心数据损坏，官方文档建议不要将 NFS 用于日志记录服务。这是基于 Red Hat 对其产品的测试。ONTAP NFS 服务器没有这些问题，并且可以轻松支持日志记录部署。最终，日志记录服务的协议选择取决于您，只需知道在使用 NetApp 平台时两者都可以很好地工作，如果这是您的偏好，则没有理由避免使用 NFS。

如果选择将 NFS 与日志记录服务一起使用，则需要将 Ansible 变量 `openshift_enable_unsupported_configurations`设置为 `true`以防止安装程序失败。

开始使用

日志记录服务可以可选地为应用程序以及 OpenShift 集群本身的核心操作部署。如果选择部署操作日志记录，则通过将变量 `openshift_logging_use_ops`指定为 `true`，将创建服务的两个实例。控制操作的日志记录实例的变量包含"ops"，而应用程序的实例不包含"ops"。

根据部署方法配置 Ansible 变量对于确保底层服务使用正确的存储非常重要。让我们来看看每个部署方法的选

项。

下表仅包含与日志记录服务相关的存储配置变量。您可以在 "Red Hat OpenShift 日志记录文档" 中找到其他选项，应根据您的部署情况进行查看、配置和使用。

下表中的变量将导致 Ansible 剧本使用提供的详细信息为日志记录服务创建 PV 和 PVC。此方法的灵活性远不如在 OpenShift 安装后使用组件安装剧本，但是，如果您有可用的现有卷，这也是一个选择。

变量	详细信息
`openshift_logging_storage_kind`	设置为 `nfs`让安装程序为日志记录服务创建 NFS PV。
`openshift_logging_storage_host`	NFS 主机的主机名或 IP 地址。这应设置为虚拟机的 dataLIF。
`openshift_logging_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷连接为 `/openshift_logging`，则此变量将使用该路径。
`openshift_logging_storage_volume_name`	要创建的 PV 的名称，例如 `pv_ose_logs`。
`openshift_logging_storage_volume_size`	NFS 导出的大小，例如 `100Gi`。

如果您的 OpenShift 集群已在运行，因此 Trident 已部署和配置，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 以使用动态配置的卷。
`openshift_logging_es_pvc_storage_class_name`	将在 PVC 中使用的存储类的名称。
`openshift_logging_es_pvc_size`	PVC 中请求的卷的大小。
`openshift_logging_es_pvc_prefix`	日志记录服务使用的 PVC 的前缀。
`openshift_logging_es_ops_pvc_dynamic`	设置为 `true`以将动态配置的卷用于 ops 日志记录实例。
`openshift_logging_es_ops_pvc_storage_class_name`	ops 日志记录实例的存储类的名称。
`openshift_logging_es_ops_pvc_size`	ops 实例的卷请求的大小。
`openshift_logging_es_ops_pvc_prefix`	ops 实例 PVC 的前缀。

部署日志记录堆栈

如果将日志记录部署为初始 OpenShift 安装过程的一部分，则只需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 完成后立即提供服务。

但是，如果您在初始安装后进行部署，则 Ansible 需要使用组件攻略。此过程可能会因 OpenShift 的不同版本而略有不同，因此请务必阅读并遵循"Red Hat OpenShift Container Platform 3.11 文档"您的版本。

指标服务

指标服务为管理员提供有关 OpenShift 集群状态、资源利用率和可用性的宝贵信息。这对于 pod 自动缩放功能也是必要的，许多组织将指标服务中的数据用于其退款和/或显示应用程序。

与日志记录服务一样，OpenShift 作为一个整体，Ansible 用于部署指标服务。此外，与日志记录服务一样，可以在集群的初始设置期间或使用组件安装方法运行后部署指标服务。下表包含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，应根据您的部署进行审查、配置和使用。

变量	详细信息
`openshift_metrics_storage_kind`	设置为 `nfs`让安装程序为日志记录服务创建 NFS PV。
`openshift_metrics_storage_host`	NFS 主机的主机名或 IP 地址。这应该设置为 SVM 的 dataLIF。
`openshift_metrics_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷连接为 `/openshift_metrics`，则您将使用该路径作为此变量。
`openshift_metrics_storage_volume_name`	要创建的 PV 的名称，例如 `pv_ose_metrics`。
`openshift_metrics_storage_volume_size`	NFS 导出的大小，例如 `100Gi`。

如果您的 OpenShift 集群已在运行，因此 Trident 已部署和配置，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`	用于度量 PVC 的前缀。
`openshift_metrics_cassandra_pvc_size`	要请求的卷的大小。
`openshift_metrics_cassandra_storage_type`	要用于指标的存储类型，必须将其设置为 dynamic，以便 Ansible 使用适当的存储类创建 PVC。
`openshift_metrics_cassanda_pvc_storage_class_name`	要使用的存储类的名称。

部署指标服务

使用主机/清单文件中定义的相应 Ansible 变量，使用 Ansible 部署服务。如果您在 OpenShift 安装时部署，则 PV 将自动创建和使用。如果使用组件剧本进行部署，在 OpenShift 安装后，Ansible 会创建所需的任何 PVC，并在 Trident 为它们配置存储之后，部署服务。

上述变量以及部署过程可能会随着 OpenShift 的每个版本而变化。确保查看并遵循 "Red Hat 的 OpenShift 部署指南" 您的版本，以便为您的环境配置。

数据保护和灾难恢复

了解 Trident 和使用 Trident 创建的卷的保护和恢复选项。您应该为每个具有持久性要求的

应用程序制定数据保护和恢复策略。

Trident 复制和恢复

您可以创建备份以在发生灾难时还原 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 "Kubernetes：备份 etcd 集群" 备份 Kubernetes 集群 etcd。
2. 将备份项目放置在 FlexVol 卷上

> NetApp 建议您通过将 FlexVol 所在的 SVM 与另一个 SVM 建立 SnapMirror 关系来保护该 SVM。

Trident 恢复

使用 Kubernetes CRD 和 Kubernetes 集群 etcd snapshot，可以恢复 Trident。

步骤

1. 从目标 SVM 中，将包含 Kubernetes etcd 数据文件和证书的卷挂载到将设置为主节点的主机上。
2. 复制与 Kubernetes 集群相关的所有必需证书，位于 `/etc/kubernetes/pki` 下，以及 etcd 成员文件，位于 `/var/lib/etcd` 下。
3. 使用 "Kubernetes：恢复 etcd 集群" 从 etcd 备份还原 Kubernetes 集群。
4. 运行 `kubectl get crd`以验证所有 Trident 自定义资源已出现并检索 Trident 对象以验证所有数据可用。

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"来复制 SVM。

在发生灾难时，您可以激活 SnapMirror 目标 SVM 以开始提供数据。系统还原后，您可以切换回主系统。

关于此任务

使用 SnapMirror SVM 复制功能时，请考虑以下事项：

- 您应该为每个启用了 SVM-DR 的 SVM 创建一个独特的后端。
- 配置存储类，仅在需要时选择复制的后端，以避免将不需要复制的卷调配到支持 SVM-DR 的后端。
- 应用程序管理员应了解与复制相关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 允许您设置选项以控制要复制的内容。您需要知道在执行 使用 Trident 进行 SVM 恢复 时选择了哪

些选项。

- "-identity-preserve true" 复制整个 SVM 配置。
- "-discard-configs network" 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 不会自动检测 SVM 故障。在发生灾难时，管理员可以手动启动 Trident 故障转移到新的 SVM。

步骤

1. 取消计划和正在进行的 SnapMirror 传输，断开复制关系，停止源 SVM，然后激活 SnapMirror 目标 SVM。
2. 如果在配置 SVM 复制时指定了 `-identity-preserve false`或 `-discard-config network`，请更新 Trident 后端定义文件中的 `managementLIF`和 `dataLIF`。
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` 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. 如果指定了 `-identity-preserve false` 或 `discard-config network`，则必须重启所有应用程序 Pod。

> 如果指定 `-identity-preserve true`，则在激活目标 SVM 时，由 Trident 配置的所有卷都开始提供数据。

卷复制和恢复

Trident 无法配置 SnapMirror 复制关系，但存储管理员可以使用 "ONTAP SnapMirror 复制和恢复" 来复制由 Trident 创建的卷。

然后，您可以使用 "tridentctl volume import" 将恢复的卷导入 Trident。

> `ontap-nas-economy`、 `ontap-san-economy` 或 `ontap-flexgroup-economy` 驱动程序不支持导入。

Snapshot 数据保护

您可以使用以下方式保护和恢复数据：

- 用于创建持久卷 (PV) 的 Kubernetes 卷快照的外部快照控制器和 CRD。

  "卷快照"

- ONTAP Snapshots 可还原卷的整个内容或恢复单个文件或 LUN。

  "ONTAP 快照"

使用 Trident 自动化有状态应用程序的故障转移

Trident 的强制分离功能允许您自动从 Kubernetes 集群中的不正常节点分离卷，从而防止数据损坏并确保应用程序可用性。此功能在节点无响应或因维护而离线的情况下特别有用。

关于强制分离的详细信息

强制分离仅适用于 `ontap-san`、 `ontap-san-economy`、 `ontap-nas`和 `ontap-nas-economy`。在启用强制分离之前，必须在 Kubernetes 集群上启用非优雅节点关闭（NGNS）。默认情况下，Kubernetes 1.28 及更高版本已启用 NGNS。有关详细信息，请参阅"Kubernetes：非 Graceful 节点关闭"。

> 使用 `ontap-nas` 或 `ontap-nas-economy` 驱动程序时，需要在后端配置中将 `autoExportPolicy` 参数设置为 `true`，以便 Trident 可以使用托管导出策略限制来自应用了污点的 Kubernetes 节点的访问。

> 由于 Trident 依赖于 Kubernetes NGNS，因此在重新安排所有无法忍受的工作负载之前，请勿从不正常的节点中删除 `out-of-service`污点。不计后果地应用或删除污点可能会危及后端数据保护。

当 Kubernetes 集群管理员将 `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute` taint 应用到节点并将 `enableForceDetach`设置为 `true`时，Trident 将确定节点状态并：

1. 停止对装载到该节点的卷的后端 I/O 访问。
2. 将 Trident 节点对象标记为 `dirty`（对新发布不安全）。

> Trident 控制器将拒绝新的发布卷请求，直到节点被 Trident 节点 Pod 重新限定（在被标记为 `dirty`之后）。在 `Trident` 能够验证节点 `clean`（对新发布安全）之前，将不接受使用已挂载 PVC 调度的任何工作负载（即使在集群节点健康且就绪之后）。

当节点运行状况恢复并清除污点时，Trident 将：

1. 识别并清除节点上过时的已发布路径。
2. 如果该节点处于 `cleanable`状态（已清除停用污点，并且该节点处于 `Ready`状态），并且所有过时、已发布的路径都是干净的，则 Trident 将重新接收该节点为 `clean`并允许新发布的卷进入该节点。

有关自动故障转移的详细信息

您可以通过与 "节点运行状况检查 (NHC) 操作器" 集成来自动执行强制分离过程。当发生节点故障时，NHC 会触发 Trident 节点修复（TNR），并通过在 Trident 命名空间中创建 TridentNodeRemediation CR 来定义故障节点，从而自动进行强制分离。TNR 仅在节点故障时创建，并在节点恢复联机或节点被删除后由 NHC 删除。

节点 **pod** 移除过程失败

自动故障转移选择要从故障节点中删除的工作负载。创建 TNR 后，TNR 控制器将节点标记为脏节点，防止任何新的卷发布，并开始删除强制拆卸支持的 Pod 及其卷附件。

自动故障转移支持强制拆卸支持的所有卷/PVC：

- 使用自动导出策略的 NAS 和 NAS-economy 卷（尚不支持 SMB）。
- SAN 和 SAN-economy 卷。

请参阅 [关于强制分离的详细信息]。

默认行为：

- 使用强制拆卸支持的卷的 Pod 将从故障节点中删除。Kubernetes 会将这些重新调度到健康的节点上。
- 使用强制拆卸不支持的卷（包括非 Trident 卷）的 Pod 不会从故障节点中删除。
- 除非设置了 Pod 注释 `trident.netapp.io/podRemediationPolicy: delete`，否则不会从故障节点中删除无状态 Pod（非 PVC）。

覆盖 **Pod** 移除行为：

可以使用 Pod 注释自定义 Pod 移除行为： `trident.netapp.io/podRemediationPolicy[retain, delete]`。当发生故障转移时，会检查并使用这些注释。将注释应用于 Kubernetes deployment/replicaset pod 规范，以防止故障转移后注释消失：

- `retain` - 在自动故障转移期间，不会从故障节点中删除 Pod。
- `delete` - 在自动故障转移期间，Pod 将从故障节点中删除。

这些注释可以应用于任何 pod。

- 对于支持强制分离的卷，I/O 操作仅在故障节点上被阻止。
- 对于不支持强制分离的卷，存在数据损坏和多重连接问题的风险。

TridentNodeRemediation CR

TridentNodeRemediation (TNR) CR 定义失败的节点。TNR 的名称是失败节点的名称。

TNR 示例：

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediation
metadata:
  name: <K8s-node-name>
spec: {}
```

TNR 状态：使用以下命令查看 TNR 的状态：
`kubectl get tnr <name> -n <trident-namespace>`

TNR 可能处于以下状态之一：

- 修复中：
  - 停止对装载到该节点的强制拆卸所支持的卷的后端 I/O 访问。
  - Trident 节点对象被标记为脏（对于新出版物不安全）。
  - 从节点中删除 Pod 和卷附件
- *NodeRecoveryPending*：
  - 控制器正在等待节点恢复联机。
  - 节点联机后，发布强制将确保节点干净，可供新卷发布。
- 如果从 K8s 中删除节点，TNR 控制器将删除 TNR 并停止对账。
- 成功：
  - 已成功完成所有修正和节点恢复步骤。节点已干净，可供新卷发布。
- 失败：
  - 不可恢复的错误。错误原因在 CR 的 status.message 字段中设置。

启用自动故障转移

前提条件：

- 在启用 automated-failover 之前，请确保已启用强制分离。有关详细信息，请参阅 [关于强制分离的详细信息]。
- 在 Kubernetes 集群中安装节点运行状况检查 (NHC)。
  - "安装 operator-sdk"。
  - 在集群中安装 Operator Lifecycle Manager (OLM)（如果尚未安装）： `operator-sdk olm install`。
  - 安装 Node Health check Operator： `kubectl create -f https://operatorhub.io/install/node-healthcheck-operator.yaml`。

> 您还可以使用其他方法来检测节点故障，如下面的 [Integrating Custom Node Health Check Solutions] 部分所述。

有关详细信息，请参见 "节点健康检查运算符"。

步骤

1. 在 Trident 命名空间中创建 NodeHealthCheck (NHC) CR 以监视集群中的工作节点。示例：

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: <CR name>
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  remediationTemplate:
    apiVersion: trident.netapp.io/v1
    kind: TridentNodeRemediationTemplate
    namespace: <Trident installation namespace>
    name: trident-node-remediation-template
  minHealthy: 0 # Trigger force-detach upon one or more node failures
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 0s
    - type: Ready
      status: Unknown
      duration: 0s
```

2. 在 `trident` 命名空间中应用节点运行状况检查 CR。

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

上述 CR 配置为监视 K8s worker 节点的节点条件 Ready: false 和 Unknown。节点进入 Ready: false 或 Ready: Unknown 状态时将触发 Automated-Failover。

CR 中的 `unhealthyConditions`使用 0 秒宽限期。这导致自动故障转移在 K8s 设置节点条件 Ready: false 时立即触发，这是在 K8s 失去节点的心跳后设置的。K8s 的默认值为最后一次心跳后等待 40 秒，然后设置 Ready: false。可以在 K8s 部署选项中自定义此宽限期。

要查看其他配置选项，请参阅 "Node-Healthcheck-Operator 文档"。

其他设置信息

安装 Trident 并启用强制拆卸功能时，系统将在 Trident 命名空间中自动创建两个额外资源，以促进与 NHC 的集成：TridentNodeRemediationTemplate (TNRT) 和 ClusterRole。

TridentNodeRemediationTemplate (TNRT)：

TNRT 充当 NHC 控制器的模板，该控制器根据需要使用 TNRT 生成 TNR 资源。

```yaml
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

ClusterRole：

启用强制拆卸后，在安装过程中也会添加群集角色。这赋予了 NHC 对 Trident 命名空间中 TNR 的权限。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.ext-remediation/aggregate-to-ext-remediation: "true"
  name: tridentnoderemediation-access
rules:
- apiGroups:
  - trident.netapp.io
  resources:
  - tridentnoderemediationtemplates
  - tridentnoderemediations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

K8s 集群升级和维护

为了防止任何故障转移，请在 K8s 维护或升级期间暂停自动故障转移，其中节点预计会停机或重新启动。您可以通过修补 CR 来暂停 NHC CR（如上所述）：

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

这会暂停自动故障转移。要重新启用自动故障转移，请在维护完成后从规范中删除 pauseRequests。

限制

- 仅在强制分离支持的卷的故障节点上阻止 I/O 操作。只有使用强制分离支持的卷/PVC 的 Pod 才会自动移除。
- 自动故障转移和强制拆卸在 trident-controller pod 内运行。如果托管 trident-controller 的节点出现故障，则自动故障转移将被延迟，直到 K8s 将 pod 移动到健康节点。

集成自定义节点运行状况检查解决方案

您可以将 Node Healthcheck Operator 替换为备用节点故障检测工具以触发自动故障转移。为了确保与自动故障转移机制的兼容性，您的自定义解决方案应：

- 检测到节点故障时创建 TNR，使用故障节点的名称作为 TNR CR 名称。
- 当节点已恢复且 TNR 处于 Succeeded 状态时，请删除 TNR。

安全性

安全性

请使用此处列出的建议，以确保 Trident 的安装安全。

在自己的命名空间中运行 **Trident**

必须防止应用程序、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，以确保可靠的存储并阻止潜在的恶意活动。

要将其他应用程序和用户与 Trident 分离，请始终在自己的 Kubernetes 命名空间中安装 Trident(`trident`）。将 Trident 放在自己的命名空间中可确保只有 Kubernetes 管理人员才能访问 Trident Pod 和存储在命名空间 CRD 对象中的工件（如后端和 CHAP 机密（如果适用））。您应该确保只允许管理员访问 Trident 命名空间，从而访问 `tridentctl`应用程序。

对 **ONTAP SAN** 后端使用 **CHAP** 身份验证

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`驱动程序）。NetApp 建议使用双向 CHAP 与 Trident 进行主机和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 可以设置双向 CHAP 并通过 `tridentctl`管理 CHAP 用户名和机密。请参阅"准备使用 ONTAP SAN 驱动程序配置后端"以了解 Trident 如何在 ONTAP 后端上配置 CHAP。

将 **CHAP** 身份验证与 **NetApp HCI** 和 **SolidFire** 后端一起使用

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 使用一个秘密对象，其中每个租户包含两个 CHAP 密码。安装 Trident 后，它会管理 CHAP 机密并将其存储在相应 PV 的 `tridentvolume` CR 对象中。创建 PV 时，Trident 使用 CHAP 机密来启动 iSCSI 会话，并通过 CHAP 与 NetApp HCI 和 SolidFire 系统进行通信。

由 Trident 创建的卷不与任何卷访问组关联。

将 **Trident** 与 **NVE** 和 **NAE** 配合使用

NetApp ONTAP 提供静态数据加密，以便在磁盘被盗、退回或重新使用时保护敏感数据。有关详细信息，请参阅 "配置 NetApp 卷加密概述"。

- 如果在后端启用了 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 `""`以创建启用 NAE 的卷。
- 如果未在后端启用 NAE，则在 Trident 中配置的任何卷都将启用 NVE，除非在后端配置中将 NVE 加密标志设置为 `false`（默认值）。

> 在启用 NAE 的后端上在 Trident 中创建的卷必须进行 NVE 或 NAE 加密。
>
> - 您可以在 Trident 后端配置中将 NVE 加密标志设置为 `true`，以覆盖 NAE 加密，并在每个卷的基础上使用特定的加密密钥。
> - 在启用 NAE 的后端上将 NVE 加密标志设置为 `false`可创建启用 NAE 的卷。无法通过将 NVE 加密标志设置为 `false`来禁用 NAE 加密。

- 您可以通过显式设置 NVE 加密标志为 `true`在 Trident 中手动创建 NVE 卷。

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 Unified Key Setup (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 加密的卷使用 aes-xts-plain64 密码和模式，如 "NIST" 所推荐。

> ASA r2 系统不支持 LUKS 加密。有关 ASA r2 系统的信息，请参见 "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）。有关更多信息，请访问 "Gitlab：cryptsetup"。
- 出于性能原因，NetApp 建议工作节点支持高级加密标准新指令 (AES-NI)。要验证 AES-NI 支持，请运行以下命令：

```
grep "aes" /proc/cpuinfo
```

如果没有返回任何内容，则表示您的处理器不支持 AES-NI。有关 AES-NI 的更多信息，请访问："Intel：高级加密标准指令 (AES-NI)"。

启用 LUKS 加密

您可以使用 Linux Unified Key Setup (LUKS) 为 ONTAP SAN 和 ONTAP SAN ECONOMY 卷启用每个卷的主机端加密。

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` 来定义使用 LUKS 加密的存储池。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 加密的卷无法利用 ONTAP 重复数据删除和压缩。

用于导入 **LUKS** 卷的后端配置

要导入 LUKS 卷，您必须在后端将 `luksEncryption`设置为(`true。 luksEncryption`选项告诉 Trident 卷是否符合 LUKS 标准(`true`）或不符合 LUKS 标准(`false`），如以下示例所示。

```
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，请将注释 `trident.netapp.io/luksEncryption` 设置为 `true` 并在 PVC 中包含启用 LUKS 的存储类，如本示例所示。

```
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。

> 在验证密码短语不再被任何卷、快照或密码引用之前，请不要忘记该密码短语。如果引用的密码短语丢失，您可能无法挂载卷，并且数据将保持加密且无法访问。

关于此任务

当在指定新的 LUKS 密码短语后创建挂载卷的 Pod 时，会发生 LUKS 密码短语轮换。创建新 Pod 时，Trident 会将卷上的 LUKS 密码短语与密码中的活动密码短语进行比较。

- 如果卷上的密码与 secret 中的活动密码不匹配，则会发生旋转。
- 如果卷上的密码与 secret 中的活动密码匹配，则此 `previous-luks-passphrase` 参数将被忽略。

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. 识别卷或快照上的现有密码短语。

   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。请确保 `previous-luke-passphrase-name` 和 `previous-luks-passphrase` 匹配之前的密码短语。

   ```
   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. 创建挂载卷的新 pod。这是启动轮换所必需的。
5. 验证密码短语已轮换。

   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

当仅在卷和快照上返回新密码短语时，密码短语已轮换。

> 如果返回两个密码短语，例如 `luksPassphraseNames: ["B", "A"]`，则旋转是不完整的。您可以触发一个新的 pod 来尝试完成旋转。

启用卷扩展

您可以在 LUKS 加密的卷上启用卷扩展。

步骤

1. 启用 `CSINodeExpandSecret`功能门（beta 1.25+）。有关详细信息，请参见"Kubernetes 1.25：使用 Secrets 进行节点驱动的 CSI 卷扩展"。
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 传输中加密

通过使用 Kerberos 在线加密，您可以为托管群集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 支持将 Kerberos 加密用于 ONTAP 作为存储后端：

- 本地 **ONTAP** - Trident 支持通过 NFSv3 和 NFSv4 连接从 Red Hat OpenShift 和上游 Kubernetes 集群到本地 ONTAP 卷的 Kerberos 加密。

可以创建、删除、调整大小、快照、克隆、只读克隆以及导入使用 NFS 加密的卷。

使用本地 **ONTAP** 卷配置运行中的 **Kerberos** 加密

您可以在托管集群与本地 ONTAP 存储后端之间的存储流量上启用 Kerberos 加密。

> 仅使用 `ontap-nas` 存储驱动程序支持针对具有本地 ONTAP 存储后端的 NFS 流量的 Kerberos 加密。

开始之前

- 请确保您拥有该 `tridentctl` 实用程序的访问权限。
- 确保您拥有对 ONTAP 存储后端的管理员访问权限。
- 确保您知道要从 ONTAP 存储后端共享的卷的名称。
- 确保已准备好 ONTAP 存储虚拟机，以支持 NFS 卷的 Kerberos 加密。请参阅 "在 dataLIF 上启用 Kerberos" 获取说明。
- 请确保您使用 Kerberos 加密的任何 NFSv4 卷配置正确。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，或创建支持 Kerberos 加密的新导出策略，用于 ONTAP 存储 VM 根卷以及与上游 Kubernetes 集群共享的任何 ONTAP 卷。您添加的导出策略规则或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

使用 NFS、NFSv3 和 NFSv4 访问协议配置导出策略。

访问详细信息

根据您对卷的需求，您可以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** -（具有身份保护的身份验证和加密）
- **Kerberos 5p** -（具有身份和隐私保护的身份验证和加密）

使用适当的访问权限配置 ONTAP 导出策略规则。例如，如果群集将使用 Kerberos 5i 和 Kerberos 5p 加密的混合方式装载 NFS 卷，请使用以下访问设置：

类型	只读访问	读/写访问权限	超级用户访问
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"
- "将规则添加到导出策略"

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

在创建配置 Kerberos 加密的存储后端配置文件时，可以使用 `spec.nfsMountOptions` 参数指定三个不同版本的 Kerberos 加密之一：

- `spec.nfsMountOptions: sec=krb5` （身份验证和加密）
- `spec.nfsMountOptions: sec=krb5i` （具有身份保护的身份验证和加密）
- `spec.nfsMountOptions: sec=krb5p`（具有身份和隐私保护的身份验证和加密）

仅指定一个 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. 在托管群集上，使用以下示例创建存储后端配置文件。用您环境中的信息替换括号 <> 中的值：

```
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. 使用上一步中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，则后端配置有问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在识别并更正配置文件的问题后，您可以再次运行 create 命令。

创建存储类

您可以创建存储类，以使用 Kerberos 加密配置卷。

关于此任务

在创建存储类对象时，可以使用 `mountOptions` 参数指定 Kerberos 加密的三个不同版本之一：

- `mountOptions: sec=krb5`（身份验证和加密）
- `mountOptions: sec=krb5i`（具有身份保护的身份验证和加密）
- `mountOptions: sec=krb5p`（具有身份和隐私保护的身份验证和加密）

仅指定一个 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级别不同，则存储类对象优先。

步骤

1. 使用以下示例创建 StorageClass Kubernetes 对象：

```
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. 确保已创建存储类：

```
kubectl get sc ontap-nas-sc
```

此时将显示与以下内容类似的输出：

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

配置卷

创建存储后端和存储类后，现在可以配置卷。有关说明，请参阅 "预配卷"。

使用 **Azure NetApp Files** 卷配置运行中的 **Kerberos** 加密

您可以在托管集群与单个 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上启用 Kerberos 加密。

开始之前

- 请确保已在托管红帽 OpenShift 群集上启用 Trident。
- 请确保您拥有该 `tridentctl` 实用程序的访问权限。
- 通过注意要求并按照 "Azure NetApp Files 文档"中的说明，确保已为 Kerberos 加密准备 Azure NetApp Files 存储后端。
- 请确保您使用 Kerberos 加密的任何 NFSv4 卷配置正确。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

当您创建配置 Kerberos 加密的存储后端配置文件时，您可以将其定义为应在以下两种可能的级别之一应用：

- 使用 `spec.kerberos` 字段的*存储后端级别*
- 使用 `spec.storage.kerberos` 字段的 虚拟池级别

在虚拟池级别定义配置时，将使用存储类中的标签选择池。

在任一级别，您都可以指定 Kerberos 加密的三个不同版本之一：

- `kerberos: sec=krb5` （身份验证和加密）
- `kerberos: sec=krb5i`（具有身份保护的身份验证和加密）
- `kerberos: sec=krb5p` （具有身份和隐私保护的身份验证和加密）

步骤

1. 在托管群集上，使用以下示例之一创建存储后端配置文件，具体取决于需要定义存储后端的位置（存储后端级别或虚拟池级别）。使用环境中的信息替换括号 <> 中的值：

存储后端级别示例

```
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
  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. 使用上一步中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，则后端配置有问题。您可以通过运行以下命令查看日志以确定原因：

```
tridentctl logs
```

在识别并更正配置文件的问题后，您可以再次运行 create 命令。

创建存储类

您可以创建存储类，以使用 Kerberos 加密配置卷。

步骤

1. 使用以下示例创建 StorageClass Kubernetes 对象：

```
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. 确保已创建存储类：

```
kubectl get sc -sc-nfs
```

此时将显示与以下内容类似的输出：

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

配置卷

创建存储后端和存储类后，现在可以配置卷。有关说明，请参阅 "预配卷"。

使用 Trident Protect 保护应用程序

了解 Trident Protect

NetApp Trident Protect 提供先进的应用程序数据管理功能，增强由 NetApp ONTAP 存储系统和 NetApp Trident CSI 存储配置程序支持的有状态 Kubernetes 应用程序的功能和可用性。Trident Protect 简化了跨公共云和本地环境的容器化工作负载的管理、保护和移动。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，使用 Trident Protect 保护应用程序。

下一步是什么？

在安装之前，您可以了解 Trident Protect 的要求：

- "Trident Protect 要求"

安装 Trident Protect

Trident Protect 要求

首先验证操作环境、应用程序群集、应用程序和许可证的准备情况。确保您的环境满足这些要求，以部署和运行 Trident Protect。

Trident Protect Kubernetes 集群兼容性

Trident Protect 与各种完全托管和自我托管的 Kubernetes 产品兼容，包括：

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Microsoft Azure Kubernetes Service (AKS)
- Red Hat OpenShift
- SUSE Rancher
- VMware Tanzu Portfolio
- 上游 Kubernetes

- 仅 Linux 计算节点支持 Trident Protect 备份。备份操作不支持 Windows 计算节点。
- 确保安装 Trident Protect 的集群配置了正在运行的快照控制器和相关 CRD。要安装快照控制器，请参阅 "这些说明"。
- 确保至少存在一个 VolumeSnapshotClass。有关详细信息，请参见 "VolumeSnapshotClass"。

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

请确保存储后端满足以下要求：

- 请确保连接到集群的 NetApp 存储使用 Trident 24.02 或更高版本（建议使用 Trident 24.10）。
- 确保您拥有 NetApp ONTAP 存储后端。
- 确保已配置用于存储备份的对象存储桶。
- 创建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；如果在自定义资源中指定不存在的命名空间，则操作将失败。

nas-economy 卷的要求

Trident Protect 支持到 nas-economy 卷的备份和还原操作。当前不支持到 nas-economy 卷的快照、克隆和 SnapMirror 复制。您需要为计划用于 Trident Protect 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。对于这些应用程序，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
```

您可以通过为每个 nas-economy 卷运行以下命令来启用快照目录，将 `<volume-UUID>` 替换为要更改的卷的 UUID：

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

您可以通过将 Trident 后端配置选项 `snapshotDir`设置为 `true`来默认为新卷启用快照目录。现有卷不受影响。

使用 KubeVirt VM 保护数据

Trident Protect 在数据保护操作期间为 KubeVirt 虚拟机提供文件系统冻结和解冻功能，以确保数据一致性。虚拟机冻结操作的配置方法和默认行为因 Trident Protect 版本而异，较新的版本通过 Helm 图表参数提供简化的配置。

在还原操作期间，不会还原为虚拟机 (VM) 创建的任何 `VirtualMachineSnapshots`。

Trident Protect 25.10 及更高版本

Trident Protect 在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统，以确保一致性。从 Trident Protect 25.10 开始，您可以在 Helm chart 安装期间使用 `vm.freeze` 参数禁用此行为。默认情况下，该参数处于启用状态。

```
helm install ... --set vm.freeze=false ...
```

Trident Protect 24.10.1 至 25.06

从 Trident Protect 24.10.1 开始，Trident Protect 在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。或者，您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
```

Trident Protect 24.10

Trident Protect 24.10 不会在数据保护操作期间自动确保 KubeVirt VM 文件系统的一致状态。如果要使用 Trident Protect 24.10 保护 KubeVirt VM 数据，则需要在数据保护操作之前手动为文件系统启用冻结/解冻功能。这可确保文件系统处于一致状态。

您可以通过 "配置虚拟化" 配置 Trident Protect 24.10 来管理数据保护操作期间虚拟机文件系统的冻结和解冻，然后使用以下命令：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=true -n trident-protect
```

SnapMirror 复制的要求

NetApp SnapMirror 复制可用于以下 ONTAP 解决方案的 Trident Protect：

- 本地 NetApp FAS、AFF 和 ASA 系统。目前 ASA r2 系统不支持使用 Trident 保护进行 SnapMirror 复制。
- NetApp ONTAP Select
- NetApp Cloud Volumes ONTAP
- Amazon FSx for NetApp ONTAP

ONTAP 集群对 SnapMirror 复制的要求

如果计划使用 SnapMirror 复制，请确保您的 ONTAP 群集满足以下要求：

- **NetApp Trident**：NetApp Trident 必须存在于使用 ONTAP 作为后端的源和目标 Kubernetes 集群上。Trident Protect 支持使用由以下驱动程序支持的存储类的 NetApp SnapMirror 技术进行复制：

- `ontap-nas`：NFS
- `ontap-san`：iSCSI
- `ontap-san`：FC
- `ontap-san`: NVMe/TCP（需要最低 ONTAP 版本 9.15.1）

- 许可证：必须在源和目标 ONTAP 集群上启用使用数据保护捆绑包的 ONTAP SnapMirror 异步许可证。有关详细信息，请参见 "ONTAP 中的 SnapMirror 许可概述"。

  从 ONTAP 9.10.1 开始，所有许可证都以 NetApp 许可证文件 (NLF) 的形式交付，该文件是启用多个功能的单个文件。有关详细信息，请参见 "ONTAP One 附带的许可证"。

  仅支持 SnapMirror 异步保护。

SnapMirror 复制的对等关系注意事项

如果您计划使用存储后端对等，请确保您的环境满足以下要求：

- 集群和 **SVM**：必须对等 ONTAP 存储后端。有关详细信息，请参见 "集群和 SVM 对等概述"。

  确保在两个 ONTAP 集群之间的复制关系中使用的 SVM 名称是唯一的。

- **NetApp Trident** 和 **SVM**：对等远程 SVM 必须可用于目标集群上的 NetApp Trident。
- 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 要求您配置至少一个同时支持源和目标集群复制的存储后端。如果源和目标集群相同，目标应用程序应使用与源应用程序不同的存储后端，以获得最佳弹性。

Kubernetes 集群 SnapMirror 复制需求

确保 Kubernetes 集群满足以下要求：

- **AppVault** 可访问性：源和目标集群必须具有网络访问权限才能从 AppVault 读取和写入以进行应用程序对象复制。
- 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 跨 WAN 之间的通信。

许多企业环境在 WAN 连接上实施严格的防火墙策略。在配置复制之前，请与基础架构团队一起验证这些网络要求。

安装和配置 Trident Protect

如果您的环境符合 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，也可以从您自己的私有注册表安装它。如果您的集群无法访问 Internet，从私有注册表安装会很有帮助。

从 **NetApp** 安装 **Trident Protect**

步骤

1. 添加 Trident Helm 存储库：

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

2. 使用 Helm 安装 Trident Protect。将 `<name-of-cluster>` 替换为集群名称，该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 100.2510.0 --create
-namespace --namespace trident-protect
```

3. （可选）要启用调试日志记录（建议用于故障排除），请使用：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2510.0 --create-namespace --namespace trident-protect
```

调试日志记录有助于 NetApp 支持疑难解答问题，而无需更改日志级别或重现问题。

从私有注册表安装 **Trident Protect**

如果您的 Kubernetes 集群无法访问互联网，您可以从私有映像注册表安装 Trident Protect。在这些示例中，使用环境中的信息替换括号中的值：

步骤

1. 将以下映像拉到本地计算机，更新标记，然后将其推送到专用注册表：

```
docker.io/netapp/controller:25.10.0
docker.io/netapp/restic:25.10.0
docker.io/netapp/kopia:25.10.0
docker.io/netapp/kopiablockrestore:25.10.0
docker.io/netapp/trident-autosupport:25.10.0
docker.io/netapp/exechook:25.10.0
docker.io/netapp/resourcebackup:25.10.0
docker.io/netapp/resourcerestore:25.10.0
docker.io/netapp/resourcedelete:25.10.0
docker.io/netapp/trident-protect-utils:v1.0.0
```

例如：

```
docker pull docker.io/netapp/controller:25.10.0
```

```
docker tag docker.io/netapp/controller:25.10.0 <private-registry-url>/controller:25.10.0
```

```
docker push <private-registry-url>/controller:25.10.0
```

> 要获取 Helm 图表，请首先在具有 Internet 访问权限的计算机上使用 `helm pull trident-protect --version 100.2510.0 --repo https://netapp.github.io/trident-protect-helm-chart` 下载 Helm 图表，然后将生成的 `trident-protect-100.2510.0.tgz`文件复制到离线环境，并在最后步骤中使用 `helm install trident-protect ./trident-protect-100.2510.0.tgz`而不是存储库引用进行安装。

2. 创建 Trident Protect 系统命名空间：

```
kubectl create ns trident-protect
```

3. 登录到注册表：

```
helm registry login <private-registry-url> -u <account-id> -p <api-token>
```

4. 创建用于专用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 存储库：

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

6. 创建一个名为 `protectValues.yaml` 的文件。确保其中包含以下 Trident Protect 设置：

```
---
imageRegistry: <private-registry-url>
imagePullSecrets:
  - name: regcred
```

> imageRegistry 和 imagePullSecrets 值适用于所有组件图像，包括 resourcebackup 和 resourcerestore。如果将图像推送到注册表中的特定存储库路径（例如 example.com:443/my-repo），请在注册表字段中包含完整路径。这将确保从 <private-registry-url>/<image-name>:<tag> 中提取所有图像。

7. 使用 Helm 安装 Trident Protect。将 <name_of_cluster> 替换为集群名称，该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 100.2510.0 --create
-namespace --namespace trident-protect -f protectValues.yaml
```

8. （可选）要启用调试日志记录（建议用于故障排除），请使用：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --set logLevel=debug --version
100.2510.0 --create-namespace --namespace trident-protect -f
protectValues.yaml
```

调试日志记录有助于 NetApp 支持疑难解答问题，而无需更改日志级别或重现问题。

> 要查看其他 Helm 图表配置选项，包括 AutoSupport 设置和命名空间筛选，请参阅 "自定义 Trident Protect 安装"。

安装 Trident Protect CLI 插件

您可以使用 Trident Protect 命令行插件（Trident tridentctl 实用程序的扩展）来创建 Trident Protect 自定义资源（CR）并与之交互。

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群集的计算机上。根据您的计算机使用的是 x64 还是 ARM CPU，请执行以下步骤。

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/25.10.0/tridentctl-protect-linux-amd64
```

为 **Linux ARM64 CPU** 下载插件

步骤

1. 下载 Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/25.10.0/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/25.10.0/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/25.10.0/tridentctl-protect-macos-arm64
```

1. 为插件二进制文件启用执行权限：

```
chmod +x tridentctl-protect
```

2. 将插件二进制文件复制到 PATH 变量中定义的位置。例如， `/usr/bin` 或 `/usr/local/bin`（您可能需要提升权限）：

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 或者，您可以将插件二进制文件复制到主目录中的某个位置。在这种情况下，建议确保该位置是 PATH 变量的一部分：

```
cp ./tridentctl-protect ~/bin/
```

将插件复制到 PATH 变量中的某个位置，可以通过键入 `tridentctl-protect`或 `tridentctl protect`从任何位置使用插件。

查看 **Trident CLI** 插件帮助

您可以使用内置插件帮助功能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
```

启用命令自动完成

安装 Trident Protect CLI 插件后，可以启用某些命令的自动完成功能。

启用 **Bash shell** 的自动完成

步骤

1. 创建完成脚本：

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. 在主目录中创建包含脚本的新目录：

```
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. 将以下行添加到主目录中的 `~/.bashrc` 文件：

```
source ~/.bash/completions/tridentctl-completion.bash
```

启用 **Z shell** 的自动完成

步骤

1. 创建完成脚本：

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. 在主目录中创建包含脚本的新目录：

```
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. 将以下行添加到主目录中的 `~/.zprofile` 文件：

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

结果

下次登录 shell 时，您可以使用 tridentctl-protect 插件自动完成命令。

自定义 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为 Trident Protect 容器指定资源限制。设置资源限制使您能够控制 Trident Protect 操作占用集群资源的数量。

步骤

1. 创建名为 `resourceLimits.yaml` 的文件。
2. 根据您的环境需求，使用 Trident Protect 容器的资源限制选项填充文件。

   以下配置文件示例显示了可用设置，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 `resourceLimits.yaml` 文件中的值：

```
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. 创建名为 `sccconfig.yaml` 的文件。
2. 将 SCC 选项添加到文件中，并根据环境需要修改参数。

   以下示例显示了 SCC 选项的参数的默认值：

```
scc:
  create: true
  name: trident-protect-job
  priority: 1
```

   下表描述了 SCC 选项的参数：

参数	说明	默认
create	确定是否可以创建 SCC 资源。仅当 `scc.create` 设置为 `true` 且 Helm 安装过程识别 OpenShift 环境时，才会创建 SCC 资源。如果不在 OpenShift 上操作，或者如果 `scc.create` 设置为 `false`，则不会创建任何 SCC 资源。	true
name	指定 SCC 的名称。	trident-protect-job
优先级	定义 SCC 的优先级。优先级值较高的 SCC 在优先级值较低的 SCC 之前进行评估。	1

3. 应用 `sccconfig.yaml` 文件中的值：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f sccconfig.yaml --reuse-values
```

这将用 `sccconfig.yaml` 文件中指定的值替换默认值。

配置其他 **Trident Protect helm** 图表设置

您可以自定义 AutoSupport 设置和命名空间筛选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	说明
autoSupport.proxy	string	为 NetApp AutoSupport 连接配置代理 URL。使用此选项可通过代理服务器路由支持包上传。示例：`http://my.proxy.url`。
autoSupport.insecure	布尔值	设置为 `true`时跳过 `AutoSupport` 代理连接的 `TLS` 验证。仅用于不安全的代理连接。（默认： `false`）
autoSupport.enabled	布尔值	启用或禁用每日 Trident Protect AutoSupport 捆绑包上传。设置为 `false`时，计划的每日上传将被禁用，但您仍然可以手动生成支持捆绑包。（默认值： `true`）
restoreSkipNamespaceAnnotations	string	要从备份和还原操作中排除的命名空间注释的逗号分隔列表。允许您根据注释筛选命名空间。
restoreSkipNamespaceLabels	string	要从备份和还原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. 创建配置文件并将其命名为 `values.yaml`。
2. 在创建的文件中，添加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. 使用正确的值填充 `values.yaml` 文件后，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. 使用以下带有 `--set` 标志的命令指定单个参数：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set autoSupport.enabled=false \
  --set autoSupport.proxy=http://my.proxy.url \
  --set-string
restoreSkipNamespaceAnnotations="{annotation1,annotation2}" \
  --set-string restoreSkipNamespaceLabels="{label1,label2}" \
  --reuse-values
```

将 **Trident Protect Pod** 限制到特定节点

您可以使用 Kubernetes nodeSelector 节点选择约束来根据节点标签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。

步骤

1. 创建名为 `nodeSelectorConfig.yaml` 的文件。
2. 将 nodeSelector 选项添加到文件中，并根据环境的需要修改文件以添加或更改节点标签以进行限制。例如

，以下文件包含默认操作系统限制，但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 `nodeSelectorConfig.yaml` 文件中的值：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

这会将默认限制替换为您在 `nodeSelectorConfig.yaml` 文件中指定的限制。

管理 Trident Protect

管理 Trident Protect 授权和访问控制

Trident Protect 使用基于角色的访问控制 (RBAC) 的 Kubernetes 模型。默认情况下，Trident Protect 提供单个系统命名空间及其关联的默认服务帐户。如果您的组织有很多用户或特定的安全需求，您可以使用 Trident Protect 的 RBAC 功能对资源和命名空间的访问进行更精细的控制。

集群管理员始终可以访问默认 `trident-protect`命名空间中的资源，也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，需要创建其他命名空间，并向这些命名空间中添加资源和应用程序。

请注意，没有用户可以在默认 `trident-protect`命名空间中创建应用程序数据管理 CR。您需要在应用程序命名空间中创建应用程序数据管理 CR（作为最佳实践，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）。

只有管理员应有权访问特权 Trident Protect 自定义资源对象，其中包括：

- **AppVault**：需要存储区凭据数据
- **AutoSupportBundle**：收集指标、日志和其他敏感 Trident Protect 数据
- **AutoSupportBundleSchedule**：管理日志收集计划

作为最佳做法，请使用 RBAC 来限制管理员对特权对象的访问。

有关 RBAC 如何规范对资源和命名空间的访问的详细信息，请参见 "Kubernetes RBAC 文档"。

有关服务帐户的详细信息，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，一个组织有一个集群管理员、一组工程用户和一组营销用户。集群管理员将完成以下任务，以创建一个环境，其中工程组和营销组各自只能访问分配给各自命名空间的资源。

步骤 **1**：创建一个命名空间以包含每个组的资源

创建命名空间可以在逻辑上分离资源，并更好地控制谁有权访问这些资源。

步骤

1. 为工程组创建命名空间：

```
kubectl create ns engineering-ns
```

2. 为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，以便将来在必要时可以在组之间进一步划分权限。

步骤

1. 为工程组创建服务帐户：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eng-user
  namespace: engineering-ns
```

2. 为营销组创建服务帐户：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mkt-user
  namespace: marketing-ns
```

步骤 **3**：为每个新服务帐户创建密码

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：创建一个 **RoleBinding** 对象以将此 **ClusterRole** 对象绑定到每个新服务帐户

当你安装 Trident Protect 时，会创建一个默认的 ClusterRole 对象。你可以通过创建并应用一个 RoleBinding 对象，将此 ClusterRole 绑定到服务账户。

步骤

1. 将 ClusterRole 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 ClusterRole 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
```

第 5 步：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营销资源：

```
kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
get applications.protect.trident.netapp.io -n marketing-ns
```

第 6 步：授予 AppVault 对象访问权限

要执行备份和快照等数据管理任务，集群管理员需要将 AppVault 对象的访问权限授予单个用户。

步骤

1. 创建并应用 AppVault 和密码组合 YAML 文件，该文件授予用户对 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 Role 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 RoleBinding CR 以将权限绑定到用户 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. 验证权限是否正确。

   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
   ```

   此时将显示与以下内容类似的输出：

```
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
```

此时将显示与以下内容类似的输出：

```
yes
```

结果

您授予 AppVault 权限的用户应能够使用授权的 AppVault 对象进行应用程序数据管理操作，并且不应能够访问分配的命名空间之外的任何资源或创建他们无权访问的新资源。

监控 Trident Protect 资源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 开源工具来监控受 Trident Protect 保护的资源的运行状况。

kube-state-metrics 服务从 Kubernetes API 通信中生成指标。将它与 Trident Protect 一起使用可提供有关环境中资源状态的有用信息。

Prometheus 是一个工具包，可以提取 kube-state-metrics 生成的数据，并将其呈现为有关这些对象的易读信息。kube-state-metrics 和 Prometheus 共同为您提供了一种方法，可以监控您使用 Trident Protect 管理的资源的运行状况和状态。

Alertmanager 是一项服务，可接收 Prometheus 等工具发送的警报，并将其路由到您配置的目标。

> 这些步骤中包含的配置和指导只是示例；您需要自定义它们以匹配您的环境。有关具体说明和支持，请参阅以下官方文档：
>
> - "kube-state-metrics 文档"
> - "Prometheus 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，需要安装和配置 kube-state-metrics、Prometheus 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 应用于集群：

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. 为 Helm 图表创建配置文件（例如， `metrics-config.yaml`）。您可以自定义以下示例配置以匹配您的环境：

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 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 "kube-state-metrics 自定义资源文档"中的说明配置 kube-state-metrics，为 Trident Protect 使用的自定义资源生成指标。

安装 Prometheus

您可以按照 "Prometheus 文档"中的说明安装 Prometheus。

安装 Alertmanager

您可以按照 "Alertmanager 文档"中的说明安装 Alertmanager。

步骤 2：配置监控工具以协同工作

安装监控工具后，需要将其配置为协同工作。

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 以将警报路由到 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，并向 Alertmanager 发送警报。您现在可以配置触发警报的条件以及应将警报发送到何处。

步骤 **3**：配置警报和警报目标

将工具配置为协同工作后，需要配置触发警报的信息类型以及应将警报发送到何处。

警报示例：备份失败

以下示例定义了在备份自定义资源的状态设置为 `Error` 5 秒或更长时间时触发的关键警报。您可以自定义此示例以匹配您的环境，并在 `prometheus.yaml`配置文件中包含此 YAML 代码段：

rules.yaml：为失败的备份定义 **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** 以向其他渠道发送警报

您可以通过在 `alertmanager.yaml` 文件中指定相应的配置，将 Alertmanager 配置为向其他渠道（如电子邮件、PagerDuty、Microsoft Teams 或其他通知服务）发送通知。

以下示例将 Alertmanager 配置为向 Slack 通道发送通知。要根据您的环境自定义此示例，请将 `api_url` 键的值替换为环境中使用的 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 使管理员能够生成捆绑包，其中包括对 NetApp Support 有用的信息，包括有关所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) 文件将支持捆绑包上传到 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（例如， `trident-protect-support-bundle.yaml`）。
2. 配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.triggerType**: (*Required*) 确定支持包是立即生成还是计划生成。计划包生成发生在 UTC 时间凌晨 12 点。可能的值：
     - 已计划
     - 手动
   - **spec.uploadEnabled**: (*Optional*) 控制是否应在生成支持捆绑包后将其上传到 NetApp 支持站点。如果未指定，则默认为 `false`。可能的值：
     - true
     - false（默认）
   - **spec.dataWindowStart**: (*Optional*) RFC 3339 格式的日期字符串，指定支持包中包含数据的窗口应开始的日期和时间。如果未指定，则默认为 24 小时前。您可以指定的最早窗口日期为 7 天前。

     示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-support-bundle.yaml`文件后，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，用环境中的信息替换括号中的值。 `trigger-type`确定是否立即创建捆绑包，或者创建时间是否由计划决定，并且可以是 `Manual`或 `Scheduled`。默认设置为 `Manual`。

   例如：

```
tridentctl-protect create autosupportbundle <my-bundle-name>
--trigger-type <trigger-type> -n trident-protect
```

监控和检索支持包

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. 将支持包检索到本地系统。从已完成的 AutoSupport 包中获取 copy 命令：

```
kubectl describe autosupportbundle trident-protect-support-bundle -n trident-protect
```

从输出中查找 `kubectl cp`命令并运行它，将 destination 参数替换为首选的本地目录。

升级 **Trident Protect**

您可以将 Trident Protect 升级到最新版本，以利用新功能或错误修复。

- 从版本 24.10 升级时，升级期间运行的快照可能会失败。此故障不会阻止创建未来的快照，无论是手动还是计划的快照。如果快照在升级过程中失败，您可以手动创建新的快照，以确保您的应用程序受到保护。

  为了避免潜在的故障，您可以在升级之前禁用所有快照计划，然后在升级后重新启用它们。但是，这会导致在升级期间丢失任何计划的快照。

- 对于私有注册表安装，请确保您的私有注册表中提供了目标版本所需的 Helm 图表和图像，并验证您的自定义 Helm 值与新图表版本兼容。有关详细信息，请参阅"从私有注册表安装 Trident Protect"。

要升级 Trident Protect，请执行以下步骤。

步骤

1. 更新 Trident Helm 存储库：

```
helm repo update
```

2. 升级 Trident Protect CRD：

> 如果您要从 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. 运行此命令可将 CRD 的管理从 `trident-protect-crds`切换到 `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. 运行此命令删除 `trident-protect-crds` 图表的 Helm 密钥：

> 请勿使用 Helm 卸载 `trident-protect-crds`图表，因为这可能会删除您的 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 Protect：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
--version 100.2510.0 --namespace trident-protect
```

> 您可以通过在 upgrade 命令中添加 `--set logLevel=debug`来配置升级过程中的日志级别。默认日志记录级别为 `warn`。建议将调试日志记录用于故障排除，因为它有助于 NetApp 支持诊断问题，而无需更改日志级别或重现问题。

管理和保护应用程序

使用 **Trident Protect AppVault** 对象管理存储桶

Trident Protect 的存储桶自定义资源 (CR) 称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含在保护操作（如备份、快照、还原操作和 SnapMirror 复制）中使用存储桶所需的配置。只有管理员可以创建 AppVaults。

在对应用程序执行数据保护操作时，您需要手动或从命令行创建 AppVault CR。AppVault CR 特定于您的环境，您可以在创建 AppVault CR 时使用此页面上的示例作为指南。

> 确保 AppVault CR 位于安装 Trident Protect 的集群上。如果 AppVault CR 不存在或您无法访问它，则命令行将显示错误。

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 和数据移动器可以通过提供商和任何相关资源进行身份验证。

Data Mover 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 secret。如果您未指定 secret，Trident Protect 将使用默认密码。

- 手动创建 AppVault CR 时，请使用 **spec.dataMoverPasswordSecretRef** 字段指定密码。
- 使用 Trident Protect CLI 创建 AppVault 对象时，请使用 `--data-mover-password-secret-ref` 参数指定密码。

创建数据移动器存储库密码机密

使用以下示例创建密码密钥。创建 AppVault 对象时，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。

- 根据您使用的数据移动器，您只需要包括该数据移动器的相应密码。例如，如果您正在使用 Restic 并且不打算将来使用 Kopia，则在创建密码时可以仅包含 Restic 密码。
- 请将密码保存在安全的地方。您将需要它来恢复同一个集群或另一个集群上的数据。如果集群或 `trident-protect`命名空间被删除，如果没有密码，您将无法还原备份或快照。

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

当您使用 Trident Protect 访问 S3 兼容存储（如 Amazon S3、Generic S3 "StorageGrid S3"或 "ONTAP S3"）时，您需要确保提供的用户凭据具有访问存储桶所需的权限。以下是授予使用 Trident Protect 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 策略的详细信息，请参见 "Amazon S3 文档" 中的示例。

EKS Pod Identity for Amazon S3 (AWS) 身份验证

Trident Protect 支持用于 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 凭据存储在 Kubernetes 密钥中。

EKS Pod Identity 与 **Trident Protect** 的要求

在将 EKS Pod Identity 与 Trident Protect 一起使用之前，请确保以下内容：

- 您的 EKS 集群已启用 Pod Identity。
- 您已创建具有必要 S3 存储桶权限的 IAM 角色。要了解更多信息，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 Identity 以及将 IAM 角色与服务帐户关联的详细说明，请参见 "AWS EKS Pod Identity 文档"。

AppVault 配置 使用 EKS Pod Identity 时，使用 `useIAM: true` 标志而不是显式凭据配置 AppVault CR：

```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
```

针对云提供商的 **AppVault** 密钥生成示例

定义 AppVault CR 时，需要包括访问提供程序托管的资源的凭据，除非您正在使用 IAM 身份验证。您为凭据生成密钥的方式将根据提供程序而有所不同。以下是多个提供程序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据创建密钥。

Google Cloud

```
kubectl create secret generic <secret-name> \
--from-file=credentials=<mycreds-file.json> \
-n trident-protect
```

Amazon 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

下面是每个提供程序的示例 AppVault 定义。

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 Kubernetes 机密，其中包含 Restic 和 Kopia 存储库加密的自定义密码。有关详细信息，请参见 Data Mover 存储库密码。
- 对于 Amazon S3 (AWS) AppVault 对象，您可以选择指定 sessionToken，这在使用单点登录 (SSO) 进行身份验证时非常有用。当您在 针对云提供商的 AppVault 密钥生成示例 中为提供商生成密钥时，将创建此令牌。
- 对于 S3 AppVault 对象，您可以选择使用 `spec.providerConfig.S3.proxyURL` 密钥为出站 S3 流量指定出口代理 URL。

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
```

Amazon 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 data mover 的 EKS 环境，你可以移除 `providerCredentials`部分，并将 `useIAM: true`添加到 `s3`配置下。

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** 示例

您可以使用以下 CLI 命令示例为每个提供程序创建 AppVault CR。

- 您可以选择指定 Kubernetes 机密，其中包含 Restic 和 Kopia 存储库加密的自定义密码。有关详细信息，请参见 Data Mover 存储库密码。
- 对于 S3 AppVault 对象，您可以选择使用 `--proxy-url <ip_address:port>` 参数为出站 S3 流量指定出口代理 URL。

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
```

Amazon 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
```

支持的 `**providerConfig.s3**`配置选项

有关 S3 提供程序配置选项，请参见下表：

参数	说明	默认	示例
`providerConfig.s3.skipCertValidation`	禁用 SSL/TLS 证书验证。	false	"true"、"false"
`providerConfig.s3.secure`	启用与 S3 端点的安全 HTTPS 通信。	true	"true"、"false"
`providerConfig.s3.proxyURL`	指定用于连接到 S3 的代理服务器的 URL。	无	http://proxy.example.com:8080
`providerConfig.s3.rootCA`	为 SSL/TLS 验证提供自定义根 CA 证书。	无	"CN=MyCustomCA"
`providerConfig.s3.useIAM`	启用 IAM 身份验证以访问 S3 存储桶。适用于 EKS Pod Identity。	false	true，false

查看 **AppVault** 信息

您可以使用 Trident Protect CLI 插件查看有关在集群上创建的 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。

> 在删除 AppVault 对象之前，请勿移除 AppVault CR 中的 `finalizers`密钥。如果这样做，可能会导致 AppVault 存储区中的剩余数据和集群中的孤立资源。

开始之前

确保已删除要删除的 AppVault 使用的所有快照和备份 CR。

使用 **Kubernetes CLI** 删除 **AppVault**

1. 删除 AppVault 对象，将 `appvault-name` 替换为要删除的 AppVault 对象的名称：

```
kubectl delete appvault <appvault-name> \
-n trident-protect
```

使用 **Trident Protect CLI** 删除 **AppVault**

1. 删除 AppVault 对象，将 `appvault-name` 替换为要删除的 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 和相关 AppVault CR 来定义要使用 Trident Protect 管理的应用程序。

创建 **AppVault CR**

您需要创建将在对应用程序执行数据保护操作时使用的 AppVault CR，并且 AppVault CR 需要驻留在安装 Trident Protect 的集群上。AppVault CR 特定于您的环境；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 来定义用于管理的应用程序。

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（例如， `maria-app.yaml`）。
   b. 配置以下属性：
      - **metadata.name**：（必需）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**: (*Required*) 使用命名空间和标签选择器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是此列表的一部分。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**: (*Optional*) 使用此属性指定要包含在应用程序定义中的群集范围的资源。此属性允许您根据其组、版本、种类和标签选择这些资源。
        - **groupVersionKind**：（必需）指定集群范围内资源的 API 组、版本和种类。
        - **labelSelector**：（可选）根据集群范围资源的标签对其进行筛选。
      - **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**：（*Optional*）此批注仅适用于从虚拟机定义的应用程序，例如在 KubeVirt 环境中，快照之前会发生文件系统冻结。指定此应用程序是否可以在快照期间写入文件系统。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，并在快照期间冻结文件系统。如果指定，但应用程序在应用程序定义中没有虚拟机，则忽略批注。如果未指定，则应用程序遵循 "全局 Trident Protect 冻结设置"。

如果需要在已创建应用程序后应用此批注，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"
```

+
示例 YAML：

+

```
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. （*Optional*）添加包含或排除标有特定标签的资源的筛选：

   - **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include`或 `Exclude`来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：

     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。

       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。
       - **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
       - **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段中的名称。

- **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。例如：`"trident.netapp.io/os=linux"`。

> 当同时使用 `resourceFilter`和 `labelSelector`时， `resourceFilter`首先运行，然后将 `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. 创建与环境匹配的应用程序 CR 后，应用 CR。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，使用环境中的信息替换括号中的值。可以使用逗号分隔的列表和示例中显示的参数在应用程序定义中包含命名空间和资源。

   在创建应用程序时，您可以选择使用注释来指定应用程序是否可以在快照期间写入文件系统。这仅适用于从虚拟机定义的应用程序，例如在 KubeVirt 环境中，在快照之前会发生文件系统冻结。如果将注释设置为 `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。如果将其设置为 `false`，应用程序将忽略全局设置，并在快照期间冻结文件系统。如果使用注释，但应用程序在应用程序定义中没有虚拟机，则该注释将被忽略。如果不使用注释，应用程序将遵循"全局 Trident Protect 冻结设置"。

   要在使用 CLI 创建应用程序时指定批注，可以使用 `--annotation`标志。

- 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文件系统冻结行为的本地应用程序设置：

```
tridentctl-protect create application <my_new_app_cr_name>
--namespaces <namespaces_to_include> --csr
<cluster_scoped_resources_to_include> --namespace <my-app-
namespace> --annotation protect.trident.netapp.io/skip-vm-freeze
=<"true"|"false">
```

您可以使用 `--resource-filter-include`和 `--resource-filter-exclude`标志来基于 `resourceSelectionCriteria`（如组、类型、版本、标签、名称和命名空间）包含或排除资源，如以下示例所示：

```
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

您可以通过使用自动保护策略或临时拍摄快照和备份来保护 Trident Protect 管理的所有应用。

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。"详细了解如何使用 Trident Protect 配置文件系统冻结"。

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显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：要快照的应用程序的 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）定义删除快照 CR 时快照的 AppArchive 会发生什么情况。这意味着即使设置为 `Retain`，快照也将被删除。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. 使用正确的值填充 `trident-protect-snapshot-cr.yaml`文件后，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，用环境中的信息替换括号中的值。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显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、快照或克隆中。

开始之前

确保 AWS 会话令牌过期时间足以支持任何长时间运行的 s3 备份操作。如果令牌在备份操作期间过期，操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
- 有关 AWS 资源凭据的详细信息，请参见 "AWS IAM 文档"。

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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **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**: (*Optional*) 一个字符串，指示要用于备份操作的备份工具。可能的值（区分大小写）：
     - `Restic`
     - `Kopia` (默认)
   - **spec.reclaimPolicy**：（可选）定义从声明中释放备份时会发生什么。可能的值：
     - `Delete`
     - `Retain` (默认)
   - **spec.snapshotRef**: (*Optional*): 要用作备份源的快照的名称。如果未提供，将创建并备份临时快照。

   示例 YAML：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  namespace: my-app-namespace
  name: my-cr-name
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  dataMover: Kopia
```

3. 使用正确的值填充 `trident-protect-backup-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-backup-cr.yaml
```

使用 **CLI** 创建备份

步骤

1. 创建备份，用环境中的信息替换括号中的值。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>
```

您可以选择使用 `--full-backup` 标志来指定备份是否应为非增量备份。默认情况下，所有备份都是增量备份。使用此标志时，备份将变为非增量备份。最佳做法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。

支持的备份注释

下表介绍了创建备份 CR 时可以使用的批注：

标注	类型	说明	默认值
protect.trident.netapp.io/full-backup	string	指定备份是否应为非增量备份。设置为 `true` 以创建非增量备份。最佳做法是定期执行完整备份，然后在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。	"false"
protect.trident.netapp.io/snapshot-completion-timeout	string	整个快照操作完成所允许的最长时间。	“60 分钟”
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	允许卷快照达到准备就绪状态的最长时间。	"30 分钟"
protect.trident.netapp.io/volume-snapshots-created-timeout	string	允许创建卷快照的最长时间。	"5 分钟"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	在操作失败之前，等待任何新创建的 PersistentVolumeClaims (PVC) 到达 `Bound` 阶段的最长时间（以秒为单位）。	"1200"（20 分钟）

创建数据保护计划

保护策略通过按定义的时间表创建快照、备份或同时创建快照和备份来保护应用。您可以选择每小时、每天、每周和每月创建快照和备份，还可以指定要保留的副本数。您可以使用 full-backup-rule 注释安排非增量完整备份。默认情况下，所有备份都是增量备份。定期执行完整备份以及之间的增量备份有助于降低与恢复相关的风险。

- 您只能通过将 `backupRetention`设置为零并将 `snapshotRetention`设置为大于零的值来创建快照计划。将 `snapshotRetention`设置为零意味着任何计划备份仍将创建快照，但这些快照是临时的，并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显式引用，或者它们引用了任何应用程序命名空间，则会包含在备份、快照或克隆中。

使用 **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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.dataMover**: (*Optional*) 一个字符串，指示要用于备份操作的备份工具。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**：（*Required*）要保留的备份数量。零表示不应创建备份（仅限快照）。
   - **spec.backupReclaimPolicy**：（可选）确定如果备份 CR 在其保留期间被删除，备份会发生什么情况。保留期限过后，始终会删除备份。可能的值（区分大小写）：
     - `Retain` (默认)
     - `Delete`
   - **spec.snapshotRetention**：（*Required*）要保留的快照数。零表示不应创建快照。
   - **spec.snapshotReclaimPolicy**：（可选）确定如果快照 CR 在其保留期间被删除，快照会发生什么情况。保留期过后，快照始终被删除。可能的值（区分大小写）：
     - `Retain`
     - `Delete` (默认)
   - **spec.granularity**：计划应运行的频率。可能的值以及必需的相关字段：
     - `Hourly` (要求您指定 `spec.minute`)
     - `Daily` （要求您指定 `spec.minute` 和 `spec.hour`）
     - `Weekly`（要求您指定 `spec.minute`, `spec.hour` 和 `spec.dayOfWeek`）
     - `Monthly`（要求您指定 `spec.minute`, `spec.hour` 和 `spec.dayOfMonth`）
     - `Custom`
   - **spec.dayOfMonth**：（*Optional*）应运行计划的月份日期（1 - 31）。如果粒度设置为 `Monthly`，则此字段为必填字段。必须以字符串形式提供该值。
   - **spec.dayOfWeek**：（*Optional*）应运行计划的星期几（0 - 7）。值 0 或 7 表示星期天。如果粒度设置为 `Weekly`，则此字段为必填字段。必须以字符串形式提供该值。
   - **spec.hour**：（*Optional*）应运行计划的一天中的小时（0 - 23）。如果粒度设置为 `Daily`、`Weekly`或 `Monthly`，则此字段为必填字段。必须以字符串形式提供该值。
   - **spec.minute**：（*Optional*）计划应运行的小时中的分钟数（0 - 59）。如果粒度设置为 `Hourly`、`Daily`、 `Weekly`或 `Monthly`，则此字段为必填字段。该值必须以字符串形式提供。

备份和快照计划的 YAML 示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-cr-name
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 YAML 示例：

```
---
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. 使用正确的值填充 `trident-protect-schedule-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

使用 **CLI** 创建计划

步骤

1. 创建保护计划，用环境中的信息替换括号中的值。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> \
  --backup-reclaim-policy <Retain|Delete (default Retain)> \
  --data-mover <Kopia_or_Restic> \
  --day-of-month <day_of_month_to_run_schedule> \
  --day-of-week <day_of_week_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> \
  --snapshot-reclaim-policy <Retain|Delete (default Delete)> \
  --full-backup-rule <string> \
  --run-immediately <true|false> \
  -n <application_namespace>
```

以下标记为您的日程安排提供了额外的控制：

- 完全备份计划：使用 `--full-backup-rule` 标志来计划非增量完全备份。此标志仅适用于 `--granularity Daily`。可能的值：
  - `Always`: 每天创建完整备份。
  - 特定工作日：指定一个或多个以逗号分隔的日期（例如， `"Monday,Thursday"`）。有效值：Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。

    此 `--full-backup-rule`标记不适用于每小时、每周或每月粒度。
- 仅快照计划：设置 `--backup-retention 0` 并为 `--snapshot-retention` 指定一个大于零的值。

支持的计划注释

下表介绍了创建计划 CR 时可以使用的批注：

标注	类型	说明	默认值
protect.trident.netapp.io/full-backup-rule	string	指定计划完整备份的规则。您可以将其设置为 `Always`以进行持续完整备份，也可以根据您的要求进行自定义。例如，如果选择每日粒度，则可以指定应进行完整备份的工作日（例如， `"Monday,Thursday"`）。有效的工作日值为：Monday、Tuesday、Wednesday、Thursday、Friday、Saturday、Sunday。请注意，此批注只能用于将 `granularity`设置为 `Daily`的计划。	未设置（所有备份都是增量备份）
protect.trident.netapp.io/snapshot-completion-timeout	string	整个快照操作完成所允许的最长时间。	“60 分钟”
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	允许卷快照达到准备就绪状态的最长时间。	"30 分钟"
protect.trident.netapp.io/volume-snapshots-created-timeout	string	允许创建卷快照的最长时间。	"5 分钟"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	在操作失败之前，等待任何新创建的 PersistentVolumeClaims (PVC) 到达 `Bound`阶段的最长时间（以秒为单位）。	"1200"（20 分钟）

删除快照

删除不再需要的计划快照或按需快照。

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。

确保将回收策略设置为 `Delete` 以从对象存储中删除所有备份数据。策略的默认设置是 `Retain` 以避免意外数据丢失。如果策略未更改为 `Delete`，则备份数据将保留在对象存储中，并需要手动删除。

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. 使用以下命令可检索备份操作的状态，用环境中的信息替换括号中的值：

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

为 **azure-netapp-files (ANF)** 操作启用备份和还原

如果您已安装 Trident Protect，则可以为使用 azure-netapp-files 存储类并在 Trident 24.06 之前创建的存储后端启用节省空间的备份和恢复功能。此功能适用于 NFSv4 卷，不会消耗容量池中的额外空间。

开始之前

确保以下内容：

- 您已安装 Trident Protect。
- 您已在 Trident Protect 中定义了一个应用程序。在完成此过程之前，此应用程序将具有有限的保护功能。
- 您已选择 `azure-netapp-files` 作为存储后端的默认存储类。

展开以查看配置步骤

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 并与应用程序关联的 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
   ```

   响应：

   ```
   snapshotDirectory: "true"
   ```

   +
   未启用快照目录时，Trident Protect 会选择常规备份功能，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可用于创建要备份的卷大小的临时卷。

结果

该应用程序可以使用 Trident Protect 进行备份和还原。每个 PVC 也可供其他应用程序用于备份和还原。

还原应用程序

使用 **Trident Protect** 还原应用程序

您可以使用 Trident Protect 从快照或备份恢复您的应用程序。将应用程序还原到同一集群时，从现有快照还原将更快。

- 还原应用程序时，为应用程序配置的所有执行挂钩都会随应用程序一起还原。如果存在还原后执行挂钩，它将作为还原操作的一部分自动运行。
- qtree 卷支持从备份还原到其他命名空间或原始命名空间。但是，qtree 卷不支持从快照还原到其他命名空间或原始命名空间。
- 您可以使用高级设置自定义还原操作。要了解更多信息，请参阅 "使用高级 Trident Protect 还原设置"。

从备份还原到其他命名空间

使用 BackupRestore CR 将备份还原到其他命名空间时，Trident Protect 会在新命名空间中还原应用程序，并为还原的应用程序创建应用程序 CR。要保护还原的应用程序，请创建按需备份或快照，或建立保护计划。

- 使用现有资源将备份还原到其他命名空间不会更改与备份中的名称共享的任何资源。要还原备份中的所有资源，请删除并重新创建目标命名空间，或将备份还原到新命名空间。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
- 有关 AWS 资源凭据的详细信息，请参见 "AWS IAM 文档"。

使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 "Kopia 文档"。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 `tridentctl-protect create --help`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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appArchivePath**：AppVault 中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**：还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`和 `my-destination-namespace`。

     ```
     ---
     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）如果需要仅选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：

   > Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了永久卷声明资源，并且它具有关联的 pod，则 Trident Protect 也将还原关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include`或 `Exclude`来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。
       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。

- **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
- **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段中的名称。
- **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-backup-restore-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用 CLI

步骤

1. 将备份还原到其他命名空间，用环境中的信息替换括号中的值。 `namespace-mapping` 参数使用冒号分隔的命名空间将源命名空间映射到格式为 `source1:dest1,source2:dest2` 的正确目标命名空间。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>
```

您可以随时将备份还原到原始命名空间。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
- 有关 AWS 资源凭据的详细信息，请参见 "AWS IAM 文档"。

使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 "Kopia 文档"。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 `tridentctl-protect create --help`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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appArchivePath**：AppVault 中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）如果需要仅选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：

   Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了永久卷声明资源，并且它具有关联的 pod，则 Trident Protect 也将还原关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include`或 `Exclude`来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。
       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。
       - **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
       - **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段

中的名称。

- **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-backup-ipr-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

使用 **CLI**

步骤

1. 将备份还原到原始命名空间，用环境中的信息替换括号中的值。 `backup` 参数使用格式为 `<namespace>/<name>` 的命名空间和备份名称。例如：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

如果原始群集出现问题，可以将备份还原到其他群集。

- 使用 Kopia 作为数据移动器还原备份时，可以选择在 CR 中指定注释或使用 CLI 控制 Kopia 使用的临时存储的行为。有关可以配置的选项的详细信息，请参见 "Kopia 文档"。有关使用 Trident Protect CLI 指定注释的详细信息，请使用 `tridentctl-protect create --help`命令。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

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 存储桶路径，备份存储在该路径中。
- 运行 `tridentctl-protect get appvaultcontent` 命令时，确保您的本地环境可以连接到 AppVault CR 中定义的对象存储桶。如果网络限制阻止访问，请在目标集群的 pod 内运行 Trident Protect CLI。
- 确保 AWS 会话令牌过期时间足以进行任何长期运行的还原操作。如果令牌在还原操作期间过期，则操作可能会失败。
  - 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
  - 有关 AWS 资源凭据的详细信息，请参见 "AWS 文档"。

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>
```

确保目标集群上存在用于应用程序还原的命名空间。

2. 从目标集群查看可用的 AppVault 的备份内容：

```
tridentctl-protect get appvaultcontent <appvault_name> \
--show-resources backup \
--show-paths \
--context <destination_cluster_name>
```

运行此命令会显示 AppVault 中的可用备份，包括其原始群集、相应的应用程序名称、时间戳和存档路径。

输出示例：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   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 名称和存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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：（必需）存储备份内容的 AppVault 的名称。
   - **spec.appArchivePath**：AppVault 中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 中提到的命令查看备份内容。

   - **spec.namespaceMapping**：还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`和 `my-destination-namespace`。

     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. 使用正确的值填充 `trident-protect-backup-restore-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用 **CLI**

1. 使用以下命令还原应用程序，用环境中的信息替换括号中的值。namespace-mapping 参数使用冒号分隔的命名空间将源命名空间映射到格式为 source1:dest1,source2:dest2 的正确目标命名空间。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>
```

从快照还原到其他命名空间

您可以使用自定义资源 (CR) 文件将数据从快照还原到其他命名空间或原始源命名空间。使用 SnapshotRestore CR 将快照还原到其他命名空间时，Trident Protect 会在新命名空间中还原应用程序，并为还原的应用程序创建应用程序 CR。要保护还原的应用程序，请创建按需备份或快照，或建立保护计划。

- SnapshotRestore 支持 `spec.storageClassMapping`属性，但仅当源和目标存储类使用相同的存储后端时。如果尝试还原到使用不同存储后端的 `StorageClass`，还原操作将失败。
- 使用 CR 还原到新命名空间时，您必须在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
- 有关 AWS 资源凭据的详细信息，请参见 "AWS IAM 文档"。

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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：（必需）存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**：AppVault 中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**：还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`和 `my-destination-namespace`。

     ```
     ---
     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）如果需要仅选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：

   > Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了永久卷声明资源，并且它具有关联的 pod，则 Trident Protect 也将还原关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include`或 `Exclude`来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。
       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。

- **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
- **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段中的名称。
- **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。
- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-snapshot-restore-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用 CLI

步骤

1. 将快照还原到其他命名空间，用环境中的信息替换括号中的值。
   - `snapshot` 参数使用格式为 `<namespace>/<name>` 的命名空间和快照名称。
   - `namespace-mapping` 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 snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
-n <application_namespace>
```

从快照还原到原始命名空间

您可以随时将快照还原到原始命名空间。

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，则快照还原操作（到原始命名空间和新命名空间）将无法正常工作。所有还原的卷将具有相同的数据，而不是每个卷的正确数据。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版或更高版本。

开始之前

确保 AWS 会话令牌过期时间足以进行任何长期运行的 s3 还原操作。如果令牌在还原操作期间过期，则操作可能会失败。

- 有关检查当前会话令牌过期的详细信息，请参见 "AWS API 文档"。
- 有关 AWS 资源凭据的详细信息，请参见 "AWS IAM 文档"。

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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：（必需）存储快照内容的 AppVault 的名称。
   - **spec.appArchivePath**：AppVault 中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）如果需要仅选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：

> Trident Protect 会自动选择一些资源，因为它们与您选择的资源之间存在关系。例如，如果您选择了永久卷声明资源，并且它具有关联的 pod，则 Trident Protect 也将还原关联的 pod。

   - **resourceFilter.resourceSelectionCriteria**：（筛选时需要）使用 `Include`或 `Exclude`来包含或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。
       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。
       - **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
       - **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-snapshot-ipr-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-ipr-cr.yaml
```

使用 CLI

步骤

1. 将快照还原到原始命名空间，用环境中的信息替换括号中的值。例如：

```
tridentctl-protect create snapshotinplacerestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
-n <application_namespace>
```

检查还原操作的状态

您可以使用命令行检查正在进行、已完成或已失败的还原操作的状态。

步骤

1. 使用以下命令可检索还原操作的状态，用环境中的信息替换括号中的值：

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

使用高级 **Trident Protect** 还原设置

您可以使用高级设置（如注释、命名空间设置和存储选项）自定义还原操作，以满足您的特定要求。

还原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匹配。将添加目标命名空间中不存在的源命名空间中的标签或注释，并覆盖已存在的任何标签或注释，以匹配源命名空间中的值。仅存在于目标命名空间上的标签或注释保持不变。

> 如果使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的关键作用。命名空间注释可确保还原的 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。有关详细信息，请参见"OpenShift 安全上下文约束文档"。

在执行还原或故障转移操作之前，可以通过设置 Kubernetes 环境变量 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`来防止目标命名空间中的特定注释被覆盖。例如：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

> 执行还原或故障切换操作时，在 `restoreSkipNamespaceAnnotations`和 `restoreSkipNamespaceLabels`中指定的任何命名空间批注和标签都将从还原或故障切换操作中排除。确保在初始 Helm 安装过程中配置了这些设置。要了解更多信息，请参阅 "配置其他 Trident Protect helm 图表设置"。

如果使用带有 `--create-namespace` 标志的 Helm 安装源应用程序，则会对 `name` 标签键进行特殊处理。在还原或故障转移过程中，Trident Protect 会将此标签复制到目标命名空间，但如果来自源的值与源命名空间匹配，则将值更新为目标命名空间值。如果此值与源命名空间不匹配，则将其复制到目标命名空间，不进行任何更改。

示例

以下示例显示了源和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作之前和之后的目标命名空间的状态，以及如何在目标命名空间中组合或覆盖注释和标签。

还原或故障转移操作之前

下表显示了还原或故障转移操作之前的示例源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"	• environment=production • 合规性=hipaa • name=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

还原操作后

下表显示了还原或故障转移操作后示例目标命名空间的状态。已添加一些键，一些键已被覆盖，并且已更新 `name` 标签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"	• name=ns-2 • 合规性=hipaa • environment=production • 角色=数据库

支持的字段

本节描述可用于还原操作的其他字段。

存储类映射

该 `spec.storageClassMapping` 属性定义从源应用程序中存在的存储类到目标集群上的新存储类的映射。您可以在具有不同存储类的集群之间迁移应用程序或更改 BackupRestore 操作的存储后端时使用此功能。

示例：

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

支持的注释

本节列出了用于在系统中配置各种行为的支持注释。如果用户未明确设置注释，系统将使用默认值。

标注	类型	说明	默认值
protect.trident.netapp.io/data-mover-timeout-sec	string	允许数据移动器操作停止的最长时间（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"
protect.trident.netapp.io/pvc-bind-timeout-sec	string	在操作失败之前，等待任何新创建的 PersistentVolumeClaims (PVC) 到达 `Bound`阶段的最长时间（以秒为单位）。适用于所有还原 CR 类型（BackupRestore、BackupInplaceRestore、SnapshotRestore、SnapshotInplaceRestore）。如果您的存储后端或集群通常需要更多时间，请使用更高的值。	"1200"（20 分钟）

使用 **NetApp SnapMirror** 和 **Trident Protect** 复制应用程序

使用 Trident Protect，您可以使用 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，在同一群集上或在不同群集之间。

还原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匹配。将添加目标命名空间中不存在的源命名空间中的标签或注释，并覆盖已存在的任何标签或注释，以匹配源命名空间中的值。仅存在于目标命名空间上的标签或注释保持不变。

> 如果使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的关键作用。命名空间注释可确保还原的 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。有关详细信息，请参见"OpenShift 安全上下文约束文档"。

在执行还原或故障转移操作之前，可以通过设置 Kubernetes 环境变量 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`来防止目标命名空间中的特定注释被覆盖。例如：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set-string
restoreSkipNamespaceAnnotations="{<annotation_key_to_skip_1>,<annotation_k
ey_to_skip_2>}" \
  --reuse-values
```

> 执行还原或故障切换操作时，在 `restoreSkipNamespaceAnnotations`和 `restoreSkipNamespaceLabels`中指定的任何命名空间批注和标签都将从还原或故障切换操作中排除。确保在初始 Helm 安装过程中配置了这些设置。要了解更多信息，请参阅 "配置其他 Trident Protect helm 图表设置"。

如果使用带有 `--create-namespace` 标志的 Helm 安装源应用程序，则会对 `name` 标签键进行特殊处理。在还原或故障转移过程中，Trident Protect 会将此标签复制到目标命名空间，但如果来自源的值与源命名空间匹配，则将值更新为目标命名空间值。如果此值与源命名空间不匹配，则将其复制到目标命名空间，不进行任何更改。

示例

以下示例显示了源和目标命名空间，每个命名空间都有不同的注释和标签。您可以查看操作之前和之后的目标命名空间的状态，以及如何在目标命名空间中组合或覆盖注释和标签。

还原或故障转移操作之前

下表显示了还原或故障转移操作之前的示例源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"	• environment=production • 合规性=hipaa • name=ns-1
命名空间 ns-2（目标）	• annotation.one/key: "true" • annotation.three/key: "false"	• 角色=数据库

还原操作后

下表显示了还原或故障转移操作后示例目标命名空间的状态。已添加一些键，一些键已被覆盖，并且已更新 `name` 标签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"	• name=ns-2 • 合规性=hipaa • environment=production • 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。"详细了解如何使用 Trident Protect 配置文件系统冻结"。

在故障转移和反向操作期间执行挂钩

在使用 AppMirror 关系来保护您的应用程序时，在故障转移和反向操作期间，您应该注意与执行挂钩相关的特定行为。

- 在故障转移过程中，执行挂钩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，执行挂钩出现在应用程序上，并将在任何相关操作期间执行。
- 在反向或反向重新同步期间，应用程序上的任何现有执行挂钩都将被删除。当源应用程序成为目标应用程序时，这些执行挂钩无效并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 执行挂钩"。

设置复制关系

设置复制关系包括以下内容：

- 选择您希望 Trident Protect 拍摄应用快照的频率（包括应用程序的 Kubernetes 资源以及每个应用程序卷的卷快照）
- 选择复制计划（包括 Kubernetes 资源和持久卷数据）
- 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 AppVault。根据您的存储提供商，修改 "AppVault 自定义资源" 中的示例以适应您的环境：

使用 **CR** 创建 **AppVault**

a. 创建自定义资源 (CR) 文件并将其命名（例如， `trident-protect-appvault-primary-source.yaml`）。

b. 配置以下属性：

- **metadata.name**: (*Required*) AppVault 自定义资源的名称。记下您选择的名称，因为复制关系所需的其他 CR 文件引用此值。
- **spec.providerConfig**：（必需）存储使用指定提供程序访问 AppVault 所需的配置。为您的提供程序选择 bucketName 和任何其他必要的详细信息。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。有关其他提供程序的 AppVault CR 示例，请参阅 "AppVault 自定义资源"。
- **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*) 指示凭据值应来自机密。
    - **key**：（*Required*）要从中选择的 secret 的有效 key。
    - **name**：（*Required*）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。**name** 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。
- **spec.providerType**：（必需）确定提供备份的内容；例如，NetApp ONTAP S3、通用 S3 、Google Cloud 或 Microsoft Azure。可能的值：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. 使用正确的值填充 `trident-protect-appvault-primary-source.yaml`文件后，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，用环境中的信息替换括号中的值：

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. 在源群集上，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（例如， `trident-protect-app-source.yaml`）。

b. 配置以下属性：

- **metadata.name**：（必需）应用程序自定义资源的名称。记下您选择的名称，因为复制关系所需的其他 CR 文件引用此值。
- **spec.includedNamespaces**：（必需）命名空间和相关标签的数组。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是此数组的一部分。

  示例 **YAML**：

```
---
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. 使用正确的值填充 `trident-protect-app-source.yaml`文件后，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，拍摄源应用程序的快照。此快照用作目标集群上应用程序的基础。如果跳过此步骤，则需要等待下一个计划快照运行，以便获得最近的快照。要创建按需快照，请参阅 "创建按需快照"。
4. 在源群集上，创建复制计划 CR：

除了下面提供的时间表外，建议创建一个单独的每日快照时间表，保留期为 7 天，以维护对等 ONTAP 群集之间的共同快照。这可确保快照最多可用 7 天，但可以根据用户要求自定义保留期。

如果发生故障转移，系统可以使用这些快照最多 7 天进行反向操作。这种方法使反向过程更快、更高效，因为只会传输自上次快照以来所做的更改，而不会传输所有数据。

如果应用程序的现有计划已满足所需的保留要求，则不需要其他计划。

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（例如， `trident-protect-schedule.yaml`）。

ii. 配置以下属性：

- **metadata.name**：（*Required*）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*）此字段为必填字段，必须将值设置为 0。
- **spec.enabled**：必须设置为 true。
- **spec.granularity**：必须设置为 `Custom`。
- **spec.recurrenceRule**：以 UTC 时间定义开始日期和重复间隔。
- **spec.snapshotRetention**：必须设置为 2。

示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-schedule.yaml`文件后，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，用环境中的信息替换括号中的值：

```
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>
```

示例：

```
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. 在目标群集上，创建与在源群集上应用的 AppVault CR 相同的源应用程序 AppVault CR，并将其命名（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 "AppVault 自定义资源" 中的示例以适应您的环境：
   a. 创建自定义资源 (CR) 文件并将其命名（例如， `trident-protect-appvault-secondary-destination.yaml`）。
   b. 配置以下属性：
      - **metadata.name**: (*Required*) AppVault 自定义资源的名称。记下您选择的名称，因为复制关系所需的其他 CR 文件引用此值。
      - **spec.providerConfig**：（必需）存储使用指定提供程序访问 AppVault 所需的配置。为您的提供程序选择 `bucketName` 和任何其他必要的详细信息。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。有关其他提供程序的 AppVault CR 示例，请参阅 "AppVault 自定义资源"。
      - **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*) 指示凭据值应来自机密。
          - **key**：（*Required*）要从中选择的 secret 的有效 key。
          - **name**：（*Required*）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。**name** 应与 **spec.providerCredentials.valueFromSecret.name** 匹配。

- **spec.providerType**：（必需）确定提供备份的内容；例如，NetApp ONTAP S3、通用 S3 、Google Cloud 或 Microsoft Azure。可能的值：
  - aws
  - azure
  - gcp
  - generic-s3
  - ontap-s3
  - storagegrid-s3

c. 使用正确的值填充 `trident-protect-appvault-secondary-destination.yaml`文件后，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 时，请在应用 CR 之前手动创建目标命名空间。Trident Protect 仅在使用 CLI 时自动创建命名空间。

使用 **CR** 创建 **AppMirrorRelationship**

a. 创建自定义资源 (CR) 文件并将其命名（例如， `trident-protect-relationship.yaml`）。

b. 配置以下属性：

- **metadata.name:** (必需) AppMirrorRelationship 自定义资源的名称。
- **spec.destinationAppVaultRef**: (*Required*) 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**: (*Required*) 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）选择集群上有效存储类的名称。存储类必须链接到与源环境对等的 ONTAP 存储虚拟机。如果未提供存储类，则默认情况下将使用集群上的默认存储类。
- **spec.recurrenceRule**：以 UTC 时间定义开始日期和重复间隔。

示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-relationship.yaml`文件后，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，用环境中的信息替换括号中的值：

```
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>
```

示例：

```
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和状态：

```
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。此过程停止复制关系，并使应用程序在目标集群上联机。Trident Protect 不会停止源集群上的应用程序（如果它运行正常）。

步骤

1. 在目标群集上，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）创建故障转移应用程序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
```

重新同步故障转移复制关系

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。

此过程在重新建立复制之前停止目标集群上的应用。

在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。这可确保捕获源集群的最新更改。
2. 在目标群集上，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。任何剩余的计划都会导致卷快照失败。

反向重新同步故障转移的复制关系

当您反向重新同步故障转移复制关系时，目标应用程序将成为源应用程序，而源应用程序将成为目标。在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。这会导致目标成为源。如果新目标集群上还有任何保护计划，请将其删除。
2. 通过应用最初用于设置关系的 CR 文件到相反的集群来设置复制关系。
3. 确保新目标（原始源集群）配置了两个 AppVault CR。
4. 在相反的集群上设置复制关系，配置相反方向的值。

反向应用程序复制方向

当您反向复制方向时，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，然后再故障切换到目标应用程序。

在这种情况下，您将交换源和目标。

步骤

1. 在源集群上，创建关闭快照：

使用 **CR** 创建关闭快照

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（例如， `trident-protect-shutdownsnapshot.yaml`）。

ii. 配置以下属性：

- **metadata.name**：（*Required*）自定义资源的名称。
- **spec.AppVaultRef**: (*Required*) 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。

示例 YAML：

```
---
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. 使用正确的值填充 `trident-protect-shutdownsnapshot.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-namespace
```

使用 **CLI** 创建关闭快照

a. 创建关机快照，用环境中的信息替换括号中的值。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，关闭快照完成后，获取关闭快照的状态：

```
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（也称为基准名；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. 执行从新目标集群到新源集群的故障切换，并进行以下更改：

> 在故障转移过程的第 2 步中，将 `spec.promotedSnapshot` 字段包含在 AppMirrorRelationship CR 文件中，并将其值设置为您在上述第 3 步中记录的基准名。

5. 执行 [反向重新同步故障转移的复制关系] 中的反向重新同步步骤。
6. 在新源集群上启用保护计划。

结果

由于反向复制，发生以下操作：

- 为初始源应用的 Kubernetes 资源拍摄一个快照。
- 通过删除应用程序的 Kubernetes 资源（将 PVC 和 PV 保留到位），可以优雅地停止原始源应用程序的 Pod。
- 关闭 Pod 后，将拍摄并复制应用卷的快照。
- SnapMirror 关系已中断，使目标卷准备好读/写。
- 使用原始源应用程序关闭后复制的卷数据，从预关闭快照中还原应用程序的 Kubernetes 资源。
- 反向重新建立复制。

将应用程序故障回切至原始源群集

使用 Trident Protect，您可以使用以下操作序列在故障转移操作后实现"回切"。在此工作流程中，为了还原原始复制方向，Trident Protect 会在反转复制方向之前将任何应用程序更改复制（重新同步）回原始源应用程序。

此过程从已完成故障转移到目标的关系开始，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。

> 请勿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 [反向重新同步故障转移的复制关系] 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。当您删除应用程序复制关系时，会产生两个独立的应用程序，它们之间没有关系。

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迁移应用程序或迁移到不同的存储类。

迁移应用程序时，为应用程序配置的所有执行挂钩都会随应用程序一起迁移。如果存在还原后执行挂钩，它将作为还原操作的一部分自动运行。

备份和还原操作

要为以下方案执行备份和还原操作，可以自动执行特定的备份和还原任务。

克隆到同一集群

要将应用程序克隆到同一集群，请创建快照或备份并将数据恢复到同一集群。

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. "从快照还原数据"。
   b. "从备份还原数据"。

克隆到不同的集群

若要将应用程序克隆到其他群集（执行跨群集克隆），请在源群集上创建备份，然后将备份还原到其他群集。确保已在目标集群上安装 Trident Protect。

您可以使用 "SnapMirror 复制" 在不同群集之间复制应用程序。

步骤

1. "创建备份"。
2. 确保已在目标集群上配置了包含备份的对象存储桶的 AppVault CR。
3. 在目标集群上，"从备份还原数据"。

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其他存储类。

例如（从还原 CR 中排除机密）：

```
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** 还原快照

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. 在创建的文件中，配置以下属性：
   - **metadata.name**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.appArchivePath**：AppVault 中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**：还原操作的源命名空间到目标命名空间的映射。使用环境中的信息替换 `my-source-namespace`和 `my-destination-namespace`。

     ```
     ---
     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. 或者，如果只需要选择要还原的应用程序的某些资源，请添加包含或排除标有特定标签的资源的筛选：
   - **resourceFilter.resourceSelectionCriteria**：（筛选必需）使用 `include or exclude` 来包括或排除在 resourceMatchers 中定义的资源。添加以下 resourceMatchers 参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatchers**：resourceMatcher 对象数组。如果在此数组中定义多个元素，则它们将作为 OR 操作进行匹配，并且每个元素（组、种类、版本）内的字段将作为 AND 操作进行匹配。
       - **resourceMatchers[].group**：（*Optional*）要筛选的资源的组。
       - **resourceMatchers[].kind**：（*Optional*）要筛选的资源的类型。
       - **resourceMatchers[].version**：（*Optional*）要筛选的资源的版本。
       - **resourceMatchers[].names**：（可选）要过滤的资源的 Kubernetes metadata.name 字段中的名称。
       - **resourceMatchers[].namespaces**: (*Optional*) 要过滤的资源的 Kubernetes metadata.name 字段中的命名空间。

- **resourceMatchers[].labelSelectors**: (*Optional*) 资源的 Kubernetes metadata.name 字段中的标签选择器字符串，如 "Kubernetes 文档" 中所定义。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. 使用正确的值填充 `trident-protect-snapshot-restore-cr.yaml`文件后，应用 CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用 **CLI** 还原快照

步骤

1. 将快照还原到其他命名空间，用环境中的信息替换括号中的值。

   - `snapshot` 参数使用格式为 `<namespace>/<name>` 的命名空间和快照名称。
   - `namespace-mapping` 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 snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

管理 Trident Protect 执行挂钩

执行挂钩是一种自定义操作，您可以将其配置为与托管应用的数据保护操作一起运行。例如，如果您有数据库应用程序，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。这可确保应用程序一致性快照。

执行挂钩的类型

Trident Protect 支持以下类型的执行挂钩，具体取决于它们可以运行的时间：

- 快照前
- 快照后
- 备份前
- 备份后
- 恢复后
- 故障转移后

执行顺序

运行数据保护操作时，执行挂钩事件按以下顺序发生：

1. 任何适用的自定义操作前执行钩子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操作前钩子，但操作之前这些钩子的执行顺序既不保证也不可配置。
2. 如果适用，文件系统会冻结。"详细了解如何使用 Trident Protect 配置文件系统冻结"。
3. 执行数据保护操作。
4. 已冻结的文件系统已解冻（如适用）。
5. 任何适用的自定义操作后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尽可能多的自定义操作后挂钩，但操作后这些挂钩的执行顺序既不保证也不可配置。

如果创建相同类型的多个执行挂钩（例如，pre-snapshot），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. 已执行快照前挂钩
2. 已执行快照后挂钩
3. 已执行备份前挂钩
4. 已执行备份后挂钩

前面的顺序示例仅适用于运行不使用现有 snapshot 的备份时。

在生产环境中启用执行挂钩脚本之前，应始终对其进行测试。您可以使用 'kubectl exec' 命令来方便地测试脚本。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是一致的。为此，可以将应用克隆到临时命名空间，还原快照或备份，然后测试应用。

如果快照前执行挂钩添加、更改或删除 Kubernetes 资源，这些更改将包含在快照或备份以及任何后续还原操作中。

关于自定义执行挂钩的重要说明

为应用规划执行挂钩时，请考虑以下内容。

- 执行挂钩必须使用脚本来执行操作。许多执行挂钩可以引用相同的脚本。
- Trident Protect 要求执行挂钩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 使用执行挂钩设置和任何匹配标准来确定哪些挂钩适用于快照、备份或还原操作。

由于执行挂钩通常会减少或完全禁用它们所运行的应用程序的功能，因此应始终尝试最大限度地减少自定义执行挂钩的运行时间。如果使用关联的执行挂钩启动备份或快照操作，但随后取消该操作，则如果备份或快照操作已经开始，则仍允许运行挂钩。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。

执行挂钩筛选器

在为应用程序添加或编辑执行挂钩时，可以向执行挂钩添加筛选器，以管理挂钩将匹配的容器。筛选器对于在所有容器上使用相同容器映像的应用程序非常有用，但可能会将每个映像用于不同的目的（例如 Elasticsearch ）。筛选器允许您创建执行挂钩在某些但不一定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筛选器，它们将与逻辑 AND 运算符组合。每个执行挂钩最多可以有 10 个活动筛选器。

添加到执行挂钩的每个筛选器都使用正则表达式来匹配集群中的容器。当挂钩与容器匹配时，挂钩将在该容器上运行其关联的脚本。筛选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筛选器。有关 Trident Protect 在执行挂钩筛选器中支持正则表达式的语法的信息，请参阅 "正则表达式 2 (RE2) 语法支持"。

如果将命名空间筛选器添加到在还原或克隆操作后运行的执行挂钩中，并且还原或克隆源和目标在不同命名空间中，则命名空间筛选器仅应用于目标命名空间。

执行挂钩示例

访问 "NetApp Verda GitHub 项目"以下载流行应用程序（如 Apache Cassandra 和 Elasticsearch）的实际执行挂钩。您还可以查看示例并获得构建自己的自定义执行挂钩的想法。

创建执行挂钩

您可以使用 Trident Protect 为应用程序创建自定义执行挂钩。您需要拥有所有者、管理员或成员权限才能创建执行挂钩。

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**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**: (*Required*) 要为其运行执行挂钩的应用程序的 Kubernetes 名称。
   - **spec.stage**：（*Required*）指示执行挂钩应在操作期间运行哪个阶段的字符串。可能的值：
     - 之前
     - 发布
   - **spec.action**：（*Required*）指示执行挂钩将采取哪些操作的字符串，假设指定的任何执行挂钩过滤器都匹配。可能的值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.enabled**：（*Optional*）指示是启用还是禁用此执行挂钩。如果未指定，则默认值为 true。
   - **spec.hookSource**: (*Required*) 包含 base64 编码的钩子脚本的字符串。
   - **spec.timeout**：（*Optional*）一个数字，定义允许执行挂钩运行的时间，以分钟为单位。最小值为 1 分钟，如果未指定，默认值为 25 分钟。
   - **spec.arguments**：（*Optional*）可以为执行挂钩指定的参数的 YAML 列表。
   - **spec.matchingCriteria**：（可选）条件键值对的可选列表，每对组成一个执行挂钩过滤器。每个执行挂钩最多可添加 10 个筛选器。
   - **spec.matchingCriteria.type**: (*Optional*) 标识执行挂钩筛选器类型的字符串。可能的值：
     - ContainerImage
     - ContainerName
     - PodName
     - PodLabel
     - NamespaceName
   - **spec.matchingCriteria.value**: (*Optional*) 标识执行挂钩过滤器值的字符串或正则表达式。

     示例 YAML：

```
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. 使用正确的值填充 CR 文件后，应用 CR：

```
kubectl apply -f trident-protect-hook.yaml
```

使用 **CLI**

步骤

1. 创建执行挂钩，用环境中的信息替换括号中的值。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>
```

手动运行执行挂钩

您可以出于测试目的手动运行执行挂钩，也可以在出现故障后手动重新运行挂钩。您需要拥有所有者、管理员或成员权限才能手动运行执行挂钩。

手动运行执行挂钩包括两个基本步骤：

1. 创建资源备份，收集资源并创建资源的备份，确定挂钩将在何处运行
2. 针对备份运行执行挂钩

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**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **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 中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}'
     ```

     示例 YAML：

     ```
     ---
     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. 使用正确的值填充 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，用环境中的信息替换括号中的值。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. 查看备份的状态。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. 验证备份是否成功：

```
kubectl describe resourcebackup <my_backup_name>
```

步骤 **2**：运行执行挂钩

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**：（*Required*）此自定义资源的名称；为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**: (*Required*) 确保此值与您在步骤 1 中创建的 ResourceBackup CR 中的应用程序名称相匹配。
   - **spec.appVaultRef**：（必填）确保此值与您在第 1 步创建的 ResourceBackup CR 中的 appVaultRef 相匹配。
   - **spec.appArchivePath**：确保该值与您在第 1 步创建的 ResourceBackup CR 中的 appArchivePath 相匹配。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.action**：（*Required*）指示执行挂钩将采取哪些操作的字符串，假设指定的任何执行挂钩过滤器都匹配。可能的值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - **spec.stage**：（*Required*）指示执行挂钩应在操作期间运行哪个阶段的字符串。此挂钩运行不会在任何其他阶段运行挂钩。可能的值：
     - 之前
     - 发布

     示例 YAML：

```
---
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. 使用正确的值填充 CR 文件后，应用 CR：

```
kubectl apply -f trident-protect-hook-run.yaml
```

使用 **CLI**

步骤

1. 创建手动执行挂钩运行请求：

```
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. 检查执行挂钩运行的状态。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

如果要从试用版升级到完整版产品，则可能需要卸载 Trident Protect 组件。

要卸载 Trident Protect，请执行以下步骤。

步骤

1. 删除 Trident Protect CR 文件：

   25.06 版及更高版本不需要此步骤。

   ```
   helm uninstall -n trident-protect trident-protect-crds
   ```

2. 删除 Trident Protect：

   ```
   helm uninstall -n trident-protect trident-protect
   ```

3. 删除 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 10 月 16 日："适用于 Kubernetes 的高级存储解决方案"
- 2025 年 8 月 19 日："增强数据一致性：使用 Trident 在 OpenShift 虚拟化中实现卷组快照"
- 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 4 月 15 日："适用于 SMB 协议的 NetApp Trident with Google Cloud NetApp Volumes"
- 2025 年 4 月 14 日："利用 Fiber Channel Protocol 与 Trident 25.02 在 Kubernetes 上实现持久存储"
- 2025 年 4 月 14 日："为 Kubernetes 块存储解锁 NetApp ASA r2 系统的强大功能"
- 2025 年 3 月 31 日："使用新的认证 Operator 简化在 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 iSCSI 存储集成：AWS 上 ROSA 集群的 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："在 Trident 中引入光纤通道支持"
- 2024 年 11 月 11 日："使用 Google Cloud NetApp Volumes 的 NetApp Trident"
- 2024 年 10 月 29 日："使用 Trident 的 Amazon FSx for NetApp ONTAP with Red Hat OpenShift Service on AWS (ROSA)"
- 2024 年 10 月 29 日："在 ROSA 和 Amazon FSx for NetApp ONTAP 上使用 OpenShift Virtualization 实现虚拟机的实时迁移"
- 2024 年 7 月 8 日："使用 NVMe/TCP 为 Amazon EKS 上的现代容器化应用使用 ONTAP 存储"
- 2024 年 7 月 1 日："采用 Google Cloud NetApp Volumes Flex 和 Astra Trident 实现无缝 Kubernetes 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："使用 Trident Protect 恢复后挂钩自动化注册表故障转移以进行灾难恢复"
- 2025 年 5 月 16 日："OpenShift 虚拟化灾难恢复与 NetApp Trident Protect"
- 2025 年 5 月 13 日："使用 Trident Protect 备份和还原进行存储类迁移"
- 2025 年 5 月 9 日："使用 Trident Protect 恢复后挂钩重新缩放 Kubernetes 应用程序"
- 2025 年 4 月 3 日："Trident Protect 启动：Kubernetes 复制保护和灾难恢复"
- 2025 年 3 月 13 日："OpenShift 虚拟化 VM 的崩溃一致性备份和还原操作"
- 2025 年 3 月 11 日："将 GitOps 模式扩展到使用 NetApp Trident 的应用数据保护"
- 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："Trident Protect 基于角色的访问控制简介"

- 2024 年 11 月 11 日："Kubernetes 驱动的数据管理：Trident Protect 的新时代"

知识和支持

常见问题解答

查找有关安装、配置、升级和故障排除 Trident 的常见问题解答。

常规问题

Trident 多久发布一次?

从 24.02 版本开始，Trident 每四个月发布一次：2 月、6 月和 10 月。

Trident 支持 **Kubernetes** 特定版本发布的所有功能吗?

Trident 通常不支持 Kubernetes 中的 alpha 功能。Trident 可能在 Kubernetes beta 版本发布后的两个 Trident 版本中支持 beta 功能。

Trident 的功能是否依赖于其他 **NetApp** 产品?

Trident 不依赖于其他 NetApp 软件产品，可作为独立应用程序使用。但是，您应该有一个 NetApp 后端存储设备。

如何获取完整的 **Trident** 配置详情?

使用 `tridentctl get` 命令可获取有关 Trident 配置的更多信息。

我可以获取有关 **Trident** 如何配置存储的指标吗?

可以。可以使用 Prometheus 端点收集有关 Trident 操作的信息，例如管理的后端数量、配置的卷数量、消耗的字节数等。您还可以使用 "Cloud Insights" 进行监控和分析。

使用 **Trident** 作为 **CSI Provisioner** 时，用户体验是否会发生变化?

否。就用户体验和功能而言，没有任何变化。使用的置备程序名称为 `csi.trident.netapp.io`。如果要使用当前和未来版本提供的所有新功能，建议使用此安装 Trident 的方法。

在 Kubernetes 集群上安装和使用 Trident

Trident 支持从私有注册表进行离线安装吗?

是的，Trident 可以脱机安装。请参阅 "了解 Trident 安装"。

我可以远程安装 **Trident** 吗?

可以。Trident 18.10 及更高版本支持从任何具有 `kubectl`访问权限的机器进行远程安装。验证 `kubectl`访问权限后（例如，从远程机器启动 `kubectl get nodes`命令进行验证），请按照安装说明进行操作。

我可以使用 **Trident** 配置高可用性吗?

Trident 作为 Kubernetes Deployment (ReplicaSet) 安装在一个实例中，因此它内置了 HA。您不应增加部署中的副本数。如果安装 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 仅为控制面，因此如果重新部署 Trident，当前挂载的 pod 不会受到影响。

Trident 是否需要访问 **kube-system** 命名空间?

Trident 从 Kubernetes API Server 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 安装程序创建了一个 Kubernetes ClusterRole，它具有对 Kubernetes 集群的 PersistentVolume 、PersistentVolumeClaim、StorageClass 和 Secret 资源的特定访问权限。请参阅 "自定义 tridentctl 安装"。

我可以在本地生成 **Trident** 用于安装的确切清单文件吗?

如果需要，您可以在本地生成和修改 Trident 用于安装的确切清单文件。请参阅 "自定义 tridentctl 安装"。

我是否可以为两个独立的 **Kubernetes** 集群的两个独立的 **Trident** 实例共享相同的 **ONTAP** 后端 **SVM**?

虽然不建议这样做，但你可以为两个 Trident 实例使用同一个后端 SVM。在安装期间为每个实例指定唯一的卷名称，和/或在 `StoragePrefix`文件中指定唯一的 `setup/backend.json`参数。这样可以确保不会为两个实例使用同一个FlexVol volume。

是否可以在 **ContainerLinux**（原 **CoreOS**）下安装 **Trident**?

Trident 只是一个 Kubernetes pod，可以安装在运行 Kubernetes 的任何地方。

我可以将 **Trident** 与 **NetApp Cloud Volumes ONTAP** 一起使用吗?

是的，AWS、Google Cloud 和 Azure 支持 Trident。

故障排除和支持

NetApp 是否支持 **Trident**?

虽然 Trident 是开源的，并且是免费提供的，但如果支持您的 NetApp 后端，则 NetApp 完全支持它。

如何提交支持个案?

要提交支持案例，请执行以下操作之一：

1. 请联系您的 Support Account Manager，获取帮助以提交工单。
2. 通过联系 "NetApp 支持"提出支持个案。

如何生成支持日志包?

您可以通过运行 `tridentctl logs -a` 来创建支持捆绑包。除了捆绑包中捕获的日志之外，还捕获 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的请求，该怎么做？

在 "Trident Github"上创建问题，并在问题的主题和描述中提及 **RFE**。

我在哪里提出缺陷？

在 "Trident Github"上创建问题。请务必提供与此问题相关的所有必要信息和日志。

如果我对 **Trident** 有需要澄清的简单问题，该怎么办？是否有社区或论坛？

如果您有任何疑问、问题或要求，请通过我们的 Trident "Discord 频道" 或 GitHub 联系我们。

我的存储系统密码已更改，**Trident** 不再工作，如何恢复？

使用 `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident` 更新后端密码。将示例中的 `myBackend`替换为后端名称，将 `/path/to_new_backend.json`替换为正确`backend.json`文件的路径。

Trident 找不到我的 **Kubernetes** 节点。如何修复此问题？

有两种可能的情况，Trident 无法找到 Kubernetes 节点。这可能是因为 Kubernetes 中的网络问题或 DNS 问题。运行在每个 Kubernetes 节点上的 Trident 节点 daemonset 必须能够与 Trident 控制器通信，以便向 Trident 注册节点。如果在安装 Trident 后发生了网络更改，则只有在将新的 Kubernetes 节点添加到集群时才会遇到此问题。

如果 **Trident pod** 被摧毁，我会丢失数据吗？

如果 Trident pod 被销毁，数据不会丢失。Trident 元数据存储在 CRD 对象中。所有由 Trident 配置的 PV 都将正常运行。

升级 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即时主要版本。您可以从版本 18.xx 升级到 19.xx，19.xx 升级到 20.xx，依此类推。您应该在生产部署之前在实验室中测试升级。

是否可以将 **Trident** 降级到以前的版本？

如果需要修复升级后发现的错误、依赖问题或未成功或未完成的升级，则应 "卸载 Trident" 并使用该版本的特定说明重新安装早期版本。这是降级到早期版本的唯一建议方法。

管理后端和卷

我是否需要在 **ONTAP** 后端定义文件中同时定义 **Management** 和 **DataLIF**？

管理 LIF 是必需的。DataLIF 各不相同：

- ONTAP SAN：不为 iSCSI 指定。Trident 使用 "ONTAP 选择性 LUN 映射" 来发现建立多路径会话所需的 iSCI LIF。如果明确定义了 `dataLIF`，则会生成警告。有关详细信息，请参阅 "ONTAP SAN 配置选项和示例"。

- ONTAP NAS：NetApp 建议指定 `dataLIF`。如果未提供，Trident 从 SVM 获取 dataLIF。您可以指定要用于 NFS 装载操作的完全限定域名 (FQDN)，允许您创建循环 DNS 以跨多个 dataLIF 进行负载平衡。有关详细信息，请参阅"ONTAP NAS 配置选项和示例"

Trident 能否为 **ONTAP** 后端配置 **CHAP**?

是的。Trident 支持 ONTAP 后端的双向 CHAP。这需要在后端配置中设置 `useCHAP=true`。

如何使用 **Trident** 管理导出策略?

Trident 可以从 20.04 版开始动态创建和管理导出策略。这使存储管理员能够在其后端配置中提供一个或多个 CIDR 块，并让 Trident 将属于这些范围的节点 IP 添加到其创建的导出策略中。通过这种方式，Trident 会自动管理给定 CIDR 中具有 IP 的节点的添加和删除规则。

IPv6 地址可以用于管理和 **DataLIF** 吗?

Trident 支持为以下项定义 IPv6 地址：

- `managementLIF` 和 `dataLIF` 适用于 ONTAP NAS 后端。
- `managementLIF` 用于 ONTAP SAN 后端。您不能在 ONTAP SAN 后端上指定 `dataLIF`。

必须使用标记 `--use-ipv6`（用于 `tridentctl`安装）、 `IPv6`（用于 Trident 操作员）或 `tridentTPv6`（用于 Helm 安装）安装 Trident 才能在 IPv6 上运行。

是否可以在后端更新管理 **LIF**?

可以，可以使用 `tridentctl update backend` 命令更新后端管理 LIF。

是否可以在后端更新 **DataLIF**?

您只能在 `ontap-nas`和 `ontap-nas-economy`上更新 DataLIF。

我可以在 **Trident for Kubernetes** 中创建多个后端吗?

Trident 可以使用相同的驱动程序或不同的驱动程序同时支持多个后端。

Trident 如何存储后端凭据?

Trident 将后端凭据存储为 Kubernetes Secrets。

Trident 如何选择特定后端?

如果后端属性不能用于为类自动选择正确的池，则使用 `storagePools` 和 `additionalStoragePools` 参数来选择一组特定的池。

如何确保 **Trident** 不会从特定后端提供?

该 `excludeStoragePools` 参数用于筛选 Trident 用于配置的池集，并将删除所有匹配的池。

如果存在多个相同类型的后端，**Trident** 如何选择使用哪个后端？

如果有多个相同类型的配置后端，Trident 会根据 `StorageClass` 和 `PersistentVolumeClaim` 中存在的参数选择适当的后端。例如，如果有多个 ontap-nas 驱动程序后端，Trident 会尝试匹配 `StorageClass` 和 `PersistentVolumeClaim` 中的参数并进行组合，以匹配能够满足 `StorageClass` 和 `PersistentVolumeClaim` 中列出的要求的后端。如果有多个后端与请求匹配，Trident 会从中随机选择一个。

Trident 支持 **Element/SolidFire** 的双向 **CHAP** 吗？

是。

Trident 如何在 **ONTAP** 卷上部署 **Qtrees**？单个卷上可以部署多少个 **Qtrees**？

`ontap-nas-economy` 驱动程序在同一 `FlexVol` 卷中创建多达 `200` 个 `Qtree`（可在 `50` 到 `300` 之间配置），每个集群节点创建 `100,000` 个 `Qtree`，每个集群创建 `2.4M` 个。当您输入由经济型驱动程序提供服务的新 `PersistentVolumeClaim`时，驱动程序会查看是否已存在可以为新 `Qtree` 提供服务的 `FlexVol` 卷。如果不存在可以为 `Qtree` 提供服务的 `FlexVol` 卷，则会创建一个新 `FlexVol` 卷。

如何为在 **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来对 Trident 配置的卷设置 Unix 权限。

如何在配置卷时配置一组显式 **ONTAP NFS** 挂载选项？

默认情况下，Trident 不会使用 Kubernetes 将挂载选项设置为任何值。要指定 Kubernetes 存储类中的挂载选项，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，请使用后端定义文件中配置的 `exportPolicy` 参数。

如何通过 **Trident** 使用 **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。有关更多信息，请参阅："Trident 如何与 NVE 和 NAE 配合使用"

通过 **Trident** 为 **ONTAP** 实施 **QoS** 的最佳方式是什么？

使用 `StorageClasses`为 ONTAP 实施 QoS。

如何通过 **Trident** 指定精简配置或厚配置？

ONTAP 驱动程序支持精简或厚配置。ONTAP 驱动程序默认为精简配置。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` 则优先考虑。为 ONTAP 配置以下内容：

1. 在 `StorageClass`上，将 `provisioningType`属性设置为 thick。

2. 在后端定义文件中，通过设置 `backend spaceReserve parameter` 为 volume 来启用厚卷。

如何确保即使我不小心删除了 **PVC**，也不会删除正在使用的卷？

从 1.10 版开始，将在 Kubernetes 上自动启用 PVC 保护。

我可以扩展由 **Trident** 创建的 **NFS PVC** 吗？

是的。您可以扩展由 Trident 创建的 PVC。请注意，卷自动增长是 ONTAP 功能，不适用于 Trident。

可以在 **SnapMirror** 数据保护 **(DP)** 或离线模式下导入卷吗？

如果外部卷处于 DP 模式或脱机，则卷导入失败。您将收到以下错误消息：

```
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.
```

资源配额如何转换为 **NetApp** 集群？

只要 NetApp 存储有容量，Kubernetes 存储资源配额就应该有效。当 NetApp 存储因容量不足而无法兑现 Kubernetes 配额设置时，Trident 尝试预配，但会出现错误。

我可以使用 **Trident** 创建卷快照吗？

可以。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，请确保已启用 `VolumeSnapshotDataSource` 功能门。

支持 **Trident** 卷快照的驱动程序有哪些？

截至今天，我们的 `ontap-nas`、 `ontap-nas-flexgroup`、 `ontap-san`、 `ontap-san-economy`、 `solidfire-san` 和 `azure-netapp-files` 后端驱动程序均支持按需快照。

如何使用 **ONTAP** 对由 **Trident** 配置的卷进行快照备份？

此功能适用于 `ontap-nas`、 `ontap-san` 和 `ontap-nas-flexgroup` 驱动程序。您还可以在 FlexVol 层级为 `ontap-san-economy` 驱动程序指定 `snapshotPolicy`。

这也适用于 `ontap-nas-economy` 驱动程序，但适用于 FlexVol 卷级粒度，而不适用于 qtree 级粒度。要启用对 Trident 配置的卷进行快照的功能，请将后端参数选项 `snapshotPolicy` 设置为在 ONTAP 后端上定义的所需快照策略。存储控制器拍摄的任何快照都不被 Trident 识别。

我可以为通过 **Trident** 配置的卷设置快照预留百分比吗？

是的，您可以通过在后端定义文件中设置 `snapshotReserve`属性，通过 `Trident` 为存储快照副本预留特定百分比的磁盘空间。如果已在后端定义文件中配置 `snapshotPolicy`和 `snapshotReserve`，则会根据后端文件中提到的 `snapshotReserve`百分比设置快照保留百分比。如果未提及 `snapshotReserve`百分比数

字，则 ONTAP 默认情况下将快照保留百分比设为 5。如果 `snapshotPolicy`选项设置为 none，则快照保留百分比设置为 0。

我可以直接访问卷 **snapshot** 目录并复制文件吗?

是的，您可以通过在后端定义文件中设置 `snapshotDir` 参数来访问 Trident 配置的卷上的快照目录。

我可以通过 **Trident** 为卷设置 **SnapMirror** 吗?

目前，SnapMirror 必须使用 ONTAP CLI 或 OnCommand System Manager 从外部进行设置。

如何将永久卷恢复到特定的 **ONTAP** 快照?

要将卷还原到 ONTAP 快照，请执行以下步骤：

1. 将正在使用 Persistent 卷的应用程序 Pod 置于静默状态。
2. 通过 ONTAP CLI 或 OnCommand System Manager 还原到所需的快照。
3. 重新启动应用程序 pod。

Trident 能否在配置了负载分担镜像的 **SVM** 上配置卷?

可以为通过 NFS 提供数据的 SVM 根卷创建负载共享镜像。ONTAP 自动更新由 Trident 创建的卷的负载共享镜像。这可能会导致卷的安装延迟。使用 Trident 创建多个卷时，配置卷取决于 ONTAP 更新负载共享镜像。

如何为每个客户/租户分离存储类使用情况?

Kubernetes 不允许在命名空间中使用存储类。但是，您可以使用 Kubernetes 通过使用每个命名空间的存储资源配额来限制每个命名空间对特定存储类的使用。若要拒绝特定命名空间访问特定存储，请将该存储类的资源配额设置为 0。

故障排除

请使用此处提供的指针对安装和使用 Trident 时可能遇到的问题进行故障排除。

要获得有关 Trident 的帮助，请使用 `tridentctl logs -a -n trident`创建支持包并将其发送给 NetApp 支持人员。

常规故障排除

- 如果 Trident pod 无法正常启动（例如，当 Trident pod 卡在 `ContainerCreating`阶段且就绪容器少于两个时），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`）也会有所帮助。
- 如果 Trident 日志中没有足够的信息，您可以尝试通过根据您的安装选项将 `-d` 标志传递给安装参数来启用 Trident 的调试模式。

  然后使用 `./tridentctl logs -n trident` 确认已设置调试，并在日志中搜索 `level=debug msg`。

使用 **Operator** 安装

```
kubectl patch torc trident -n <namespace> --type=merge -p
'{"spec":{"debug":true}}'
```

这将重新启动所有 Trident pod，这可能需要几秒钟的时间。您可以通过观察 `kubectl get pod -n trident` 输出中的 'AGE' 列来检查这一点。

对于 Trident 20.07 和 20.10，请使用 `tprov`代替 `torc`。

已使用 **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
```

- 您还可以通过在后端定义中包含 `debugTraceFlags`来获取每个后端的调试日志。例如，包含 `debugTraceFlags: {"api":true, "method":true,}`即可在 Trident 日志中获取 API 调用和方法遍历。现有后端可以通过 `debugTraceFlags`与 `tridentctl backend update`一起配置。
- 使用 Red Hat Enterprise Linux CoreOS (RHCOS) 时，请确保 `iscsid` 已在工作节点上启用并默认启动。这可以使用 OpenShift MachineConfigs 或通过修改点火模板来完成。
- 使用 Trident 与 "Azure NetApp Files"时可能遇到的常见问题是租户和客户端密码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 PV 挂载到容器时出现问题，请确保 `rpcbind` 已安装并正在运行。请使用主机操作系统所需的软件包管理器，并检查 `rpcbind` 是否正在运行。您可以通过运行 `systemctl status rpcbind` 或其等效命令来检查 `rpcbind` 服务的状态。
- 如果 Trident 后端报告尽管以前工作过，但仍处于 `failed`状态，则可能是由于更改了与后端相关联的 SVM/管理员凭据。使用 `tridentctl update backend`更新后端信息或弹跳 Trident pod 将修复此问题。
- 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 `--in cluster=false` 标志安装 Trident。这将不使用安装程序 Pod，并避免因 `trident-installer` 用户而导致的权限问题。
- 使用 `uninstall parameter <Uninstalling Trident>`进行失败运行后的清理。默认情况下，该脚本不会删除 Trident 创建的 CRD，因此即使在正在运行的部署中，也可以安全地卸载并重新安装。
- 如果要降级到早期版本的 Trident，请先运行 `tridentctl uninstall` 命令以删除 Trident。下载所需的 "Trident 版本" 并使用 `tridentctl install` 命令进行安装。
- 成功安装后，如果 PVC 卡在 `Pending`阶段，运行 `kubectl describe pvc`可以提供有关 Trident 未能为此 PVC 配置 PV 的其他信息。

使用操作员部署 **Trident** 失败

如果您使用操作员部署 Trident， `TridentOrchestrator` 的状态会从 `Installing` 更改为 `Installed`。如

果观察到 `Failed` 状态，且操作员无法自行恢复，则应运行以下命令检查操作员的日志：

```
tridentctl logs -l trident-operator
```

跟踪 trident-operator 容器的日志可以指向问题所在。例如，一个这样的问题可能是无法从气隙环境中的上游注册表中提取所需的容器映像。

要了解 Trident 安装失败的原因，您应该查看 `TridentOrchestrator` 状态。

```
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'
```

此错误表示已存在用于安装 Trident 的 `TridentOrchestrator`。由于每个 Kubernetes 集群只能有一个 Trident 实例，因此操作员确保在任何给定时间只存在一个可以创建的活动 `TridentOrchestrator`。

此外，观察 Trident pod 的状态通常可以指示是否有问题。

```
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，由于未获取一个或多个容器镜像，因此 pod 无法完全初始化。

要解决此问题，您应该编辑 `TridentOrchestrator` CR。或者，您可以删除 `TridentOrchestrator`，并使用修改后的准确定义创建一个新定义。

使用 `tridentctl`进行 Trident 部署失败

为了帮助找出问题所在，您可以使用 `-d` 参数再次运行安装程序，该参数将打开调试模式并帮助您了解问题所在：

```
./tridentctl install -n trident -d
```

解决问题后，可以按照以下步骤清理安装，然后再次运行 `tridentctl install` 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.
```

完全移除 Trident 和 CRD

您可以完全删除 Trident 和所有创建的 CRD 以及相关的自定义资源。

此操作无法撤消。除非您想要全新安装 Trident，否则请勿执行此操作。要卸载 Trident 而不删除 CRD，请参阅 "卸载 Trident"。

Trident 操作员

要使用 Trident 操作员卸载 Trident 并完全删除 CRD：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

Helm

要使用 Helm 卸载 Trident 并完全删除 CRD：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>tridentctl</code>

使用 `tridentctl` 卸载 Trident 后完全删除 CRD

```
tridentctl obliviate crd
```

在 **Kubernetes 1.26** 上使用 **RWX** 原始块命名空间的 **NVMe** 节点取消暂存失败

如果运行的是 Kubernetes 1.26，则在将 NVMe/TCP 与 RWX 原始块命名空间一起使用时，节点取消暂存可能会失败。以下场景提供了故障的解决方案。或者，您可以将 Kubernetes 升级到 1.27。

已删除命名空间和 **pod**

考虑将 Trident 托管命名空间（NVMe 持久卷）附加到 pod 的情况。如果直接从 ONTAP 后端删除命名空间，则在尝试删除 pod 后，取消暂存过程将停滞。此场景不会影响 Kubernetes 集群或其他功能。

临时解决策

从相应节点卸载永久卷（对应于该命名空间）并删除它。

被阻止的 **dataLIF**

```
 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

将 `hostNQN`添加回子系统。

NFSv4.2 客户端在升级 ONTAP 后报告"invalid argument"，当预期启用"v4.2-xattrs"时

升级 ONTAP 后，NFSv4.2 客户端在尝试装载 NFSv4.2 导出时可能会报告"无效参数"错误。如果未在 SVM 上启用 `v4.2-xattrs`选项，则会出现此问题。解决方法：在 SVM 上启用 `v4.2-xattrs`选项或升级到 ONTAP 9.12.1 或更高版本，默认情况下启用此选项。

支持

NetApp 以各种方式为 Trident 提供支持。我们提供全天候免费自助服务选项，例如知识库 (KB) 文章和 Discord 频道。

Trident 支持生命周期

Trident 根据您的版本提供三个级别的支持。请参阅 "NetApp 软件版本支持的定义"。

完全支持
: Trident 从发布日期起提供 12 个月的全面支持。

有限支持
: Trident 在发布日期后的第 13 - 24 个月提供有限支持。

自助支持
: Trident 文档自发布之日起 25 - 36 个月内可用。

版本	完全支持	有限支持	自助支持
"25.10"	2026 年 10 月	2027 年 10 月	2028 年 10 月
"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"	—	2026 年 10 月	2027 年 10 月
"24.06"	—	2026 年 6 月	2027 年 6 月

"24.02"	—	2026 年 2 月	2027 年 2 月
"23.10"	—	—	2026 年 10 月
"23.07"	—	—	2026年7月
"23.04"	—	—	2026年4月
"23.01"	—	—	2026年1月

自助支持

有关疑难解答文章的完整列表，请参阅 "NetApp 知识库（需要登录）"。

社区支持

我们的 "Discord 频道" 上有一个充满活力的容器用户（包括 Trident 开发人员）公共社区。这是询问有关该项目的一般性问题并与志同道合的同事讨论相关主题的好地方。

NetApp 技术支持

要获得有关 Trident 的帮助，请使用 `tridentctl logs -a -n trident`创建支持包并将其发送到 `NetApp Support <Getting Help>`。

了解更多信息

- "Trident 资源"
- "Kubernetes Hub"

参考

Trident 端口

详细了解 Trident 用于通信的端口。

概述

Trident 使用各种端口在 Kubernetes 集群内以及与存储后端进行通信。以下是关键端口、其用途和安全注意事项的摘要。

- 出站焦点：Kubernetes 节点（控制器和工作节点）主要启动到存储 LIF/IP 的流量，因此 iptables 规则应允许从节点 IP 出站到这些端口上的特定存储 IP。避免宽泛的"任意对任意"规则。
- 入站限制：将内部 Trident 端口限制为集群内部流量（例如，使用像 Calico 这样的 CNI）。主机防火墙上没有不必要的入站曝光。
- 协议安全性：
  - 尽可能使用 TCP（更可靠）。
  - 如果敏感，为 iSCSI 启用 CHAP/IPsec；为管理启用 TLS/HTTPS（端口 443/8443）。
  - 对于 NFSv4（Trident 中的默认值），如果不需要，请修剪 UDP/较旧的 NFSv3 端口（例如，4045-4049）。
  - 仅限于受信任的子网；使用 Prometheus（可选端口 8001）等工具进行监控。

控制器节点的端口

这些端口主要用于 Trident operator（后端管理）。所有内部端口都是 pod 级；仅当主机防火墙干扰 CNI 时才允许在节点上使用。

端口/协议	方向	目的	驱动程序/协议	安全说明
TCP 8000	入站/出站（集群内部）	Trident REST 服务器（操作员-控制器通信）	全部	仅限于 pod CIDR；无外部暴露。
TCP 8443	入站/出站（集群内部）	反向通道 HTTPS（安全内部 API）	全部	TLS 加密；如果使用，则限制为 Kubernetes 服务网格。
TCP 8001	入站（集群内部，可选）	Prometheus 指标	全部	仅暴露给监控工具（例如，使用 RBAC）；如果未使用，则禁用。
TCP 443	出站	HTTPS 到 ONTAP SVM/集群管理 LIF	ONTAP（all）、ANF	需要 TLS 证书验证；仅限于管理 LIF IP。
TCP 8443	出站	HTTPS 到 E-Series Web Services Proxy	E 系列 (iSCSI)	默认 REST API；使用证书；可在后端 YAML 中配置。

工作节点的端口

这些端口用于 CSI 节点守护程序集和 pod 挂载。数据端口出站到存储数据 LIF；如果使用 NFSv3，则包括 NFSv3 附加组件（NFSv4 可选）。

端口/协议	方向	目的	驱动程序/协议	安全说明
TCP 17546	入站（本地到 pod）	CSI 节点活跃度/就绪探测	全部	可配置（--probe-port）；确保没有主机冲突；仅限本地。
TCP 8000	入站/出站（集群内部）	Trident REST 服务器	全部	如上所述；pod-internal。
TCP 8443	入站/出站（集群内部）	反向通道 HTTPS	全部	同上。
TCP 8001	入站（集群内部，可选）	Prometheus 指标	全部	同上。
TCP 443	出站	HTTPS 到 ONTAP SVM/集群管理 LIF	ONTAP（all）、ANF	如上；用于发现。
TCP 8443	出站	HTTPS 到 E-Series Web Services Proxy	E 系列 (iSCSI)	同上。
TCP/UDP 111	出站	RPCBIND/portmapper	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	v3 必需；v4 可选（防火墙卸载）；如果仅使用 NFSv4，则限制。
TCP/UDP 2049	出站	NFS 守护进程	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	核心数据；众所周知；使用 TCP 实现可靠性。
TCP/UDP 635	出站	挂载守护进程	ONTAP-NAS (NFSv3/v4)、ANF (NFS)	挂载；可以进行双向回调（如果需要，允许入站临时连接）。
UDP 4045	出站	NFS 锁定管理器 (nlockmgr)	ONTAP-NAS（NFSv3）	文件锁定；跳过 v4（pNFS 句柄）；仅限 UDP。
UDP 4046	出站	NFS 状态监控 (statd)	ONTAP-NAS（NFSv3）	通知；可能需要入站临时端口 (1024-65535) 进行回拨。
UDP 4049	出站	NFS 配额守护进程 (rquotad)	ONTAP-NAS（NFSv3）	配额；跳过 v4。
TCP 3260	出站	iSCSI 目标（发现/数据/CHAP）	ONTAP-SAN (iSCSI)、E-Series (iSCSI)	众所周知；通过此端口进行 CHAP 身份验证；启用双方 CHAP 以实现安全。
TCP 445	出站	SMB/CIFS	ONTAP-NAS (SMB)、ANF (SMB)	众所周知；使用 SMB3 进行加密（Trident annotation netapp.io/smb-encryption=true）。
TCP/UDP 88（可选）	出站	Kerberos 身份验证	ONTAP（带 Kerb 的 NFS/SMB/iSCSI）	如果使用 Kerberos（非默认）；到 AD 服务器，而不是存储。
TCP/UDP 389（可选）	出站	LDAP	ONTAP（使用 LDAP 的 NFS/SMB）	类似；用于名称解析/身份验证；限制为 AD。

在安装过程中，可以使用 `--probe-port`标志更改活动/就绪探头端口。请务必确保此端口未被工作节点上的其他进程使用。

Trident REST API

虽然 "tridentctl 命令和选项" 是与 Trident REST API 交互的最简单方法，但如果您愿意，可以直接使用 REST 端点。

何时使用 REST API

REST API 对于在非 Kubernetes 部署中使用 Trident 作为独立二进制文件的高级安装非常有用。

为了提高安全性，默认情况下在 pod 内运行时，Trident `REST API` 被限制为 localhost。要更改此行为，需要在其 pod 配置中设置 Trident 的 `-address` 参数。

使用 REST API

有关如何调用这些 API 的示例，请传递 debug (`-d` 标志。有关详细信息，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>`

列出此类型的所有对象。

`GET <trident-address>/trident/v1/<object-type>/<object-name>`

获取命名对象的详细信息。

POST

`POST <trident-address>/trident/v1/<object-type>`

创建指定类型的对象。

- 需要为要创建的对象提供 JSON 配置。有关每种对象类型的规格，请参阅 "使用 tridentctl 管理 Trident"。
- 如果对象已存在，行为会有所不同：后端更新现有对象，而所有其他对象类型将使操作失败。

DELETE

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`

删除命名资源。

与后端或存储类关联的卷将继续存在；必须单独删除这些卷。有关详细信息，请参阅 "使用 tridentctl 管理 Trident"。

命令行选项

Trident 为 Trident orchestrator 提供了多个命令行选项。您可以使用这些选项修改部署。

日志记录

-debug

启用调试输出。

-loglevel <level>

设置日志记录级别（debug、info、warn、error、fatal）。默认为 info。

Kubernetes

-k8s_pod

使用此选项或 `-k8s_api_server` 来启用 Kubernetes 支持。设置此选项会导致 Trident 使用其包含的 pod 的 Kubernetes 服务帐户凭据来联系 API 服务器。仅当 Trident 在启用了服务帐户的 Kubernetes 集群中作为 pod 运行时，此功能才有效。

-k8s_api_server <insecure-address:insecure-port>

使用此选项或 `-k8s_pod` 来启用 Kubernetes 支持。指定时，Trident 使用提供的不安全地址和端口连接到 Kubernetes API 服务器。这使 Trident 能够部署在 pod 之外；但是，它仅支持到 API 服务器的不安全连接。要安全地连接，请使用 `-k8s_pod` 选项在 pod 中部署 Trident。

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>

侦听此端口而不是 UNIX 域套接字。

-config <file>

必需；必须指定后端配置文件的此路径。

REST

-address <ip-or-host>

指定 Trident 的 REST 服务器应侦听的地址。默认为 localhost。在 localhost 上监听并在 Kubernetes pod 内运行时，无法从 pod 外部直接访问 REST 接口。使用 `-address ""`使 REST 接口可从 pod IP 地址访问。

> 可以将 Trident REST 接口配置为仅在 127.0.0.1（用于 IPv4）或 [::1]（用于 IPv6）下侦听和服务。

-port <port-number>

指定 Trident 的 REST 服务器应监听的端口。默认为 8000。

`-rest`

启用 REST 接口。默认为 true。

Kubernetes 和 Trident 对象

您可以通过读取和写入资源对象，使用 REST API 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，其他对象通过 Trident 进行管理。

这些对象如何相互作用?

也许理解对象、它们的用途以及它们如何交互的最简单方法是遵循 Kubernetes 用户的单个存储请求：

1. 用户创建了一个 `PersistentVolumeClaim`，请求从管理员之前配置的 Kubernetes `StorageClass`中分配一个特定大小的 `PersistentVolume`。
2. Kubernetes `StorageClass` 将 Trident 识别为其配置程序，并包含告诉 Trident 如何为请求的类配置卷的参数。
3. Trident 查看其自己的 `StorageClass`，该名称用于标识匹配的 `Backends`和 `StoragePools`，它可以使用这些来为类配置卷。
4. Trident 在匹配的后端上配置存储并创建两个对象：Kubernetes 中的一个 `PersistentVolume`，它告诉 Kubernetes 如何查找、挂载和处理卷，以及 Trident 中的一个卷，它保留 `PersistentVolume`与实际存储之间的关系。
5. Kubernetes 将 `PersistentVolumeClaim` 绑定到新的 `PersistentVolume`。包含 `PersistentVolumeClaim` 的 Pod 会在其运行的任何主机上挂载该 PersistentVolume。
6. 用户使用指向 Trident 的 `VolumeSnapshotClass`，为现有 PVC 创建 `VolumeSnapshot`。
7. Trident 标识与 PVC 关联的卷，并在其后端创建卷的快照。它还创建了一个 `VolumeSnapshotContent`，指导 Kubernetes 如何识别快照。
8. 用户可以创建一个 `PersistentVolumeClaim`，并使用 `VolumeSnapshot`作为源。
9. Trident 识别所需的快照，并执行创建 `PersistentVolume`和 `Volume`所涉及的同一组步骤。

如需进一步了解 Kubernetes 对象，我们强烈建议您阅读 Kubernetes 文档的 "持久卷"部分。

Kubernetes `PersistentVolumeClaim` 对象

Kubernetes `PersistentVolumeClaim` 对象是 Kubernetes 集群用户发出的存储请求。

除了标准规范之外，如果要覆盖您在后端配置中设置的默认值，Trident 允许用户指定以下特定于卷的注释：

标注	卷选项	支持的驱动程序
trident.netapp.io/fileSystem	fileSystem	ontap-san, solidfire-san, ontap-san-economy

标注	卷选项	支持的驱动程序
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas，ontap-san，solidfire-san，azure-netapp-files，ontap-san-economy
trident.netapp.io/splitOnClone	splitOnClone	ontap-nas，ontap-san
trident.netapp.io/protocol	protocol	任意
trident.netapp.io/exportPolicy	exportPolicy	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/snapshotPolicy	snapshotPolicy	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san
trident.netapp.io/snapshotReserve	snapshotReserve	ontap-nas、ontap-nas-flexgroup、ontap-san
trident.netapp.io/snapshotDirectory	snapshotDirectory	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/unixPermissions	unixPermissions	ontap-nas、ontap-nas-economy、ontap-nas-flexgroup
trident.netapp.io/blockSize	blockSize	solidfire-san
trident.netapp.io/skipRecoveryQueue	skipRecoveryQueue	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，ontap-san-economy

如果创建的 PV 具有 `Delete`回收策略，则 `Trident` 会在 `PV` 释放时（即用户删除 `PVC` 时）同时删除 `PV` 和备份卷。如果删除操作失败，`Trident` 会将 `PV` 标记为此类状态，并定期重试该操作，直到操作成功或 `PV` 被手动删除。如果 `PV` 使用 `Retain`策略，`Trident` 会忽略它，并假设管理员将从 `Kubernetes` 和后端清理它，允许在删除之前备份或检查卷。请注意，删除 `PV` 不会导致 `Trident` 删除备份卷。您应该使用 `REST API (`tridentctl）将其删除。

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克隆现有 PVC 的数据源。这样，PV 的时间点副本可以以快照的形式暴露给 Kubernetes。然后，可使用快照创建新的 PV。看一 `On-Demand Volume Snapshots`看这是如何运作的。

Trident 还提供用于创建克隆的 `cloneFromPVC`和 `splitOnClone`注释。您可以使用这些注释来克隆 PVC，而无需使用 CSI 实现。

示例如下：如果用户已经有一个名为 `mysql`的 `PVC`，则用户可以使用注释创建一个名为 `mysqlclone`的新 `PVC`，例如 `trident.netapp.io/cloneFromPVC: mysql`。使用此注释集，Trident 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 及其克隆应位于同一个 Kubernetes 命名空间中，并具有相同的存储类。
- 使用 `ontap-nas`和 `ontap-san`驱动程序时，可能需要将 PVC 注释 `trident.netapp.io/splitOnClone`与 `trident.netapp.io/cloneFromPVC`结合设置。将 `trident.netapp.io/splitOnClone`设置为 `true`时，Trident 会将克隆卷从父卷中拆分出来，从而使克隆卷的生命周期与其父卷完全脱钩，代价是失去一些存储效率。不设置 `trident.netapp.io/splitOnClone`或将其设置为 `false`会减少后端的空间消耗，代价是在父卷和克隆卷之间

创建依赖关系，因此除非先删除克隆，否则无法删除父卷。拆分克隆有意义的场景是克隆一个空的数据库卷，预计该卷及其克隆会大大分化，并且不会从 ONTAP 提供的存储效率中受益。

该 `sample-input`目录包含用于 Trident 的 PVC 定义示例。有关与 Trident 卷相关的参数和设置的完整说明，请参阅。

Kubernetes `PersistentVolume` 对象

Kubernetes `PersistentVolume` 对象表示可供 Kubernetes 集群使用的一块存储。它的生命周期与使用它的 Pod 无关。

Trident 创建 `PersistentVolume`对象，并根据其配置的卷自动将其注册到 Kubernetes 集群。您无需自行管理。

当您创建引用基于 Trident 的 `StorageClass` PVC 时，Trident 使用相应的存储类来配置新卷，并为该卷注册新的 PV。在配置已配置的卷和对应的 PV 时，Trident 遵循以下规则：

- Trident 为 Kubernetes 生成一个 PV 名称以及用于配置存储的内部名称。在这两种情况下，它都确保名称在其范围内是唯一的。
- 卷的大小与 PVC 中请求的大小尽可能相匹配，但可能会四舍五入到最接近的可分配数量，具体取决于平台。

Kubernetes `StorageClass` 对象

Kubernetes `StorageClass` 对象在 `PersistentVolumeClaims`中按名称指定，以配置具有一组属性的存储。存储类本身标识要使用的置备程序，并以置备程序理解的术语定义该属性集。

它是需要由管理员创建和管理的两个基本对象之一。另一个是 Trident 后端对象。

使用 Trident 的 Kubernetes `StorageClass` 对象如下所示：

```
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 特定的，告诉 Trident 如何为该类配置卷。

storage class 参数为：

属性	类型	必填项	说明
属性	map[string]string	不可以	请参见下面的属性部分

属性	类型	必填项	说明
storagePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射
additionalStoragePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射
excludeStoragePools	map[string]StringList	不可以	后端名称到其中存储池列表的映射

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

这些参数确定应使用哪些 Trident 管理的存储池来配置给定类型的卷。

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

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

[1]：ONTAP Select 系统不支持

在大多数情况下，请求的值直接影响调配；例如，请求厚调配会导致厚调配的卷。但是，Element 存储池使用其提供的 IOPS 最小值和最大值来设置 QoS 值，而不是请求的值。在这种情况下，请求的值仅用于选择存储池。

理想情况下，您可以单独使用 `attributes`来模拟满足特定类别需求所需的存储质量。Trident 会自动发现并选择与您指定的所有 `attributes`匹配的存储池。

如果您发现自己无法使用 `attributes` 为某个类自动选择正确的池，则可以使用 `storagePools` 和 `additionalStoragePools` 参数进一步优化池，甚至可以选择一组特定的池。

您可以使用 `storagePools` 参数进一步限制与任何指定 `attributes` 匹配的池集。换句话说，Trident 使用由 `attributes` 和 `storagePools` 参数标识的池的交集进行配置。您可以单独使用一个参数，也可以同时使用两个参数。

您可以使用 `additionalStoragePools` 参数扩展 Trident 用于配置的池集，而不考虑 `attributes` 和 `storagePools` 参数选择的任何池。

您可以使用 `excludeStoragePools` 参数来筛选 Trident 用于配置的池集。使用此参数将删除所有匹配的池。

在 `storagePools`和 `additionalStoragePools`参数中，每个条目都采用 `<backend>:<storagePoolList>`形式，其中 `<storagePoolList>`是指定后端的存储池的逗号分隔列表。例如， `additionalStoragePools`的值可能看起来像 `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`获取后端及其池的列表。

Kubernetes 属性

在动态配置期间，这些属性不会影响 Trident 对存储池/后端的选择。相反，这些属性只提供 Kubernetes 持久卷支持的参数。工作节点负责文件系统创建操作，并且可能需要文件系统实用程序，例如 xfsprogs。

属性	类型	值	说明	相关驱动程序	**Kubernetes** 版本
fsType	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
allowVolumeExpansion	布尔值	true，false	启用或禁用对增加 PVC 尺寸的支持	ontap-nas，ontap-nas-economy，ontap-nas-flexgroup，ontap-san，ontap-san-economy，solidfire-san，azure-netapp-files	1.11+
volumeBindingMode	string	立即，WaitForFirstConsumer	选择何时进行卷绑定和动态配置	全部	1.19 - 1.26

- `fsType` 参数用于控制 SAN LUNs 所需的文件系统类型。此外，Kubernetes 还会在存储类中使用 `fsType` 的存在来指示文件系统已存在。只有在设置了 `fsType` 时，才能使用 pod 的 `fsGroup` 安全上下文来控制卷的所有权。请参阅 "Kubernetes：为 Pod 或容器配置安全上下文"，了解如何使用 `fsGroup` 上下文设置卷所有权的概述。Kubernetes 仅在以下情况下应用 `fsGroup` 值：
  - `fsType` 在存储类中设置。
  - PVC 访问模式为 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。要使用 `fsGroup`，存储类仍需要指定 `fsType`。您可以将其设置为 `nfs`或任何非空值。
- 有关卷扩展的更多详细信息，请参见 "扩展卷"。
- Trident 安装程序捆绑包提供了几个示例存储类定义，可与 `sample-input/storage-class-*.yaml` 中的 Trident 一起使用。删除 Kubernetes 存储类也会导致相应的 Trident 存储类被删除。

Kubernetes `VolumeSnapshotClass` 对象

Kubernetes `VolumeSnapshotClass` 对象类似于 `StorageClasses`。它们有助于定义多个存储类，并由卷快照引用，以将快照与所需的快照类相关联。每个卷快照都与单个卷快照类相关联。

`VolumeSnapshotClass` 应由管理员定义以创建快照。使用以下定义创建卷快照类：

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

`driver`向 Kubernetes 指定 `csi-snapclass`类的卷快照请求由 Trident 处理。`deletionPolicy`指定必须删除快照时要采取的操作。当 `deletionPolicy`设置为 `Delete`时，删除快照时将删除卷快照对象以及存储集群上的底层快照。或者，将其设置为 `Retain`意味着保留 `VolumeSnapshotContent`和物理快照。

Kubernetes VolumeSnapshot 对象

Kubernetes VolumeSnapshot 对象是创建卷的快照的请求。正如 PVC 表示用户对卷的请求一样，卷快照是用户为创建现有 PVC 的快照而发出的请求。

当卷快照请求进入时，Trident 会自动管理后端上卷的快照创建，并通过创建唯一 `VolumeSnapshotContent`对象公开快照。您可以从现有 PVC 创建快照，并在创建新 PVC 时将快照用作 DataSource。

VolumeSnapshot 的生命周期独立于源 PVC：即使删除了源 PVC，快照也会持续存在。删除具有关联快照的 PVC 时，Trident 会将此 PVC 的后备卷标记为 **Deleting** 状态，但不会将其完全删除。删除所有关联的快照后，该卷将被删除。

Kubernetes VolumeSnapshotContent 对象

Kubernetes VolumeSnapshotContent 对象表示从已配置的卷中获取的快照。它类似于 PersistentVolume，表示在存储集群上调配的快照。与 `PersistentVolumeClaim`和 `PersistentVolume`对象类似，创建快照时， `VolumeSnapshotContent`对象会维护到已请求创建快照的 `VolumeSnapshot`对象的一对一映射。

`VolumeSnapshotContent` 对象包含唯一标识快照的详细信息，例如 `snapshotHandle`。这个 `snapshotHandle`是 PV 名称和 `VolumeSnapshotContent`对象名称的唯一组合。

当快照请求进入时，Trident 会在后端创建快照。创建快照后，Trident 将配置一个 `VolumeSnapshotContent`对象，从而将快照公开给 Kubernetes API。

通常，您无需管理 VolumeSnapshotContent 对象。此操作的例外情况是，您希望 "导入卷快照" 在 Trident 外部创建。

Kubernetes VolumeGroupSnapshotClass 对象

Kubernetes VolumeGroupSnapshotClass 对象类似于 VolumeSnapshotClass。它们有助于定义多个存储类，并由卷组快照引用，以将快照与所需的快照类相关联。每个卷组快照都与单个卷组快照类相关联。

`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
```

`driver` 向 Kubernetes 指定 `csi-group-snap-class` 类的卷组快照请求由 Trident 处理。 `deletionPolicy` 指定必须删除组快照时要采取的操作。当 `deletionPolicy` 设置为 `Delete`
时，删除快照时将删除卷组快照对象以及存储集群上的底层快照。或者，将其设置为 `Retain` 表示保留 `VolumeGroupSnapshotContent` 和物理快照。

Kubernetes VolumeGroupSnapshot 对象

Kubernetes VolumeGroupSnapshot 对象是创建多个卷的快照的请求。正如 PVC 表示用户对卷的请求一样，卷组快照是用户为创建现有 PVC 的快照而提出的请求。

当卷组快照请求进入时，Trident 会自动管理后端上卷的组快照的创建，并通过创建唯一 `VolumeGroupSnapshotContent`对象来公开快照。您可以从现有 PVC 创建快照，并在创建新 PVC 时将快照用作 DataSource。

VolumeGroupSnapshot 的生命周期独立于源 PVC：即使删除了源 PVC，快照也会持续存在。删除具有关联快照的 PVC 时，Trident 会将此 PVC 的后备卷标记为 **Deleting** 状态，但不会将其完全删除。删除所有关联的快照后，会删除卷组快照。

Kubernetes VolumeGroupSnapshotContent 对象

Kubernetes VolumeGroupSnapshotContent 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。与 `PersistentVolumeClaim`和 `PersistentVolume`对象类似，创建快照时， `VolumeSnapshotContent`对象会维护到已请求创建快照的 `VolumeSnapshot`对象的一对

一映射。

`VolumeGroupSnapshotContent` 对象包含标识快照组的详细信息，例如 `volumeGroupSnapshotHandle` 和存储系统上存在的各个 volumeSnapshotHandles。

当快照请求进入时，Trident 在后端创建卷组快照。创建卷组快照后，Trident 配置一个 `VolumeGroupSnapshotContent`对象，从而将快照公开给 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 支持创建用于存储对象集合的自定义资源。您可以通过运行 `kubectl get crds` 来获取这些资源定义。

自定义资源定义 (CRD) 及其关联的对象元数据由 Kubernetes 存储在其元数据存储中。这消除了对 Trident 单独存储的需要。

Trident 使用 `CustomResourceDefinition`对象来保留 Trident 对象的身份，例如 Trident 后端、Trident 存储类和 Trident 卷。这些对象由 Trident 管理。此外，CSI 卷快照框架还引入了定义卷快照所需的一些 CRD。

CRD 是 Kubernetes 构造。上述定义的资源对象由 Trident 创建。举个简单的例子，当使用 `tridentctl`创建后端时，会创建相应的 `tridentbackends` CRD 对象供 Kubernetes 使用。

以下是关于 Trident CRD 需要记住的几点：

- 安装 Trident 后，将创建一组 CRD，可以像任何其他资源类型一样使用。
- 使用 `tridentctl uninstall` 命令卸载 Trident 时，Trident pod 会被删除，但不会清理创建的 CRD。请参阅"卸载 Trident"了解如何从头开始完全移除和重新配置 Trident。

Trident StorageClass 对象

Trident 为 Kubernetes `StorageClass`对象创建匹配的存储类，这些对象在其 `provisioner` 字段中指定 `csi.trident.netapp.io`。存储类名称与其所代表的 Kubernetes `StorageClass`对象名称相同。

使用 Kubernetes 时，当注册使用 Trident 作为 provisioner 的 Kubernetes `StorageClass` 时，这些对象会自动创建。

存储类包括卷的一组要求。Trident 将这些要求与每个存储池中存在的属性进行匹配；如果它们匹配，则该存储池是使用该存储类调配卷的有效目标。

您可以创建存储类配置，通过使用 REST API 直接定义存储类。但是，对于 Kubernetes 部署，我们希望在注册新的 Kubernetes `StorageClass` 对象时创建它们。

Trident 后端对象

后端代表 Trident 在其上配置卷的存储提供程序；单个 Trident 实例可以管理任意数量的后端。

这是您自己创建和管理的两种对象类型之一。另一个是 Kubernetes `StorageClass` 对象。

有关如何构造这些对象的详细信息，请参阅 "配置后端"。

Trident `StoragePool` 对象

存储池表示每个后端上可用于设置的不同位置。对于 ONTAP，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。每个存储池都有一组不同的存储属性，这些属性定义了其性能特征和数据保护特征。

与这里的其他对象不同，存储池候选项始终会被自动发现和管理。

Trident `Volume` 对象

卷是资源调配的基本单位，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，请确保该卷具有存储类，该类确定可以调配该卷的位置以及大小。

- 在 Kubernetes 中，这些对象是自动管理的。您可以查看它们以查看 Trident 配置的内容。
- 删除具有关联快照的 PV 时，相应的 Trident 卷将更新为 **Deleting** 状态。要删除 Trident 卷，应删除卷的快照。

卷配置定义了已配置卷应具有的属性。

属性	类型	必填项	说明
version	string	不可以	Trident API 版本（"1"）
name	string	可以	要创建的卷的名称
storageClass	string	可以	调配卷时要使用的存储类
大小	string	可以	要设置的卷的大小（以字节为单位）
protocol	string	不可以	要使用的协议类型；"file" 或 "block"
internalName	string	不可以	存储系统上的对象名称；由 Trident 生成
cloneSourceVolume	string	不可以	ontap (nas, san) & solidfire-*：要从中克隆的卷的名称
splitOnClone	string	不可以	ONTAP (nas, san)：从其父项拆分克隆
snapshotPolicy	string	不可以	ontap-*：要使用的 Snapshot 策略
snapshotReserve	string	不可以	ontap-*：为 Snapshot 预留的卷百分比
exportPolicy	string	不可以	ontap-nas*：要使用的导出策略
snapshotDirectory	布尔值	不可以	ontap-nas*：快照目录是否可见

属性	类型	必填项	说明
unixPermissions	string	不可以	ontap-nas*：初始 UNIX 权限
blockSize	string	不可以	solidfire-*：块/扇区大小
fileSystem	string	不可以	文件系统类型
skipRecoveryQueue	string	不可以	在卷删除过程中，绕过存储中的恢复队列并立即删除卷。

Trident 在创建卷时会生成 `internalName`。这包括两个步骤。首先，它将存储前缀（默认 `trident`或后端配置中的前缀）添加到卷名前面，从而生成形式为 `<prefix>-<volume-name>`的名称。然后继续清理名称，替换后端不允许的字符。对于 `ONTAP` 后端，它用下划线替换连字符（因此，内部名称变为 `<prefix>_<volume-name>`）。对于 Element 后端，它用连字符替换下划线。

您可以使用卷配置直接使用 REST API 配置卷，但在 Kubernetes 部署中，我们希望大多数用户使用标准的 Kubernetes `PersistentVolumeClaim` 方法。Trident 会在配置过程中自动创建此卷对象。

Trident `Snapshot` 对象

快照是卷的时间点副本，可用于配置新卷或恢复状态。在 Kubernetes 中，它们直接对应于 `VolumeSnapshotContent`对象。每个快照都与一个卷相关联，该卷是快照的数据源。

每个 `Snapshot` 对象都包含以下列出的属性：

属性	类型	必填项	说明
version	字符串	是	Trident API 版本（"1"）
name	字符串	是	Trident 快照对象的名称
internalName	字符串	是	存储系统上的 Trident 快照对象的名称
volumeName	字符串	是	为其创建快照的永久卷的名称
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。您可以查看它们以查看 Trident 配置的内容。

当创建 Kubernetes `VolumeSnapshot`对象请求时，`Trident` 通过在后端存储系统上创建一个快照对象来工作。该快照对象的 `internalName`是通过将前缀 `snapshot-`与 `UID`的 `VolumeSnapshot`对象组合生成的（例如， `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660`）。`volumeName`和 `volumeInternalName`通过获取后端卷的详细信息来填充。

Trident `ResourceQuota` 对象

Trident daemonset 使用 `system-node-critical`优先级类（Kubernetes 中可用的最高优先级类），以确保

Trident 能够在正常节点关闭期间识别和清理卷，并允许 Trident daemonset pod 在资源压力较大的集群中抢占较低优先级的工作负载。

为了实现这一点，Trident 使用了一个 `ResourceQuota`对象来确保满足 Trident daemonset 上的"system-node-critical"优先级类。在部署和创建 daemonset 之前，Trident 会查找 `ResourceQuota`对象，如果未发现，则应用它。

如果需要对默认 Resource Quota 和 Priority Class 进行更多控制，则可以使用 Helm chart 生成 `custom.yaml`或配置 `ResourceQuota`对象。

以下是优先处理 Trident daemonset 的 `ResourceQuota` 对象示例。

```
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 Resource Quotas 的详细信息，请参阅 "Kubernetes：资源配额"。

安装失败时清理 `ResourceQuota`

在创建 `ResourceQuota` 对象后安装失败的极少数情况下，请先尝试 "卸载"，然后重新安装。

如果这不起作用，请手动删除 `ResourceQuota` 对象。

删除 `ResourceQuota`

如果您希望控制自己的资源分配，则可以使用以下命令删除 Trident `ResourceQuota` 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) 定义权限级别并限制 Pod 的行为。OpenShift 安全上下文约束 (SCC) 同样定义了特定于 OpenShift Kubernetes 引擎的 Pod 限制。为了提供这种定制，Trident 在安装期间启用某些权限。以下各节详细介绍 Trident 设置的权限。

PSS 取代了 Pod Security Policies (PSP)。PSP 在 Kubernetes v1.21 中被弃用，并将在 v1.25 中删除。有关详细信息，请参阅 "Kubernetes：安全"。

必需的 Kubernetes 安全上下文和相关字段

权限	说明
特权	CSI 要求挂载点为双向，这意味着 Trident 节点 Pod 必须运行特权容器。有关详细信息，请参阅 "Kubernetes：挂载传播"。
主机网络	iSCSI 守护程序所需。 `iscsiadm`管理 iSCSI 挂载并使用主机网络与 iSCSI 守护程序通信。
主机 IPC	NFS 使用进程间通信 (IPC) 与 NFSD 通信。
主机 PID	需要启动 `rpc-statd` 以用于 NFS。Trident 查询主机进程以确定 `rpc-statd`是否正在运行，然后再挂载 NFS 卷。

功能	该 `SYS_ADMIN`功能是特权容器默认功能的一部分。例如，`Docker` 为特权容器设置这些功能：
	`CapPrm: 0000003fffffffff`
	`CapEff: 0000003fffffffff`

Seccomp	Seccomp 配置文件在特权容器中始终为"Unconfined"；因此，它不能在 Trident 中启用。
SELinux	在 OpenShift 上，特权容器在 `spc_t`（"Super Privileged Container"）域中运行，非特权容器在 `container_t`域中运行。在 `containerd`上，安装 `container-selinux`后，所有容器都在 `spc_t`域中运行，这有效地禁用了 `SELinux`。因此，`Trident` 不会向容器添加 `seLinuxOptions`。
DAC	特权容器必须以 root 身份运行。非特权容器以 root 身份运行，以访问 CSI 所需的 unix 套接字。

Pod 安全标准 (PSS)

标签	说明	默认
`pod-security.kubernetes.io/enforce pod-security.kubernetes.io/enforce-version`	允许 Trident 控制器和节点进入安装命名空间。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 未被调度，出现"Error creating: …"或"Warning: trident-csi-…"。如果发生这种情况，请检查 `privileged` 的命名空间标签是否已更改。如果是，请重新安装 Trident。

Pod 安全策略 (PSP)

字段	说明	默认
allowPrivilegeEscalation	特权容器必须允许特权提升。	true
allowedCSIDrivers	Trident 不使用内联 CSI 短暂卷。	空
allowedCapabilities	非特权 Trident 容器不需要比默认设置更多的功能，特权容器被授予所有可能的功能。	空
allowedFlexVolumes	Trident 不使用 "FlexVolume 驱动程序"，因此它们不包括在允许的卷列表中。	空
allowedHostPaths	Trident 节点 Pod 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 不需要任何不安全的 sysctls。	空
defaultAddCapabilities	特权容器中不需要添加任何功能。	空
defaultAllowPrivilegeEscalation	允许权限提升在每个 Trident pod 中处理。	false
forbiddenSysctls	不允许 sysctls。	空
fsGroup	Trident 容器以 root 身份运行。	RunAsAny
hostIPC	装载 NFS 卷需要主机 IPC 与 `nfsd` 进行通信	true
hostNetwork	iscsiadm 要求主机网络与 iSCSI 后台程序进行通信。	true
hostPID	需要主机 PID 来检查 `rpc-statd`是否在节点上运行。	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 运行特权容器，不能丢弃功能。	none
runAsGroup	Trident 容器以 root 身份运行。	RunAsAny
runAsUser	Trident 容器以 root 身份运行。	runAsAny
runtimeClass	Trident 不使用 RuntimeClasses。	空

字段	说明	默认
seLinux	Trident 没有设置 seLinuxOptions，因为当前容器运行时和 Kubernetes 发行版处理 SELinux 的方式存在差异。	空
supplementalGroups	Trident 容器以 root 身份运行。	RunAsAny
volumes	Trident Pod 需要这些卷插件。	hostPath, projected, emptyDir

安全上下文约束 (SCC)

标签	说明	默认
allowHostDirVolumePlugin	Trident 节点 Pod 挂载节点的根文件系统。	true
allowHostIPC	装载 NFS 卷需要主机 IPC 与 `nfsd` 进行通信。	true
allowHostNetwork	iscsiadm 要求主机网络与 iSCSI 后台程序进行通信。	true
allowHostPID	需要主机 PID 来检查 `rpc-statd`是否在节点上运行。	true
allowHostPorts	Trident 不使用任何主机端口。	false
allowPrivilegeEscalation	特权容器必须允许特权提升。	true
allowPrivilegedContainer	Trident 节点 Pod 必须运行特权容器才能挂载卷。	true
allowedUnsafeSysctls	Trident 不需要任何不安全的 sysctls。	none
allowedCapabilities	非特权 Trident 容器不需要比默认设置更多的功能，特权容器被授予所有可能的功能。	空
defaultAddCapabilities	特权容器中不需要添加任何功能。	空
fsGroup	Trident 容器以 root 身份运行。	RunAsAny
groups	此 SCC 特定于 Trident 并绑定到其用户。	空
readOnlyRootFilesystem	Trident 节点 Pod 必须写入节点文件系统。	false
requiredDropCapabilities	Trident 节点 Pod 运行特权容器，不能丢弃功能。	none
runAsUser	Trident 容器以 root 身份运行。	RunAsAny

标签	说明	默认
`seLinuxContext`	Trident 没有设置 `seLinuxOptions`，因为当前容器运行时和 Kubernetes 发行版处理 SELinux 的方式存在差异。	空
`seccompProfiles`	特权容器始终运行 "Unconfined"。	空
`supplementalGroups`	Trident 容器以 root 身份运行。	`RunAsAny`
`users`	提供了一个条目来将此 SCC 绑定到 Trident 命名空间中的 Trident 用户。	不适用
`volumes`	Trident Pod 需要这些卷插件。	`hostPath, downwardAPI, projected, emptyDir`

法律声明

法律声明提供对版权声明、商标、专利等信息的访问。

版权

"https://www.netapp.com/company/legal/copyright/"

商标

NETAPP、NETAPP 标识和 NetApp 商标页面上所列的商标是 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

当前 NetApp 拥有的专利列表可以在以下网址找到：

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 Trident 的 NetApp 软件中使用的第三方版权和许可证 https://github.com/NetApp/trident/。