Trident 26.10 文档

Trident

NetApp
November 19, 2026

This PDF was generated from https://docs.netapp.com/zh-cn/trident/index.html on November 19, 2026.
Always check docs.netapp.com for the latest.

目录

Trident 26.10 文档 1
发行说明 2
新增功能 2
26.10.1 中的新增功能 2
26.10 中的新增功能 2
26.06.2 中的新增功能 5
26.06.1 中的新增功能 6
26.06 中的新增功能 6
26.02.1 中的新增功能 9
26.02 中的新增功能 10
25.10 中的新增功能 12
25.06.2 中的变更 14
25.06.1 中的变更 14
25.06 中的变化 14
25.02.1 中的变化 16
25.02中的变化 16
24.10.1中的变更 18
24.10中的变化 18
24.06中的变化 20
24.02中的变化 21
23.10中的变化 21
23.07.1中的变更 22
23.07中的变更 22
23.04年度的变动 23
23.01.1中的变更 24
2001年23月的变更 24
22.10中的更改 25
22.07中的更改 26
22.04中的更改 27
22.01.1 中的更改 27
22.01.0 中的更改 28
21.10.1 中的更改 28
21.10.0 中的更改 29
已知问题 29
了解更多信息 30
文档的早期版本 30
NetApp Trident 支持 ONTAP ASA r2 存储系统 31
支持的操作 31
不支持的操作 31

已知问题 32
从 Trident 22.10 或更早版本升级后，卷仍保持连接状态 32
VolumeSnapshots 未到达 ReadyToUse 状态 32
还原大型文件的Restic备份可能会失败 32
入门 33
了解Trident 33
了解Trident 33
Trident架构 34
概念 37
Trident快速入门 40
下一步是什么? 41
Trident要求 41
有关Trident的关键信息 41
支持的前端（编排程序） 42
支持的后端（存储） 42
Trident 支持 KubeVirt、OpenShift 虚拟化和 SUSE 虚拟化 43
Trident 支持 Kubernetes 上的 Kubernetes 和 OpenShift 托管控制平面 43
功能要求 44
已测试主机操作系统 45
主机配置 45
存储系统配置： 45
Trident端口 45
容器映像以及相应的 Kubernetes 版本 45
安装 Trident 47
了解Trident安装 47
有关 Trident 26.10 的关键信息 47
开始之前 47
选择安装方法 47
选择安装模式 49
根据您的方法和模式选择过程 49
在安装方法之间移动 50
其他已知配置选项 50
使用Trident操作员安装 50
手动部署Trident操作员(标准模式) 50
手动部署Trident操作员(脱机模式) 55
使用Helm部署Trident操作员(标准模式) 60
使用Helm部署Trident操作员(脱机模式) 67
自定义Trident操作员安装 74
使用tridentctl进行安装 84
使用trdentcl安装Trident 84
自定义 Trident tridentctl 安装 88

使用OpenShift认证操作员进行安装 88
使用OpenShift OperatorHub安装Trident 88
切换到 OpenShift 认证的 Trident 操作员 92
使用Trident 96
准备 Trident Worker 节点 96
选择合适的工具 96
节点服务发现 96
NFS volumes 97
iSCSI 卷 97
NVMe/TCP卷 101
基于FC卷的SCSI 101
准备配置SMB卷 103
使用 COSI 配置对象存储 104
概述 105
要求和限制 105
启用 COSI 105
配置 StorageGRID 对象存储 106
配置 ONTAP S3 对象存储 107
创建 BucketClass 并请求存储桶 107
授予对存储桶的工作负载访问权限 108
配置和管理后端 109
配置 Trident 后端 109
Azure NetApp Files 109
Google Cloud NetApp卷 125
配置 NetApp HCI 或 SolidFire 后端 161
ONTAP SAN 驱动程序 165
ONTAP NAS 驱动程序 191
使用 ONTAP 后端的均衡放置 221
适用于 NetApp ONTAP 的 Amazon FSX 224
使用 kubectl 创建后端 256
管理后端 262
创建和管理存储类 271
创建 Trident 存储类 271
管理 Trident 存储类 273
配置和管理卷 275
预配 Trident 卷 275
移动卷 277
扩展 Trident 卷 280
了解 RWX NVMe 子系统限制 290
控制器可扩展性 291
自动卷扩展 296

配置 ONTAP 卷自动调整大小 303
管理 Autogrow 策略 306
导入 Trident 卷 315
自定义 Trident 卷名和标签 326
跨命名空间共享 Trident NFS 卷 328
跨命名空间克隆 Trident 卷 332
使用SnapMirror复制卷 334
使用 CSI 拓扑 340
使用 Trident 快照 347
使用 Trident 卷组快照 354
管理和监控Trident 359
升级Trident 359
升级Trident 359
使用操作员升级 360
使用 tridentctl 升级 Trident 364
使用trdentctrd管理Trident 365
命令和全局标志 365
命令选项和标志 366
插件支持 371
监控 Trident 存储操作 371
概述 371
第 1 步：定义 Prometheus 目标 372
第 2 步：创建 Prometheus ServiceMonitor 372
第 3 步：使用 PromQL 查询 Trident 指标 373
了解Trident AutoSupport遥测 375
禁用Trident指标 375
卸载 Trident 376
确定原始安装方法 376
卸载TRIdent操作员安装 376
卸载 `tridentctl` 安装 377
适用于Docker的Trident 378
部署的前提条件 378
验证要求 378
NVMe工具 380
FC工具 380
部署Trident 382
Docker 托管插件方法（ 1.13/17.03 及更高版本） 382
传统方法（ 1.12 或更早版本） 384
在系统启动时启动 Trident 385
升级或卸载 Trident 386
升级 386

卸载 388
使用卷 388
创建卷 388
删除卷 389
克隆卷 389
访问外部创建的卷 390
驱动程序专用的卷选项 390
收集日志 394
收集日志以进行故障排除 394
一般故障排除提示 395
管理多个Trident实例 396
Docker 托管插件（ 1.13/17.03 或更高版本）的步骤 396
传统（ 1.12 或更早版本）的步骤 396
存储配置选项 397
全局配置选项 397
ONTAP 配置 398
Element 软件配置 404
已知问题和限制 406
将 Trident Docker 卷插件从旧版本升级到 20.10
及更高版本会导致升级失败，并且不会显示此类文件或目录错误。 406
卷名称的长度必须至少为 2 个字符。 406
Docker Swarm的某些行为会使Trident无法为每个存储和驱动程序组合提供支持。 406
如果要配置 FlexGroup ，则在第二个 FlexGroup 具有一个或多个与要配置的 FlexGroup
相同的聚合时， ONTAP 不会配置第二个 FlexGroup 。 406
最佳实践和建议 407
Trident 部署建议 407
部署到专用命名空间 407
使用配额和范围限制来控制存储消耗 407
Trident 存储配置的最佳实践 407
平台概述 407
ONTAP 和 Cloud Volumes ONTAP 最佳实践 407
SolidFire 最佳实践 411
如何查找更多信息 412
将 Trident 与 Kubernetes 集成 413
驱动程序选择和部署 413
存储类设计 415
虚拟池设计 416
卷操作 417
指标服务 420
Trident 数据保护和灾难恢复 421
Trident复制和恢复 421

SVM复制和恢复 422
卷复制和恢复 423
Snapshot数据保护 423
使用Trident实现有状态应用程序的故障转移自动化 423
有关强制断开的详细信息 423
有关自动故障转移的详细信息 424
安全性 428
Trident 安全建议 428
Trident Linux 统一密钥设置（LUKS） 429
修复 RWX Linux 统一密钥设置（LUKS）卷的停滞租约 435
Trident Kerberos 传输中加密 437
使用Trident Protect 保护应用程序 445
了解Trident Protect 445
下一步是什么? 445
安装Trident Protect 445
Trident保护要求 445
安装并配置Trident Protect 448
安装Trident Protect CLI 插件 451
定制Trident Protect 安装 454
管理Trident Protect 458
管理Trident Protect 授权和访问控制 459
监控Trident保护资源 464
生成Trident Protect 支持包 469
升级Trident保护 470
管理和保护应用程序 472
使用Trident Protect AppVault 对象来管理存储桶。 472
加密 Trident Protect 快照和备份的 AppVault 元数据 483
使用Trident Protect 定义管理应用程序 489
使用Trident Protect 保护应用程序 493
将卷组快照与 Trident Protect 结合使用 502
还原应用程序 507
使用NetApp SnapMirror复制应用程序 527
使用Trident Protect 迁移应用程序 542
使用 Trident Protect 保护 KubeVirt 虚拟机 546
管理Trident Protect 执行钩子 556
卸载Trident Protect 563
Trident和Trident Protect 博客 565
Trident博客 565
Trident Protect博客 565
知识和支持 567
常见问题解答 567

一般问题 567
在Kubbernetes集群上安装和使用Trident 567
故障排除和支持 568
升级Trident 569
管理后端和卷 569
故障排除 573
常规故障排除 573
使用操作员无法成功部署TRIdent 575
使用无法成功部署TRIYent `tridentctl` 576
完全删除Trident和CRD 577
使用Kubnetes 1.26上的rwx原始块命名区卸载NVMe节点失败 577
当预期启用“v4.2-xattrs”时，NFSv4.2 客户端在升级ONTAP后报告“无效参数” 578
支持 578
Trident支持生命周期 578
自助支持 579
社区支持 579
NetApp技术支持 579
有关详细信息 … 579
参考 580
Trident端口 580
概述 580
Trident REST API 582
何时使用REST API 582
使用REST API 582
命令行选项 582
日志记录 583
Kubernetes 583
Docker 583
REST 583
Kubernetes 和 Trident 对象 584
对象如何相互交互? 584
Kubernetes `PersistentVolumeClaim` 对象 584
Kubernetes `PersistentVolume` 对象 586
Kubernetes `StorageClass` 对象 586
Kubernetes `VolumeSnapshotClass` 对象 589
Kubernetes `VolumeSnapshot` 对象 589
Kubernetes `VolumeSnapshotContent` 对象 590
Kubbernetes `VolumeGroupSnapshotClass`对象 590
Kubbernetes `VolumeGroupSnapshot`对象 590
Kubbernetes `VolumeGroupSnapshotContent`对象 591
Kubernetes `CustomResourceDefinition` 对象 591

Trident `StorageClass`对象 . . . 591
Trident 后端对象 . . . 592
Trident `StoragePool`对象 . . . 592
Trident `Volume`对象 . . . 592
Trident `Snapshot`对象 . . . 593
Trident `ResourceQuota`对象 . . . 594
POD安全标准(PSS)和安全上下文限制(SCC) . . . 595
所需的Kubernetes安全上下文和相关字段 . . . 595
POD安全标准(PSS) . . . 595
POD安全策略(PSP) . . . 596
安全上下文限制(SCC) . . . 597
法律声明 . . . 599
版权 . . . 599
商标 . . . 599
专利 . . . 599
隐私政策 . . . 599
开放源代码 . . . 599

Trident 26.10 文档

发行说明

新增功能

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

> 安装程序 zip 文件中提供的 `tridentctl` 二进制文件是经过测试且受支持的版本。请注意， zip 文件的 ` 或 Extras` 部分中提供的 `macos` 二进制文件未经过测试或不受支持。

26.10.1 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

修复

- **Kubernetes**：
  - 修复了从 Trident 25.10 或更早版本升级后，Trident 节点 pod 在启动时强制分离正在使用的卷，导致 pod 失败并提示 `volume … is not staged`的问题。

26.10 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

> 请勿升级到 Trident 26.10.0。在启动过程中，Trident 节点 Pod 可能会分离仍在使用的卷。受影响的 Pod 可能会失败并出现错误 `volume … is not staged`。此问题已在 Trident 26.10.1 中得到修复。请改为升级到 Trident 26.10.1 或更高版本。

Trident

增强功能

- **Kubernetes**：
  - 添加了对从 StorageGRID 基于 COSI 动态配置对象存储分段的支持。有关详细信息，请参见 "使用 COSI 配置对象存储"。
  - 添加了对定期刷新 NVMe 数据 LIF 的支持，以检测和适应配置更改。有关详细信息，请参见 "刷新 NVMe 数据 LIF"。
  - 添加了对由 ONTAP-SAN 和 ONTAP-SAN-Economy 驱动程序配置的新卷上的 ONTAP 自动调整大小的支持。有关详细信息，请参见 "配置 ONTAP 卷自动调整大小"。
  - 增加了对 Kubernetes 1.37 的支持。有关支持的版本和所需的映像，请参见 "Trident要求"。
  - 为 CommVault 和 Rubrik 备份添加了备份克隆 PVC 识别，以跳过 ONTAP 卷恢复队列（问题 "#1182" ）。
  - 为新的 NVMe RWX 卷添加了 `maxNamespacesPerSuperSubsystem`后端选项。有关详细信息，请参见"NVMe RWX 子系统限制"。

- 通过 `TridentNode` 规范和状态增加了基于 CRD 的 CSI 节点注册，替换了 Kubernetes 部署的 HTTPS 控制器反向通道。
- 添加了安装程序支持，以便在 OpenShift 上为 Google Cloud Workload Identity Federation (WIF) 投影具有可配置受众的服务帐户令牌。有关安装和后端配置的详细信息，请参见 "使用 Workload Identity Federation 配置 Google Cloud NetApp Volumes"。
- 在 `trident_backend_info`指标中添加了 `backend_state`标签，以在 Prometheus 中标识各个后端的运行状况（Issue "#1088"）。有关 Trident 指标的信息，请参见"监控Trident"。

- 添加了对通过 iSCSI 由 ONTAP-SAN 驱动程序配置的 RWX 卷的 LUKS 加密的支持。有关详细信息，请参见 "使用Linux统一密钥设置(LUKS)"。
- 为 ONTAP-NAS-FlexGroup 驱动程序和 Docker 添加了 GA 并发支持；ASA r2 并发现在是 GA，已从 Technical Preview 毕业。有关支持的驱动程序和行为，请参见"控制器可扩展性"。
- 通过使用路径删除和 iSCSI 自我修复，增加了在并非所有 iSCSI 路径都可用时对 CSI 卷扩展的支持。有关详细信息，请参见"展开卷"。
- 添加了 `trident.netapp.io/unixGroupID` PVC 注释，以便在创建具有 root squash 的 ONTAP-NAS 和 ONTAP-NAS-FlexGroup 卷时设置 ONTAP 卷 UNIX 组 ID（Issue "#1079"）。
- 添加了对 NVMe/TCP、NFSv4 和 iSCSI（问题 "#806"）的 Talos OS 支持。有关受支持的平台和工作节点要求，请参见 "Trident要求" 和 "准备工作节点"。
- 添加了可选 `storageAPITimeout` 后端参数以配置每个后端的 ONTAP REST 和 ZAPI 客户端超时。
- 添加了 `useRESTSQLBypass` ONTAP 后端选项，以将 `SQL=false`添加到 ONTAP REST 名称查找中。
- 已将 ONTAP-NAS-Economy qtree 导入从 Technical Preview 提升为 GA。有关导入说明，请参见 "导入 Trident 卷"。

修复

- **Kubernetes**：
  - 修复了当 `enableConcurrency` 为 `true` 时，在后端更新或控制器重新启动期间 ONTAP-NAS 导出策略规则被错误删除的问题（问题 "#1166"，"#1179"）。有关导出策略配置，请参见 "ONTAP NAS配置选项和示例"。
  - 修复了非实质性后端更新（例如凭据或 QoS 更改）触发每个卷的孤立扫描，从而可能阻塞大型后端上的 CSI 操作的问题。现在，扫描仅在 SVM 或存储前缀更改时运行。
  - 已修复出现在调试日志中的凭据，包括 CSI gRPC 请求转储、启动环境输出和已记录的 `TridentBackendConfig`规范。启动输出现在仅记录 `Trident` `TRIDENT_*` 变量的值，并按名称列出其他变量。
  - 修复了 CSI 发布和卷移动之间的竞争，这可能会使用过时的访问信息附加卷。有关卷移动的详细信息，请参见 "移动卷"。
  - 修复了 `NodeGetInfo` 在节点注册完成之前返回空拓扑标签的问题，这可能会导致拓扑感知配置失败（问题 "#1131"）。有关拓扑感知配置，请参见 "使用 CSI 拓扑配置存储"。
  - 修复了一个罕见的 NVMe 断开连接竞态条件，该竞态条件可能会在连接另一个 Pod 时拆除共享子系统。
  - 修复了在并发下克隆已挂载卷时 NVMe PVC 克隆发布失败和不正确的父命名空间映射的问题。
  - 修复了在配置了 `nameTemplate`的 ONTAP-SAN-Economy 后端上 PVC 的快照创建失败问题。有关后端配置示例，请参见 "ONTAP SAN配置选项和示例"。
  - 修复了如果在 REST 名称索引同步之前删除新创建的 FlexVol，ONTAP REST 后端会将其孤立的问题；Trident 现在通过 UUID 删除卷。

- 修复了无法访问后端 LIF 时控制器启动挂起的问题。受影响的后端被标记 `Failed`并继续启动。
- 修复了使用多键节点选择器配置时，Operator 管理的控制器和节点 Pod 在每次调谐时重新启动的问题。有关节点选择器配置，请参见 "自定义Trident操作员安装"。
- 修复了 TridentConfigurator 创建的 Azure NetApp Files `TridentBackendConfig` 资源缺少创建后验证所需的 `trident.netapp.io/configurator` 标签的问题。
- 修复了 Trident CSI 服务使用 IPv6 ClusterIP 时节点注册失败的问题（Issue "#1164"）。
- 修复了在过时的卷发布仍在取消发布时，如果工作节点被删除并重新加入，会导致 Trident 节点被永久移除的问题。
- 修复了当 Trident 的缓存导出策略过期时，ONTAP-NAS 卷在已附加的节点上丢失 NFS 导出访问权限的问题。有关导出策略配置，请参阅 "ONTAP NAS配置选项和示例"。
- 修复了 `tridentctl`安装清理会删除节点修复模板 CRD 的问题，这可能会阻止后续启用强制分离的安装。有关安装详细信息，请参见"使用trdentcl安装Trident"。
- 通过添加重试延迟，修复了快照删除调用导致 ONTAP API 不堪重负的问题。有关快照操作，请参见 "管理快照"。
- 修复了 Trident operator Helm chart 的 AutoSupport 镜像助手注释和 cloudProvider 值引用。
- 修复了针对繁忙或现有作业的 ONTAP ZAPI 卷创建处理，以便 CSI 重试不会过早销毁 FlexVols 或导致创建/销毁循环。
- 修复了重试不再存在的一致性组的卷组快照时出现的崩溃问题。有关卷组快照，请参见 "创建卷组快照"。
- 删除了基于支持的最低和最高 Kubernetes 版本的 Trident 支持警告。

- 修复了共享命名锁中的竞争条件，该竞争条件可能允许在同一资源上执行重叠操作。
- 修复了在找不到快照时 ASA 驱动程序初始化失败的问题。
- 修复了当集群节点不在主 HA 区域时，针对区域 Flex Unified 池的 GCNV 配置问题；Trident 不再将该 HA 区域视为必需的 CSI 拓扑。可用区池保持不变。有关 Flex Unified 配置，请参见 "配置块存储 (iSCSI)"。
- 修复了当规范后端标识与 Trident 卷名称不同时，GCNV Flex 统一 NAS 和 SAN 卷在后端协调期间被错误标记为孤立的问题（问题 "#1189"）。有关 GCNV 后端配置，请参见 "配置 Google Cloud NetApp Volumes"。
- 修复了传输内置于内核的主机上的 NVMe/TCP 检测（问题 "#1019"）。有关节点要求，请参见 "准备工作节点"。
- 修复了 ONTAP-NAS 和 ONTAP-NAS-FlexGroup 扩展在 ONTAP 卷已经足够大以满足请求（包括快照预留）时无限期重试的问题。有关卷扩展，请参见 "展开卷"。
- 修复了由于缺少连接路径而导致卷挂载失败的问题。有关 ONTAP NAS 配置，请参见 "配置 ONTAP NAS 驱动程序"。
- 修复了当 `storagePrefix`包含短划线时 ONTAP-SAN-Economy 卷创建失败的问题。有关后端配置选项，请参见 "ONTAP SAN配置选项和示例"。
- 修复了使用自动导出策略发布只读 ONTAP-NAS 克隆的问题，从而保留源 FlexVol 导出规则。有关 ONTAP NAS 配置，请参见"ONTAP NAS配置选项和示例"。
- 已修复 HTTP 传输重试行为（问题 "#1139"）。
- 修复了 ghost-device 检测跳过 SCSI 设备删除（问题 "#1161"）的问题。
- 已修复启用并发的配置选择所选节点的首选区域以外的拓扑区域的问题。有关并发和拓扑行为，请参见 "控制器可扩展性"。

- 修复了启用并发时等效后端之间卷放置不均衡的问题；现在会像非并发配置一样，随机均匀地选择后端。有关详细信息，请参见 "控制器可扩展性"。
- 修复了在 Trident 节点 pod 启动期间主机网络地址表发生更改时，节点注册时缺少 IP 地址（这可能会删除 NFS 导出策略规则）的问题。
- 修复了克隆创建在 REST 轮询之后继续进行并将 PVC 绑定为未拆分的 FlexClone 的问题；Trident 现在会观察到后续的创建作业失败。
- 已修复 ONTAP-SAN 卷扩展在节点上错误报告调整大小失败的问题。有关卷扩展，请参阅 "展开卷"。

试验性增强功能

不适用于生产环境。

- ［技术预览］ **Kubernetes**：添加了对 ONTAP S3 中基于 COSI 的对象存储桶动态配置的支持。有关详细信息，请参见 "使用 COSI 配置对象存储"。

Trident保护

增强功能

- 添加了对 AppVault 快照和备份元数据加密的支持。为每个 AppVault 使用自定义保管库密码。轮换保管库密码会重新加密元数据加密密钥，并更新 AppVault 中的每个 Kopia 和 Restic 存储库。要了解更多信息，请参阅 "加密 Trident Protect 快照和备份的 AppVault 元数据"。
- 添加了在还原期间保留 KubeVirt 虚拟机 MAC 地址的支持，这在应用程序许可证绑定到 MAC 地址时非常有用。要启用它，请将 protect.trident.netapp.io/retain-mac-address 注释设置为“true”。要了解更多信息，请参阅 "使用高级Trident Protect 恢复设置"。
- 添加了对卷组快照的支持，该快照可在同一时间点捕获应用程序的相关持久卷声明。通过标签、KubeVirt 虚拟机或两者，在应用程序自定义资源上的 `spec.volumeGroups`中定义卷组。要了解更多信息，请参阅"将卷组快照与 Trident Protect 结合使用"。
- 添加了对故障切换到已复制应用程序的早期快照的支持，允许从数据损坏或受损之前的某个时间点进行恢复。要了解更多信息，请参阅 "使用 Trident Protect 故障切换到较旧的快照"。

修复

- 修复了 Trident Protect 安装在 `trident-protect-pre-upgrade-hook`作业中失败的问题，原因是缺少 `trident-protect-controller-manager` ServiceAccount（"问题 #1154"）。
- 已修复仅限 PVC 的恢复，使其仅在所有者存在时才包含 `ownerReference`。

26.06.2 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

增强功能

修复

- **Kubernetes**：

- 修复了当 Trident 控制器重新启动或 `TridentBackendConfig`更新时，ONTAP-NAS 导出策略规则被静默删除的问题，该问题发生在 `enableConcurrency`设置为 `true`时。这可能导致已挂载的卷无法访问（"问题 #1166"、"问题 #1179"）。
- 修复了后端更新在发生非实质性更改（例如凭据和 QoS）时运行每卷孤立扫描的问题，这可能会在大型后端上阻塞 CSI 操作数小时。现在，扫描仅在 SVM 或存储前缀更改时运行。
- 修复了 Trident 调试日志中出现的凭据，包括 CSI gRPC 请求转储、启动环境转储以及 CRD 控制器记录的 `TridentBackendConfig`规范。启动环境转储现在仅记录 Trident 自身的 `TRIDENT_*`变量的值，并按名称列出其他变量。

26.06.1 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

修复

- **Kubernetes**：
  - 修复了 CSI 发布和卷移动之间的竞争问题，该问题可能导致使用过时的访问信息挂载卷。
  - 修复了在节点注册完成之前 NodeGetInfo 可能返回空拓扑标签，导致拓扑感知配置失败的问题。
  - 修复了一个罕见的 NVMe 断开连接竞态问题，该问题可能会在另一个 Pod 仍在连接时拆除共享子系统。
  - 修复了在并发情况下创建已挂载卷的克隆时 NVMe PVC 克隆发布失败（以及父命名空间映射不正确）的问题。
  - 修复了在 ONTAP-SAN-Economy 后端上使用 nameTemplate 的 PVC 快照创建失败的问题。

增强功能

- **Kubernetes**：
  - 在 ONTAP 后端配置上添加了可选 `storageAPITimeout`（秒），以设置每个后端的 REST 和 ZAPI API 客户端超时。

26.06 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

增强功能

- **Kubernetes**：
  - 添加了对 ONTAP-NAS-Economy 和 ONTAP-SAN-Economy 驱动程序并发的正式发布 (GA) 支持，将此功能从 Tech Preview 升级，并添加了对 Azure NetApp Files (ANF) 和 SolidFire 驱动程序的 GA 并发支持。
  - 增加了对 Kubernetes 1.36 的支持。
  - 添加了对 Google Cloud NetApp Volumes (GCNV) NAS 和 SAN 驱动程序的卷自动增长的正式发布 (GA) 支持。

- 添加了 Unified ONTAP 平衡放置功能，以便在配置新卷时进行最佳聚合选择（"问题 #64"）。
- 添加了对 ONTAP-NAS-Flexgroup 驱动程序的 ONTAP AFX 支持。
- 添加了与 NetApp Shift Toolkit 的集成，以便在使用 Red Hat OpenShift Migration Toolkit for Virtualization 时简化虚拟机迁移。
- 为 GCNV SAN 驱动程序添加了 Cloud Identity 支持。
- 添加了对 Trident Volume Publication (TVP) 传播和自动增长工作流的从属卷支持。
- 新增了在启用并发的情况下运行 Trident 时对卷更新和卷组快照的支持。
- 为 GCNV 添加了动态 gRPC 速率限制。
- 添加了控制器就绪性执行探针，并调整了控制器和 sidecar 的默认 Kubernetes API QPS 和 Burst 设置。
- 通过分离连接、加密设置和文件系统格式化步骤，改进了节点操作顺序。
- 将 Trident 容器根文件系统挂载为只读，以提高安全性。
- 在删除克隆卷的源快照时，为 ONTAP 后端中的克隆卷快照添加了尽力而为的快照删除功能。
- 添加了备份克隆 PVC 识别，以防止由 Veeam Kasten、Trilio 和 Cohesity 创建的临时 PVC 卡在 ONTAP 卷恢复队列中。
- 增强的 AWS ARN 处理以支持特殊区域。
- 防止 ONTAP-SAN-ECONOMY 驱动程序在调整大小操作期间 Flexvol 收缩。

试验性增强功能

不适用于生产环境。

- **[Tech Preview]**：新增对 ONTAP-NAS-Economy qtree 导入的支持。
- **[Tech Preview]**：增加了对 ASA-R2（SAN 和 NVMe）驱动程序并发性的支持。
- **[Tech Preview]**：为 ONTAP-SAN 和 ONTAP-NAS 驱动程序添加了 GCNV ONTAP 模式 HTTP 传输。

修复

- **Kubernetes**：
  - 修复了并发发布/取消发布、克隆、缓存和后端更新操作的竞争条件和状态处理。
  - 修复了 ONTAP-NAS 和 ONTAP-NAS-Economy 驱动程序中从属卷和只读克隆的导出策略竞争条件和并发发布/取消发布处理。
  - 修复了当快照策略不同时，在 MetroCluster 故障转移后进行 ONTAP-NAS-Economy FlexVol 发现的问题（"问题 #1082"）。
  - 修复了中断长时间运行的 qtree 删除的 ONTAP-NAS-Economy 卷删除重试（"问题 #1121"）。
  - 修复了 ONTAP-SAN 和 ONTAP-SAN-Economy 导入和调整大小的行为，包括 fsType 验证、卷元数据处理和自动增长模式行为。
  - 修复了并发卷创建过程中的 NVMe/TCP 命名空间竞争条件（"问题 #1089"）。
  - 修复了可用路径和门户不匹配时发生的 iSCSI 调整大小失败问题，并改进了重复调整大小的行为。
  - 修复了由卷繁忙条件导致的 ONTAP-SAN 卷删除失败问题。

- 修复了 ONTAP REST 错误处理，并改进了 SVM 状态超时行为。
- 修复了 GCNV SAN 分区池和主机组处理、规范化卷标识符转换以及低于最小大小的规范化创建请求。
- 修复了本机 GCNV 容量池选择，以排除 ONTAP 模式池。
- 修复了使用 Trident Automated Workload Failover 加速虚拟机故障转移的虚拟机的 KubeVirt 故障转移。
- 修复了 Operator 管理的安装 RBAC 问题，包括缺少节点 ClusterRoles 以及升级过程中 ClusterRole 和 ClusterRoleBinding 协调不正确的问题。这适用于使用 Trident Operator chart 的 Operator 和基于 Helm 的安装。
- 通过省略默认优先级值并纠正 YAML 模板缩进，修复了 TridentConfigurator 及相关 CRD 上持续存在的 GitOps 差异（"问题 #915"，"问题 #1120"）。
- 修复了在 Helm 升级期间修补 TridentVolumePublication CRD 的问题（"问题 #1151"）。
- 修复了 CSI 启动排序问题，现在在节点注册重试之前先启动 CSI gRPC 服务。
- 修复了并发操作期间安全 SMB 克隆配置叠加的问题。
- 修复了升级后 Trident 控制器 Pod 上缺少注释的问题（"问题 #1004"）。
- 改进了大型 SVM 上的慢速 qtree 配置（"问题 #1142"）。
- 已修复 ControllerPublish 以使用卷配置文件系统类型，而不是错误的默认文件系统类型。
- 修复了孤立 dm 设备可能导致重复 CSI NodeUnstageVolume 失败的问题。
- 改进了过时节点和多路径清理，在软删除节点清理后删除虚拟多路径残留项和过时的 TridentNode 资源。

Trident保护

增强功能

- 添加了 OpenShift Service CA Operator 集成，以自动化 Trident Protect 准入 Webhook 的 TLS 证书管理。通过在 Helm 值中将 `useOpenshiftServiceCA` 设置为 `true` 来启用。要了解更多信息，请参阅 "定制Trident Protect 安装"。
- 新增了对在 OpenShift Virtualization 上备份和恢复 KubeVirt VM 的支持。保护命名空间中的所有 VM 或按名称定位特定 VM，并选择性地从备份或快照中恢复单个 VM。要了解更多信息，请参见 "使用 Trident Protect 保护 KubeVirt 虚拟机"。
- 添加了对资源转换的支持，使您可以在还原过程中添加、删除和修改 Kubernetes 资源属性。要了解更多信息，请参阅 "使用自定义资源还原资源时修改资源"。
- 新增对 `storageClassMapping` 在 AppMirrorRelationship (AMR) 中的支持，在单个 AMR 定义中实现单个 PVC 的存储类映射。要了解更多信息，请参阅 "使用 Trident Protect 设置和管理应用程序复制"。

修复

- 修复了当目标集群上不存在源应用程序命名空间时，跨集群虚拟机还原失败并出现准入 webhook 错误的问题。
- 修复了由于与现有应用程序自定义资源的规范不匹配而导致在应用程序创建步骤失败的选择性虚拟机就地还原问题；现有应用程序现在将被重用。
- 修复了当虚拟机及其 PersistentVolumeClaim 均被删除时，虚拟机就地还原作业挂起在 `Running`状态的问题；缺少 PersistentVolumeClaim 现在将被视为致命错误。

- 已修复编辑现有保护计划错误地触发即时备份或快照运行的问题。
- 已修复快照终结器在应用程序自定义资源被删除后仍尝试获取该资源的问题；现在，当未找到应用程序时，终结器将跳过。
- 修复了在 PersistentVolumeClaim 复制步骤期间使用稀疏副本执行虚拟机就地恢复后，受保护存储容量显著增加的问题。
- 修复了 Trident Protect 在确认删除 ONTAP 中的后备快照之前删除 Kubernetes 快照对象的问题；终结器现在等待 `VolumeSnapshot`删除。
- 修复了当虚拟机冻结操作失败时快照无法继续的问题；冻结和解冻操作现在在失败时仍可继续执行。
- 修复了 KubeVirt 虚拟机应用程序在 AppMirrorRelationship 反向、反向重新同步和重新同步操作期间错误地重新创建 PersistentVolumeClaims 的问题。

26.02.1 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

修复

- **Kubernetes**：
- 修复了 Google Cloud NetApp Volumes (GCNV) SAN 行为，以支持针对块 (iSCSI) 工作负载的 Flex Unified ZONAL 池。现在支持分区和区域配置。
- 通过在 `InternalID`不存在时改进卷创建、验证、命名和回退卷查找操作，修复了 Google Cloud NetApp Volumes (GCNV) NAS UNIFIED 池处理问题。
- 修复了在可用路径和门户不匹配时 iSCSI 设备调整大小操作可能失败的问题。
- 修复了 `blkid` 错误识别 LUKS 设备的文件系统类型、阻止新卷创建的间歇性问题。
- 修复了 LUKS 大规模发布和暂存操作可能会阻止 Trident 控制器的问题。用户现在需要跟踪与卷关联的 LUKS 密码。
- 修复了 iSCSI 多路径分区可能显示为 ghost 设备，从而阻止 CSI `NodeUnstage` 操作成功的问题。
- 修复了并发卷创建过程中的 NVMe/TCP 命名空间竞争条件（问题 #1089）。
- 修复了在同一秒内执行快照时并发克隆快照操作的问题。
- 修复了阻止使用 `dataVolumes` 的 KubeVirt 虚拟机通过 Trident Automated Workload Failover 进行故障转移的问题。
- 修复了使用 Trident 并发核心时删除无效 `TridentMirrorRelationship`自定义资源时发生的错误。
- 修复了 `tridentactionmirrorupdates` Kubernetes CRD YAML 定义中的缩进（问题 #1120）。
- 修复了 REST qtree 异步删除 API 调用（Issue #1121）。
- 已修复 `ControllerPublish`使用卷配置文件系统类型而不是错误的默认文件系统类型的问题。
- 已修复控制器启动日志记录以正确报告并发驱动程序状态。
- 修复了并发 Trident 中暂停后端的处理。
- 增强的 AWS ARN 处理以支持特殊区域。

Trident保护

修复

更新 Trident Protect 依赖项以解决安全漏洞：

- gRPC-Go："CVE-2026-33186"
- Go x/crypto SSH："CVE-2024-45337"
- OpenTelemetry-Go："CVE-2026-24051"
- Go 运行时："CVE-2025-22871"
- 其他依赖项："CVE-2025-23109"

26.02 中的新增功能

了解 NetApp Trident 和 Trident Protect 的新功能，包括增强功能、修复和弃用。

Trident

增强功能

- **Kubernetes**：
  - 添加了对 ONTAP-NAS（仅限 NFS）、ONTAP-SAN（iSCSI、FCP、NVMe）和 Google Cloud NetApp Volumes (GCNV) 驱动程序并发性的一般可用性 (GA) 支持，使此功能从 Tech Preview 升级。有关详细信息，请参见"控制器可扩展性文档"。
  - 添加了对 Trident 基于自定义、用户定义的 Trident AutoGrow 策略的卷自动增长的支持。有关详细信息，请参见 "自动卷扩展文档"。
  - 增强 Trident 节点并发性，提高 NVMe 卷节点操作的可扩展性。有关详细信息，请参见 "NVMe 卷文档"。
  - 添加了对通过 PVC 注释(`tieringPolicy` `tieringMinimumCoolingDays`进行 GCNV NAS 卷自动分层的支持，包括池选择和克隆继承。有关详细信息，请参见"为 Google Cloud NetApp Volumes 文档配置自动分层"。
  - 添加了对使用 `google-cloud-netapp-volumes-san` 驱动程序的 GCNV SAN 块 (iSCSI) 卷的支持，包括调配、针对 LUN 访问的每节点主机组映射以及卷克隆操作。有关详细信息，请参见 "Google Cloud NetApp Volumes 块配置文档"。
  - 添加了对 Amazon FSx for NetApp ONTAP 的自动后端配置支持。当您创建包含所需参数的 StorageClass 时，Trident 会自动创建相应的后端和 VolumeSnapshotClass（如果需要）。有关更多信息，请参见 "Amazon FSx for NetApp ONTAP 后端配置文档"。
  - 添加了对不同 Microsoft Azure 云的支持，例如 Azure Government 和 Azure China，以及 Azure NetApp Files 后端的自定义云配置。有关详细信息，请参见 "Azure NetApp Files 后端配置文档"。
  - 增加了对 Kubernetes 1.35 的支持。有关详细信息，请参见 "需求文档"。

试验性增强功能

> 不适用于生产环境。

- **[Tech Preview]**：增加了对 ONTAP-NAS-Economy 和 ONTAP-SAN-Economy 驱动程序的并发支持。

已知问题

> `csi-snapshotter` sidecar 存在已知问题。在所有 Kubernetes 版本中，VolumeGroupSnapshots `v1beta1` 阻止 VolumeSnapshots 到达 `ReadyToUse`状态。

有两种解决方法可供选择：

1. 删除 VolumeGroupSnapshots CRD 以禁用 VolumeGroupSnapshots，然后重新安装 Trident。
2. 安装 VolumeGroupSnapshots `v1beta2` 和 `snapshot-controller` 版本 8.4.0 或更高版本，然后重新安装 Trident。VolumeGroupSnapshots 在 v1.34 之前的 Kubernetes 版本上不起作用。

修复

- **Kubernetes**：
  - 修复了在 ONTAP-NAS、ONTAP-NAS-Economy 和 Google Cloud NetApp Volumes 驱动程序中取消发布只读克隆从源卷删除导出策略规则的问题"问题 #1086"。
  - 将 kubectl 映像切换为基于 Alpine 的轻量级变体，以防止 Bitnami 公开映像弃用后的拉取失败 "问题 #1080"。
  - 修复了 Trident 升级期间保留现有部署的注释"问题 #1004"的问题。
  - 如果两个存储类都指向同一后端，则允许跨不同存储类进行克隆"问题 #1104"。
  - 修复了具有网络延迟的云环境中超时导致的节点准备故障。增加了基于云的安装的超时值。
  - 修复了创建 LUN 时导致文件系统类型属性在进程进入重试状态时保持未设置的问题。
  - 修复了 REST API 卷查找以忽略卷状态，从而防止在卷查询期间出现错误否定。
  - 大规模使用时，提高了 ontap-nas-economy 驱动程序的 Trident 控制器效率。
  - 在 ontap-san-economy 驱动程序中设置 LUN 导入期间的 internalID。
  - 增加 Azure Resource Graph 查询限制以处理更多子网。
  - 改进了 CSI 和 ONTAP 克隆拆分超时，以避免某些备份应用程序出现竞争情况"问题 #1098""问题 #1100"。
  - 修复了 LUKS 错误消息的抑制 "问题 #1069"。
  - 修复了对 iSCSI 和 NVMe 协议的过时 LUKS 映射器的处理。增强的清理逻辑可防止来自孤立设备映射器的装载失败。
- 已修复 RWX NVMe 卷的规模限制。
- 已更新 OpenTelemetry-Go 软件包以修复 "CVE-2026-24051"。

Trident保护

增强功能

- Trident Protect 现在会自动禁用保护计划，并在就地还原之前取消正在进行的操作，并在还原完成后重新启用它们。要了解更多信息，请参阅 "使用Trident Protect 恢复应用程序"。

- 将 `runImmediately` 字段添加到计划 CR 和 `--run-immediately` CLI 标志，以便在计划创建时触发即时备份或快照。要了解更多信息，请参阅"创建数据保护计划"。
- 添加了使用还原 CR 中的 `destinationApplicationName`字段或 `--destination-app-name` CLI 标志为还原的应用程序指定自定义名称的支持。要了解更多信息，请参阅"使用Trident Protect 恢复应用程序"。

修复

- 修复了在所需服务帐户可用之前创建 pod 导致的还原失败。
- 已修复在应用程序还原期间跳过 `Roles` 和 `RoleBindings` 的问题。
- 修复了尽管已正确配置，但未在 `tridentctl-protect get appvaultcontent` 输出中显示原始集群名称的问题。
- 修复了由于缺少 `pipefail` 处理而被忽略的 Kopia 恢复错误。
- 修复了由排除持久卷的资源筛选器导致的快照和备份失败。
- 修复了跨命名空间使用相同命名 PVC 的多命名空间应用程序中不正确的 PVC 还原，这可能会导致数据丢失。

25.10 中的新增功能

了解Trident和Trident Protect 的新特性，包括增强功能、修复和弃用。

Trident

增强功能

- **Kubernetes**：
  - 除了ONTAP -SAN（iSCSI 和 FC）驱动程序之外，还为ONTAP-NAS NFS 和ONTAP -SAN-Economy 驱动程序添加了对 CSI 卷组快照的支持，v1beta1 卷组快照 Kubernetes API。看"使用卷组快照"。
  - 为ONTAP-NAS 和ONTAP-NAS-Economy（NAS 驱动程序中不包括 SMB）以及ONTAP-SAN 和ONTAP-SAN-Economy 驱动程序添加了强制卷分离的自动工作负载故障转移支持。看"使用Trident实现有状态应用程序的故障转移自动化"。
  - 增强Trident节点并发性，以提高 FCP 卷节点操作的可扩展性。
  - 为ONTAP NAS 驱动程序添加了ONTAP AFX 支持。看"ONTAP NAS配置选项和示例"。
  - 增加了通过 TridentOrchestrator CR 和 Helm chart 值配置Trident容器的 CPU 和内存资源请求和限制的支持。（"第1000期"，"第927期"，"第853期"，"第592期"，"第110期"）。
  - 为 ASAr2 人格添加了 FC 支持。看"ONTAP SAN配置选项和示例"。
  - 新增了一个选项，可以使用 HTTPS 而不是 HTTP 来提供 Prometheus 指标。看"监控Trident"。
  - 添加了一个选项 `--no-rename`导入卷时，保留原始名称，但让Trident管理卷的生命周期。看"导入卷"。
  - Trident部署现在以系统集群关键优先级运行。
- 为Trident控制器添加了通过 helm、operator 和 tridentctl 使用主机网络的选项（"第858期"）。
- 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。

试验性增强功能

不适用于生产环境。

- **[技术预览]**：除了现有的ONTAP -SAN 驱动程序技术预览（统一ONTAP 9 中的 iSCSI 和 FCP 协议）之外，还增加了对ONTAP-NAS（仅限 NFS）和ONTAP -SAN（统一ONTAP 9 的 NVMe）的并发支持。

修复

- **Kubernetes**：
  - 通过将 Linux DaemonSet 标准化为 node-driver-registrar 来修复 CSI node-driver-registrar 容器名称不一致的问题，使其与 Windows DaemonSet 和容器镜像命名保持一致。
  - 修复了旧版 qtree 的导出策略未正确升级的问题。
- **Openshift**：
  - 修复了由于 SCC 将 allowHostDirVolumePlugin 设置为 false 而导致的 Openshift 中 Windows 节点上的Trident节点 pod 无法启动的问题（"第950期"）。
- 修复了 Kubernetes API QPS 无法通过 Helm 设置的问题（"第975期"）。
- 修复了无法在同一个 Kubernetes 节点上基于 NVMe XFS 文件系统 PVC 的快照挂载持久卷声明 (PVC) 的问题。
- 修复了在 NDVP 模式下主机/Docker 重启后 UUID 更改的问题，方法是为每个后端添加唯一/共享的子系统名称（例如 netappdvp_subsystem）。
- 修复了Trident从 23.10 之前的版本升级到 24.10 及以上版本期间 iSCSI 卷的挂载错误，解决了“无效的 SANType”问题。
- 修复了Trident后端状态在不重启Trident控制器的情况下无法转换到在线/离线状态的问题。
- 修复了导致PVC尺寸调整缓慢的间歇性竞争条件。
- 修复了卷克隆失败时快照未被清理的问题。
- 修复了内核更改卷的设备路径时无法取消暂存卷的问题。
- 修复了由于 LUKS 设备已关闭而导致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 超时时间来检查Trident版本。

Trident保护

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

增强功能

- 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`

看"支持的备份和计划注释"。

- 在计划 CR 中添加了一个注释，用于配置 PVC 绑定超时，该超时将由备份 CR 使用：`protect.trident.netapp.io/pvc-bind-timeout-sec` 。看"支持的备份和计划注释"。
- 改进 `tridentctl-protect`备份和快照列表新增一个字段，用于指示执行钩子失败。

25.06.2 中的变更

Trident

修复

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

25.06.1 中的变更

Trident

> 对于使用 SolidFire 的客户，由于取消发布卷时出现已知问题，请勿升级到 25.06.1。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 设备导致无法分离卷的问题。
  - 修复了当设备路径自创建以来发生变化时 iSCSI 卷取消暂存的问题。
  - 阻止跨存储类的卷克隆。
- **OpenShift**：修复了 OCP 4.19 中 iSCSI 节点准备失败的问题。
- 增加了使用SolidFire后端克隆卷时的超时时间（"问题描述#1008" ）。

25.06 中的变化

Trident

增强功能

- **Kubernetes**：
  - 增加了对 CSI 卷组快照的支持 `v1beta1`适用于 ONTAP-SAN iSCSI 驱动程序的卷组快照 Kubernetes API。请参阅。 "使用卷组快照"

    > VolumeGroupSnapshot 是 Kubernetes 中的一个 Beta 功能，包含 Beta 版 API 。VolumeGroupSnapshot所需的最低版本为 Kubernetes 1.32。

  - 除了 iSCSI 之外，还增加了对ONTAP ASA r2 的 NVMe/TCP 支持。看"ONTAP SAN配置选项和示例"。

- 为 ONTAP-NAS 和 ONTAP-NAS-Economy 卷添加了安全的 SMB 支持。ActiveDirectory 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 Trident 节点并发性，以提高 iSCSI 卷节点操作的可扩展性。
- 额外 `--allow-discards`打开 LUKS 卷时允许丢弃/TRIM 命令以回收空间。
- 格式化 LUKS 加密卷时的性能增强。
- 增强了对失败但部分格式化的 LUKS 设备的 LUKS 清理。
- 增强了 Trident 节点幂等性，用于 NVMe 卷的连接和分离。
- 额外 `internalID`字段到 ONTAP-SAN-Economy 驱动程序的 Trident 卷配置。
- 增加了对使用 SnapMirror 对 NVMe 后端进行卷复制的支持。请参阅。 "使用SnapMirror复制卷"

试验性增强功能

> 不适用于生产环境。

- [技术预览] 通过以下方式启用并发 Trident 控制器操作 `--enable-concurrency`功能标志。这允许控制器操作并行运行，从而提高繁忙或大型环境中的性能。

  > 此功能尚处于实验阶段，目前支持使用 ONTAP-SAN 驱动程序（iSCSI 和 FCP 协议）的有限并行工作流程。

- [技术预览] 使用 ANF 驱动程序添加了手动 QOS 支持。

修复

- **Kubernetes**：
  - 修复了 CSI NodeExpandVolume 的问题，当底层 SCSI 磁盘不可用时，多路径设备可能会出现大小不一致的情况。
  - 修复了无法清理 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`，则使用主机的默认 NFS 版本（NFSv3 或 NFSv4）。
  - 修复了使用 Kustomize 安装 Trident 时出现的部署问题（ "问题描述#831" ）。
  - 修复了从快照创建的 PVC 缺少导出策略的问题（ "问题描述#1016" ）。
  - 修复了 ANF 卷大小未自动与 1 GiB 增量对齐的问题。
  - 修复了将 NFSv3 与 Bottlerocket 结合使用时的问题。
- 修复了 ONTAP-NAS-Economy 卷尽管调整大小失败但仍可扩展至 300 TB 的问题。
- 修复了使用 ONTAP REST API 时克隆拆分操作同步完成的问题。

弃用：

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

Trident保护

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

增强功能

- 改善了恢复时间，提供了进行更频繁的完整备份的选项。
- 通过 Group-Version-Kind (GVK) 过滤提高了应用程序定义和选择性恢复的粒度。
- 将 AppMirrorRelationship (AMR) 与 NetApp SnapMirror 结合使用时可实现高效的重新同步和反向复制，以避免完整的 PVC 复制。
- 增加了使用 EKS Pod Identity 创建 AppVault 存储桶的功能，无需使用 EKS 集群的存储桶凭证指定机密。
- 如果需要，添加了在恢复命名空间中跳过恢复标签和注释的功能。
- AppMirrorRelationship (AMR) 现在将检查源 PVC 扩展并根据需要对目标 PVC 执行适当的扩展。

修复

- 修复了先前快照的快照注释值被应用到新快照的错误。现在所有快照注释均已正确应用。
- 如果未定义，则默认定义数据移动器加密（Kopia / Restic）的秘密。
- 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 被排除在恢复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计划，导致快照被过早删除。

25.02.1 中的变化

Trident

修复

- **Kubernetes**：
  - 修复了使用非默认图像注册表()时Trident运算符错误填充的侧向图像名称和版本的问题"问题描述#983"。
  - 修复了在ONTAP故障转移恢复()期间多路径会话无法恢复的问题"问题描述#961"。

25.02中的变化

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

Trident

增强功能

- **Kubernetes**：

- 增加了对适用于iSCSI的ONTAP ASA R2的支持。
- 增加了对在非正常节点关闭情况下强制断开ONTAP NAS卷的支持。现在、新的ONTAP NAS卷将使用Trident管理的每个卷导出策略。为现有卷提供了一个升级路径、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cloneFronSnapshot标注。
- 增加了对跨命名空间卷克隆的支持。
- 增强了iSCSI自我修复扫描修复功能、可按确切的主机、通道、目标和LUN ID启动重新扫描。
- 增加了对Kubnetes 1.32的支持。

- **OpenShift**：
  - 增加了对ROSA集群上的RHCOS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驱动程序上增加了光纤通道支持。
- 增加了NVMe Ks支持。
- 已切换到所有基本映像的暂存映像。
- 添加了iSCSI连接状态发现和日志记录(iSCSI会话应登录但不应登录)"问题描述#961"。
- 通过gosle-Cloud NetApp卷驱动程序增加了对SMB卷的支持。
- 增加了对允许ONTAP卷在删除时跳过恢复队列的支持。
- 增加了对使用SHA (而不是标记)覆盖默认图像的支持。
- 已将image-pull机密标志添加到trdentctrd安装程序中。

修复

- **Kubernetes**：
  - 修复了自动导出策略()中缺少的节点IP地址"问题描述#965"。
  - 修复了自动导出策略在ONTAP NAS经济环境中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"问题描述#913"()。
  - 增加了在Trident运算符中禁用自动配置器协调的选项"问题描述#924"()。
  - 增加了CSI -非标准容器的SecurityContext"问题描述#976" ()。

Trident保护

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

增强功能

- 为 KubeVirt / OpenShift 虚拟化虚拟机添加了备份和恢复支持，支持 volumeMode: File 和 volumeMode: Block（原始设备）存储。此支持与所有Trident驱动程序兼容，并且在使用NetApp SnapMirror和Trident Protect 复制存储时增强了现有的保护功能。
- 增加了在Kubevirt环境的应用程序级别控制冻结行为的功能。

- 增加了对配置AutoSupport代理连接的支持。
- 增加了为数据移动程序加密(Kerberos / Restic)定义密钥的功能。
- 增加了手动运行执行挂钩的功能。
- 增加了在Trident Protect 安装过程中配置安全上下文约束 (SCC) 的功能。
- 增加了在Trident Protect 安装过程中配置 nodeSelector 的支持。
- 增加了对AppVault对象的HTTP/HTTPS出口代理的支持。
- 扩展的ResourceFilter可用于排除集群范围的资源。
- 增加了对S3 AppVault凭据中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、可跳过ONTAP卷恢复队列。
- SCC标注现在恢复为原始值。
- 通过支持并行操作提高了恢复效率。
- 增强了对大型应用程序执行挂接超时的支持。

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(iSCSI会话应登录但不应登录)"问题描述#961"。

修复

- 修复了自动导出策略()中缺少的节点IP地址"问题描述#965"。
- 修复了自动导出策略在ONTAP NAS经济环境中过早切换到每个卷策略的问题。
- 更新了Trident和Trident CVE-ASUP的依赖关系、以解决CVE-2024-45337和CVE-2024-45310。
- 删除了在iSCSI自我修复期间因非CHAP门户间歇性运行状况不正常而注销的问题"问题描述#961"()。

24.10中的变化

增强功能

- Google Cloud NetApp卷驱动程序现已广泛适用于NFS卷、并支持区域感知型配置。
- GCP工作负载标识将用作具有GKE的Google Cloud NetApp卷的云标识。
- 向ONTAP－SAN和LUN－SAN－Economy驱动程序中添加了 `formatOptions`配置参数，以允许用户指定ONTAP格式选项。
- 已将Azure NetApp Files最小卷大小减少到50 GiB。Azure新的最小大小预计将于11月全面上市。
- 添加了 `denyNewVolumePools`配置参数、用于将ONTAP－NAS－经济型和ONTAP－SAN经济型驱动程序

限制为原有的FlexVol池。

- 增加了对在所有ONTAP驱动程序中从SVM添加、删除或重命名聚合的检测功能。
- 向 LUKS LUN 添加了 18 MiB 开销，以确保报告的 PVC 大小可用。
- 改进了ONTAP - SAN和ONTAP - SAN经济型节点阶段和取消暂存错误处理、以便在出现故障阶段后取消暂存以删除设备。
- 添加了一个自定义角色生成器、允许客户在ONTAP中为Trident创建一个精简角色。
- 添加了用于故障排除的其他日志记录 `lsscsi`"问题描述#792"()。

Kubernetes

- 为KubeNet本机工作流添加了新的Trident功能：
  - 数据保护
  - 数据迁移
  - 灾难恢复
  - 应用程序移动性

    "了解更多关于Trident Protect的信息"。
- 添加了新标志 `--k8s-api-qps`安装程序设置Trident与 Kubernetes API 服务器通信所使用的 QPS 值。
- 为安装程序添加了 `--node-prep`一个标志、用于自动管理Kubernetes集群节点上的存储协议依赖关系。已测试并验证与Amazon Linux 2023 iSCSI存储协议的兼容性
- 增加了对在非正常节点关闭情况下强制断开ONTAP - NAS经济型卷的支持。
- 使用后端选项时、新的NFS-NAS经济型ONTAP卷将使用每个qtree导出策略 `autoExportPolicy`。只有在发布时、qtrees才会映射到节点限制性导出策略、以提高访问控制和安全性。当Trident从所有节点取消发布卷时、现有qtrees将切换到新的导出策略模型、这样做不会影响活动工作负载。
- 增加了对Kubnetes 1.31的支持。

试验性增强功能

- 增加了对ONTAP驱动程序的光纤通道支持的技术预览。

修复

- **Kubernetes**：
  - 阻止Trident Helm安装的固定兰彻入场网钩"问题描述#839"()。
  - Helm图表值中的固定相关性键"问题描述#898"()。
  - 固定tentControllerPluginNodeSelector/tIdentNodePluginNodeSelector不 能与"true"值一起使用"问题描述#899"()。
  - 已删除克隆期间创建的一段时间快照"问题描述#901"()。
- 增加了对Windows Server 2019的支持。
- 在Trident repo()中修复`Go mod Tidy `"问题描述#767"。

已弃用

- * Kubernetes：*
  - 已将支持的最小Kubbernetes更新为1.25。
  - 不再支持POD安全策略。

产品品牌重塑

从24.10版开始、Astra Trident更名为Trident (NetApp Trident)。此品牌重塑不会影响Trident的任何功能、支持的平台或互操作性。

24.06中的变化

增强功能

- 重要： `limitVolumeSize` 现在，参数限制ONTAP经济型驱动程序中的qtree/LUN大小。使用新 `limitVolumePoolSize` 参数控制这些驱动程序中的FlexVol大小。"问题描述#341"()。
- 增加了iSCSI自我修复功能，以便在使用弃用的igroup时按确切的LUN ID启动SCSI扫描"问题描述#883"()。
- 增加了对卷克隆操作和调整大小操作的支持、即使后端处于暂停模式也是如此。
- 增加了将Trident控制器的用户配置日志设置传播到Trident节点Pod的功能。
- 在Trident中增加了对ONTAP的支持、以便默认情况下使用REST、而不是使用ONTAPI (ZAPI)来运行9.15.1及更高版本。
- 增加了对新永久性卷在ONTAP存储后端的自定义卷名称和元数据的支持。
- 增强了 `azure-netapp-files` (ANF)驱动程序功能、可在NFS挂载选项设置为使用NFS 4.x时默认自动启用Snapshot目录
- 增加了对NFS卷的Bottleroclet支持。
- 增加了对Google Cloud NetApp卷的技术预览支持。

Kubernetes

- 增加了对Kubnetes 1.30的支持。
- 新增了Trident DemonSet在启动时清理zombie挂载和剩余跟踪文件的功能"问题描述#883"()。
- 添加了用于动态导入LVM卷的PVC标注 `trident.netapp.io/luksEncryption`"问题描述#849"()。
- 为ANF驱动程序添加了拓扑感知功能。
- 增加了对Windows Server 2022节点的支持。

修复

- 修复了因事务陈旧而导致的Trident安装失败问题。
- 修复了忽略来自Kubbernetes ()的警告消息的trdentcdc"问题描述#892"。
- 已将Trident控制器优先级更 `SecurityContextConstraint`改为 `0`"问题描述#887"()。
- ONTAP驱动程序现在接受低于 20 MiB 的卷大小（ "问题[#883." ）。

- 修复了Trident、以防止在对ONTAP驱动程序执行调整大小操作期间缩减FlexVol卷。
- 修复了NFS v4.1中ANF卷导入失败的问题。

24.02中的变化

增强功能

- 增加了对云身份的支持。
  - 带有ANF的AK—Azure工作负载标识将用作云标识。
  - 具有FSxN - AWS IAM角色的EKS将用作云身份。
- 增加了从EKS控制台将Trident作为附加项安装在EKS集群上的支持。
- 增加了配置和禁用iSCSI自我修复("问题描述#864"）。
- 在ONTAP驱动程序中添加了Amazon FSx特性，以实现与AWS IAM和SecretsManager的集成，并使Trident能够删除带有备份的FSx卷"问题描述#453."()。

Kubernetes

- 增加了对Kubnetes 1.29的支持。

修复

- 修复了未启用ACP时出现的ACP警告消息("问题描述#866"）。
- 增加了在删除ONTAP驱动程序的快照期间、如果克隆与快照关联、则在执行克隆拆分之前的10秒延迟。

已弃用

- 从多平台映像清单中删除了内置证明框架。

23.10中的变化

修复

- 修复了在新请求的大小小于ONTAP－NAS和ONTAP－NAS－Flexgroup存储驱动程序的卷总大小时进行卷扩展的问题("问题描述#834"）。
- 固定卷大小、以便在导入ONP-NAS和ONP-NAS Flexgroup存储驱动程序期间仅显示卷的可用大小("问题描述#的 第请选择"）。
- 针对ONTAP -NAS经济的固定FlexVol名称转换。
- 修复了重新启动Windows节点时该节点上的Trident初始化问题。

增强功能

Kubernetes

增加了对Kubnetes 1.28的支持。

Trident

- 增加了对Azure托管身份(AMI)与azure-NetApp-files存储驱动程序的使用支持。
- 增加了对ONTAP SAN驱动程序基于TCP的NVMe的支持。
- 新增了在用户将后端设置为暂停状态时暂停卷配置的功能("问题描述#558."）。

23.07.1中的变更

*Kubernetes:*修复了删除守护程序集的问题，以支持零停机升级("问题描述740"）。

23.07中的变更

修复

Kubernetes

- 修复了通过升级三项功能忽略处于终止状态的旧Pod的问题("问题描述740"）。
- 为"瞬 时-三端版本-POD "定义添加了容差("问题描述#C了"）。

Trident

- 修复了ONTAPI (ZAPI)请求、以确保在节点暂存操作期间获取LUN属性以识别和修复虚影iSCSI设备时查询LUN序列号。
- 修复了存储驱动程序代码("问题描述#816"）。
- 固定了使用ONTAP驱动程序和use-rest=true时的配额大小调整。
- 修复了在ONTAP SAN经济模式下创建LUN克隆的问题。
- 从还原发布信息字段 `rawDevicePath` to `devicePath`；添加了用于填充和恢复的逻辑(在某些情况下) `devicePath` 字段。

增强功能

Kubernetes

- 增加了对导入预配置快照的支持。
- 最小化部署和守护进程Linux权限("问题描述#十十一号"）。

Trident

- 不再报告"联机"卷和快照的状态字段。
- 如果ONTAP后端处于脱机状态("问题801"， "#543."）。
- LUN序列号始终在ControllerVolumePubl出版 工作流期间进行检索和发布。
- 添加了其他逻辑来验证iSCSI多路径设备序列号和大小。
- 对iSCSI卷进行额外验证、以确保取消暂存正确的多路径设备。

试验性增强

为ONTAP SAN驱动程序添加了基于TCP的NVMe技术预览支持。

文档。

在组织和格式方面进行了许多改进。

已弃用

Kubernetes

- 不再支持v1beta1快照。
- 不再支持CSI之前的卷和存储类。
- 已将支持的最小Kubbernetes更新为1.22。

23.04年度的变动

> 只有启用了非正常节点关闭功能门的Kubernetes版本才支持对ONP-SANON-*卷强制执行卷断开。必须在安装时使用启用强制断开 `--enable-force-detach` 三元安装程序标志。

修复

- 修复了在规范中指定的情况下使用IPv6 localhost进行安装的Trident操作员。
- 修复了要与捆绑包权限("问题描述 #799"）。
- 采用rwx模式在多个节点上附加原始块卷的固定问题描述。
- 修复了SMB卷的FlexGroup 克隆支持和卷导入。
- 修复了在问题描述 中、三端存储控制器无法立即关闭的问题("问题描述 #811"）。
- 添加了一个修复程序，用于列出与使用ONTAP SAN-*驱动程序配置的指定LUN关联的所有igrop名称。
- 添加了一个修复程序、允许外部进程运行到完成状态。
- 修复了s390架构的编译错误("问题描述 #537"）。
- 修复了卷挂载操作期间日志记录级别不正确的问题("问题描述 #781."）。
- 修复了潜在类型断言错误("问题描述 #802."）。

增强功能

- Kubernetes：
  - 增加了对Kubnetes 1.27的支持。
  - 增加了对导入LUKS卷的支持。
  - 增加了对ReadWriteOncePod PVC访问模式的支持。
  - 增加了对在非正常节点关闭情况下对ONTAP－SAN－*卷强制断开的支持。
  - 现在、所有ONTAP SAN-*卷都将使用每个节点的igroup。只有在将LUN主动发布到这些节点时、这些LUN才会映射到igroup、以改善我们的安全防护。当Trigent确定可以安全地切换到新的igrop方案而不

会影响活动工作负载时、现有卷将适时切换到新的igrop方案("问题描述 #758"）。

- 通过从ONTAP SAN-*后端清除未使用的通过三叉点管理的igroup、提高了三叉点的安全性。

- 通过Amazon FSx向ONGP-NAS经济型和ONGP-NAS Flexgroup存储驱动程序增加了对SMB卷的支持。
- 通过ONTAP -NAS、ONTAP -NAS经济模式和ONTAP -NAS Flexgroup存储驱动程序增加了对SMB共享的支持。
- 增加了对ARM64节点的支持("问题描述 #732"）。
- 通过先停用API服务器改进了通过使用操作步骤 的三项功能("问题描述 #811"）。
- 为Makefile添加了对Windows和ARM64主机的跨平台构建支持；请参见Build .md。

已弃用

- Kubenetes:**配置ONTAP－SAN和ONTAP－SAN－Economy驱动程序时，将不再创建后端范围的igrou("问题描述 #758"）。

23.01.1中的变更

修复

- 修复了在规范中指定的情况下使用IPv6 localhost进行安装的Trident操作员。
- 修复了Trident操作员集群角色权限与捆绑包权限同步的问题 "问题描述 #799"。
- 添加了一个修复程序、允许外部进程运行到完成状态。
- 采用rwx模式在多个节点上附加原始块卷的固定问题描述。
- 修复了SMB卷的FlexGroup 克隆支持和卷导入。

2001年23月的变更

现在、在Trident中支持Kubnetes 1.27。在升级Kubernetes之前升级Trident。

修复

- Kubernetes：添加了一些选项、用于排除通过Helm ("问题783、794"）。

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("问题描述 #757"）。
- 增加了自动化功能、可检测和修复主机节点上中断或陈旧的iSCSI会话。
- 增加了对扩展LUKS加密卷的支持。
- Kubernetes：增加了对LUKS加密卷的凭据轮换支持。

Trident

- 在SMS-NAS存储驱动程序中增加了对ONTAP卷的Amazon FSx for NetApp ONTAP支持。

- 增加了对使用SMB卷时的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。
- 增加了在使用ontap-nas-flexgroup存储驱动程序创建FlexGroup时可选使用FlexgroupAggregateList的支持。
- 提高了管理多个FlexVol卷时ONTAP -NAS经济型存储驱动程序的性能
- 已为所有ONTAP NAS存储驱动程序启用数据LIF更新。
- 更新了Trident部署和DemonSet命名约定、以反映主机节点操作系统。

已弃用

- Kubernetes：已将支持的最低Kubernetes更新为1.21。
- 配置或 `ontap-san-economy`驱动程序时、不应再指定`DataLIF `ontap-san`。

22.10中的更改

*在升级到Trident 22.10.*之前，必须阅读以下重要信息



有关Trident 22.10 的信息

- 现在、在Trident中支持Kubnetes 1.25。您必须先将Trident升级到22.10、然后才能升级到Kubbernetes 1.25。
- 现在、Trident会在SAN环境中严格强制使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

修复

- 已修复使用创建的ONTAP 后端专用的问题描述 `credentials` 字段在22.07.0升级期间无法联机("问题描述 #759"）。
- "** Docker："*修复了导致Docker卷插件在某些环境中无法启动的问题描述 ("问题描述 #548" 和 "问题描述 760"）。
- 修复了ONTAP SAN后端特有的SLM问题、以确保仅发布属于报告节点的部分数据LIF。
- 修复了连接卷时发生不必要的iSCSI LUN扫描的性能问题描述。
- 删除了Trident iSCSI工作流中的细粒度重试、以快速失败并缩短外部重试间隔。
- 修复了问题描述 、在刷新iSCSI设备时、如果已刷新相应的多路径设备、则会返回错误。

增强功能

- Kubernetes：
  - 增加了对Kubnetes 1.25的支持。您必须先将Trident升级到22.10、然后才能升级到Kubbernetes 1.25。
  - 为Trident部署和DemonSet添加了单独的ServiceAccount、ClusterRole和ClusterRoleBinding-以增强未来的权限。

- 增加了对的支持 "跨命名空间卷共享"。

- 所有Trident `ontap-*` 现在、存储驱动程序可与ONTAP REST API配合使用。
- 添加了新的运算符YAML (`bundle_post_1_25.yaml`)、而不使用 `PodSecurityPolicy` 以支持Kubernetes 1.25。
- 已添加 "支持LUKS加密卷" 适用于 `ontap-san` 和 `ontap-san-economy` 存储驱动程序。
- 增加了对Windows Server 2019节点的支持。
- 已添加 "支持Windows节点上的SMB卷" 通过 `azure-netapp-files` 存储驱动程序。
- ONTAP 驱动程序的自动MetroCluster 切换检测现已全面推出。

已弃用

- ** Kubernetes：*已将支持的最低Kubernetes更新为1.20。
- 已删除Astra数据存储(ADS)驱动程序。
- 删除了对的支持 `yes` 和 `smart` 选项 `find_multipaths` 为iSCSI配置工作节点多路径时。

22.07中的更改

修复

*

- 修复了使用Helm或Trident运算符配置Trident时用于处理节点选择器的布尔值和数字值的问题描述。（"GitHub问题描述 700"）
- 修复了问题描述 处理非CHAP路径错误的问题、以便kubelet在失败时重试。 "GitHub问题描述 #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
- 现在、ONTAP SAN卷将使用每个节点的igroup、并且仅将LUN映射到igroup、而将其主动发布到这些节点、以改善我们的安全状况。如果Trident确定在不影响活动工作负载的情况下安全执行此操作、现有卷将有机会切换到新的igroup方案。
- 包含一个包含Trident安装的ResourceQuota、以确保在默认情况下限制使用PriorityClass时计划Trident DemonSet。
- 在Azure NetApp Files驱动程序中增加了对网络功能的支持。（"GitHub问题描述 #717"）
- 为ONTAP 驱动程序添加了技术预览自动MetroCluster 切换检测功能。（"GitHub问题描述 #228"）

已弃用

- 。。。。
- 后端配置不再允许在一个配置中使用多种身份验证类型。

删除

- 已删除AWS CVS驱动程序(自22.04起已弃用)。

- Kubernetes
  - 从节点Pod中删除了不必要的SYS_ADMIN功能。
  - 将nodeprep减少为简单的主机信息和主动服务发现、以便尽力确认工作节点上是否提供NFS/iSCSI服务。

文档。

添加了一个新的"POD安全标准"(PSS)部分，详细介绍了Trident在安装时启用的权限。

22.04中的更改

NetApp 不断改进和完善其产品和服务。下面是Trident中的一些最新功能。有关以前版本的信息，请参阅 "文档的早期版本"。

> 如果要从先前的任何 Trident 版本升级并使用 Azure NetApp Files ，则 `location` config 参数现在是一个必填字段。

修复

- 改进了 iSCSI 启动程序名称的解析。（"GitHub问题描述 #681"）
- 修复了不允许使用 CSI 存储类参数的问题描述 。（"GitHub问题描述 598"）
- 修复了 Trident CRD 中的重复密钥声明。（"GitHub问题描述 #6771"）
- 修复了不准确的 CSI Snapshot 日志。（"GitHub问题描述 #629"）
- 修复了已删除节点上的卷已取消发布的问题描述 。（"GitHub 问题描述 第 691 号"）
- 增加了对块设备上文件系统不一致问题的处理。（"GitHub问题描述 #656"）
- 修复了在安装期间设置 `imageRegistry` 标志时问题描述 提取自动支持映像的问题。（"GitHub问题描述 #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描述问题。

增强功能

- 现在，与 Trident 安全端点的入站连接至少需要 TLS 1.3 。（"GitHub问题描述 #698"）
- 现在， Trident 会将 HSTS 标头添加到其安全端点的响应中。
- Trident 现在会尝试自动启用 Azure NetApp Files UNIX 权限功能。
- * Kubernetes * ： Trident demonset 现在以 system-node-critical 优先级类运行。（"GitHub问题描述 #694"）

删除

已删除 E 系列驱动程序（自 2007 年 20 月 20 日起禁用）。

22.01.1 中的更改

修复

- 修复了已删除节点上的卷已取消发布的问题描述 。（"GitHub 问题描述 第 691 号"）
- 修复了访问 ONTAP API 响应中聚合空间的 " 无 " 字段时的崩溃问题。

22.01.0 中的更改

修复

- * Kubernetes： * 增加大型集群的节点注册回退重试时间。
- 修复了问题描述 ，其中 azure-netapp-files 驱动程序可能会被同名的多个资源混淆。
- 如果使用方括号指定ONTAP SAN IPv6数据LIF、则这些LIF现在可以正常工作。
- 修复的问题描述 ，尝试导入已导入的卷时，返回的 EOF 将使 PVC 处于待定状态。（"GitHub 问题描述 489"）
- 修复了在SolidFire卷上创建32个快照时Trident性能降低的问题。
- 在创建 SSL 证书时将 SHA-1 替换为 SHA-256 。
- 修复了Azure NetApp Files驱动程序、允许重复的资源名称并将操作限制在一个位置。
- 修复了Azure NetApp Files驱动程序、允许重复的资源名称并将操作限制在一个位置。

增强功能

- Kubernetes 增强功能：
  - 增加了对 Kubernetes 1.23 的支持。
  - 通过 Trident 操作员或 Helm 安装 Trident Pod 时，为其添加计划选项。（"GitHub 问题描述 #651"）
- 在 GCP 驱动程序中允许跨区域卷。（"GitHub 问题描述 #633"）
- 增加了对Azure NetApp Files卷"unixPermissions (unixPermissions)"选项的支持。（"GitHub 问题描述 #666"）

已弃用

Trident REST 接口只能在 127.0.0.1 或 [：：： 1 ）地址处侦听和提供服务

21.10.1 中的更改

> v21.10.0 版本具有一个问题描述，在删除节点并将其重新添加回 Kubernetes 集群时， Trident 控制器可以将其置于 CrashLoopBackOff 状态。此问题描述在 v21.10.1 中得到了修复（ GitHub 问题描述 669 ）。

修复

- 修复了在 GCP CVS 后端导入卷导致导入失败的潜在争用情况。
- 修复了一个问题描述，在删除节点并将其重新添加回 Kubernetes 集群时，可能会将 Trident 控制器置于 CrashLoopBackOff 状态（ GitHub 问题描述 669 ）。

- 修复了在未指定 SVM 名称的情况下不再发现 SVM 的问题描述（ GitHub 问题描述 612 ）。

21.10.0 中的更改

修复

- 修复了问题描述，其中无法将 XFS 卷的克隆挂载到与源卷相同的节点上（ GitHub 问题描述 514 ）。
- 修复了Trident在关闭时记录致命错误的问题(GitHub问题597)。
- 与 Kubernetes 相关的修复程序：
  - 使用 `ontap-NAS` 和 `ontap-nas-flexgroup` 驱动程序创建快照时，将卷的已用空间返回为最小还原大小（ GitHub 问题描述 645 ）。
  - 修复了问题描述，其中 **无法扩展文件系统** 在调整卷大小后记录了错误（ GitHub 问题描述 560 ）。
  - 修复了 POD 可能停留在 **正在终止** 状态的问题描述（ GitHub 问题描述 572 ）。
  - 修复了 `ontap-san-economy-lun` FlexVol 可能已满快照 LUN 的情况（ GitHub 问题描述 533 ）。
  - 使用不同映像修复了自定义 YAML 安装程序问题描述（ GitHub 问题描述 613" ）。
  - 固定快照大小计算（ GitHub 问题描述 611 ）。
  - 修复了所有Trident安装程序都可以将纯KubeNet标识为OpenShift的问题(GitHub第639期)。
  - 修复了 Trident 操作员在无法访问 Kubernetes API 服务器时停止协调的问题（ GitHub 问题描述 599 ）。

增强功能

- 为 GCP-CVS 性能卷增加了对 `unixPermissions` 选项的支持。
- 增加了对 GCP 中 600 GiB 到 1 TiB 范围内的扩展优化 CVS 卷的支持。
- Kubernetes 相关增强功能：
  - 增加了对 Kubernetes 1.22 的支持。
  - 已启用 Trident 操作员和 Helm 图表以使用 Kubernetes 1.22 （ GitHub 问题描述 628 ）。
  - 已将操作员映像添加到 `tridentctl` images 命令（ GitHub 问题描述 570 ）中。

实验增强功能

- 在 `ontap-san` 驱动程序中增加了对卷复制的支持。
- 增加了对 `ontap-nas-flexgroup` ， `ontap-san` 和 `ontap-nas-economy` 驱动程序的 * 技术预览 * REST 支持。

已知问题

已知问题用于确定可能会阻止您成功使用本产品的问题。

- 在将安装了Trident的Kubernetes集群从1.24升级到1.25或更高版本时、您必须 `helm upgrade`先更新`values.yaml`以设置为或添加到 `true`命令、 `--set excludePodSecurityPolicy=true`然后才能升级集群。 `excludePodSecurityPolicy`

- Trident现在会 (`fsType=""`)对存储类中未指定的卷强制使用空白 `fsType`) `fsType`。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、在使用安全上下文强制实施时、您需要在StorageClass上 `fsGroup`设置 `fsType`。
- 在多个Trident实例中使用后端时、每个后端配置文件应为ONTAP后端设置不同的值、或者为SolidFire后端 `storagePrefix`设置不同的值 `TenantName`。Trident无法检测其他Trident实例已创建的卷。尝试在ONTAP或SolidFire后端创建现有卷会成功、因为Trident会将卷创建视为一项具有等量功能的操作。如果`storagePrefix`或 `TenantName`不不同、则在同一后端创建的卷可能会发生名称冲突。
- 在安装Trident (使用或Trident操作员)以及使用 `tridentctl`管理Trident时 `tridentctl`、您应确保 `KUBECONFIG`已设置环境变量。这是指示应处理的Kubbernetes集群所必需的 `tridentctl`。在使用多个Kubnetes环境时、您应确保`KUBECONFIG`文件的来源准确无误。
- 要对 iSCSI PV 执行联机空间回收，工作节点上的底层操作系统可能需要将挂载选项传递到卷。这一点适用于RERL/Red Hat Enterprise Linux CoreTM OS (RHCOS)实例、此实例需要 `discard` "挂载选项"；请确保在n ü^中包含discard mountOption、以支持联机块丢弃[`StorageClass`。
- 如果每个Kubornetes集群具有多个Trident实例、则Trident无法与其他实例进行通信、并且无法发现其创建的其他卷、如果在一个集群中运行多个实例、则会导致出现意外且不正确的行为。每个Kubnetes集群只应有一个Trident实例。
- 如果在Trident脱机时从Kubnetes中删除基于Trident的 `StorageClass`对象、则`Trident`不会在其数据库恢复联机后从其数据库中删除相应的存储类。您应使用或`REST API`删除这些存储类 `tridentctl`。
- 如果用户在删除相应的PVC之前删除了Trident配置的PV、则Trident不会自动删除后备卷。您应通过或REST API删除此卷 `tridentctl`。
- ONTAP 不能同时配置多个 FlexGroup ，除非聚合集对于每个配置请求是唯一的。
- 使用基于IPv6的Trident时、应在后端定义中使用方括号指定 `managementLIF`和 `dataLIF`。例如 `[fd20:8b1e:b258:2000:f816:3eff:feec:0]`。

> 您不能在ONTAP SAN后端指定 `dataLIF`。Trident会发现所有可用的iSCSI LUN并使用它们建立多路径会话。

- 如果使用 `solidfire-san` 驱动程序对于OpenShift 4.5、请确保底层工作节点使用MD5作为CHAP身份验证算法。Element 12.7提供了符合FIPS的安全CHAP算法SHA1、SHA-256和SHA3-256。

了解更多信息

- "Trident GitHub"
- "Trident博客"

文档的早期版本

如果您未运行 Trident 26.10，则可以根据 "Trident支持生命周期" 获得先前版本的文档。

- "Trident 26.06"
- "Trident 26.02"
- "Trident 25.10"
- "Trident25.06"
- "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。
- Trident 25.10 及更高版本支持 FC 以及 iSCSI 和 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

- Linux 统一密钥设置（LUKS）加密
- SnapMirror 卷复制
- 限制聚合使用
- 空间预留模式

- 快照
- 分层

有关详细信息，请参见 "ONTAP SAN配置选项和示例" 。

已知问题

已知问题可确定可能妨碍您成功使用此版本产品的问题。

以下已知问题会影响当前版本：

从 **Trident 22.10** 或更早版本升级后，卷仍保持连接状态

此问题影响在附加卷时直接从 Trident 22.10 或更早版本升级到 Trident 26.10 的客户。从 Trident 23.01 或更高版本升级不会受到影响。

升级后，受影响的卷可能无法分离。使用 ReadWriteOnce（RWO）卷的 Pod 无法移动到其他节点，并且可能会报告多重连接错误。删除 PersistentVolume（PV）也可能挂起。

要解决此问题，请在存在陈旧卷挂载的节点上，从该卷的 `globalmount`目录中删除残留的 `volumePublishInfo.json`文件。

VolumeSnapshots 未到达 **ReadyToUse** 状态

> `csi-snapshotter` sidecar 存在已知问题。在所有 Kubernetes 版本中，VolumeGroupSnapshots `v1beta1` 阻止 VolumeSnapshots 到达 `ReadyToUse`状态。

有两种解决方法可供选择：

1. 删除 VolumeGroupSnapshots CRD 以禁用 VolumeGroupSnapshots，然后重新安装 Trident。
2. 安装 VolumeGroupSnapshots `v1beta2` 和 `snapshot-controller` 版本 8.4.0 或更高版本，然后重新安装 Trident。VolumeGroupSnapshots 在 v1.34 之前的 Kubernetes 版本上不起作用。

还原大型文件的**Restic**备份可能会失败

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

入门

了解Trident

了解Trident

Trident 是由 NetApp 维护的一个受全面支持的开源项目。它旨在帮助您使用容器存储接口(CSI)等行业标准接口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、Amazon FSx for NetApp ONTAP 和 Google Cloud NetApp Volumes。

Trident是一款与本机集成的符合容器存储接口(CI)的动态存储流程编排程序"Kubernetes"。Trident在集群中的每个工作节点上作为一个控制器Pod加一个节点Pod运行。有关详细信息、请参见 "Trident架构" 。

Trident还可以直接与适用于NetApp存储平台的Docker生态系统集成。NetApp Docker卷插件(nDVP)支持从存储平台到Docker主机配置和管理存储资源。有关详细信息、请参见 "部署适用于Docker的Trident" 。

> 如果这是您首次使用Kubarnetes、您应熟悉 "Kubbernetes概念和工具"。

支持的 Kubernetes 平台

Trident 支持一系列 Kubernetes 发行版和平台。

支持的平台包括： * 上游 Kubernetes * Red Hat OpenShift * SUSE 虚拟化（Harvester） 1.8.x - 1.9.x （ONTAP NFS、iSCSI、NVMe/TCP 和 FC）

Kubnetes与NetApp产品的集成

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

▼ 适用于 NetApp ONTAP 的 Amazon FSX

"适用于 NetApp ONTAP 的 Amazon FSX" 是一项完全托管的AWS服务、可用于启动和运行由NetApp ONTAP存储操作系统提供支持的文件系统。

▼ Azure NetApp Files

"Azure NetApp Files" 是一种企业级 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，同时享受 NetApp 应有的性能和丰富的数据管理功能。

▼ Cloud Volumes ONTAP

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

▼ Google Cloud NetApp卷

"Google Cloud NetApp卷" 是Google Cloud中的一项完全托管的文件存储服务、可提供高性能企业级文件存

储。

▼ **Element** 软件

"Element" 通过保证性能并简化存储占用空间，使存储管理员能够整合工作负载。

▼ **NetApp HCI**

"NetApp HCI" 通过自动化执行日常任务并使基础架构管理员能够专注于更重要的功能，简化数据中心的管理和扩展。

Trident 可以直接在底层 NetApp HCI 存储平台上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、混合或纯HDD配置、并提供许多不同的部署模式：内部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在集群中的每个工作节点上作为一个控制器Pod加一个节点Pod运行。节点Pod必须运行在可能要挂载Trident卷的任何主机上。

了解控制器Pod和节点Pod

Trident在Kubernetes集群上部署为一个三项控制器Pod或多个[三级节点块]、并使用标准Kubernetes _CSI Sidecar Containers_来简化CSI插件的部署。"Kubernetes CSI Sidecar Containers"由Kubbernetes存储社区维护。

Kubornetes"节点选择器"和"容忍和损害"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。

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

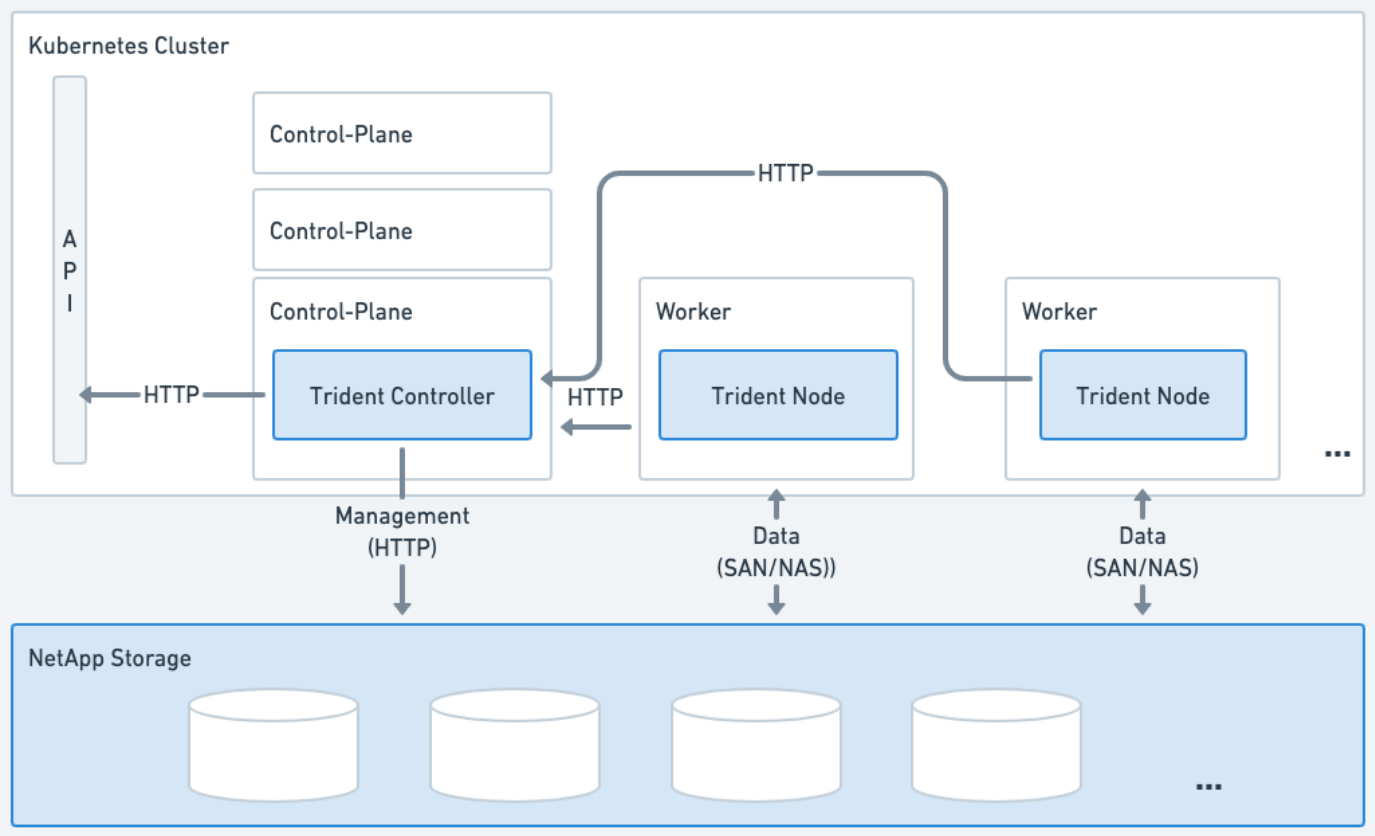


图 **1. Trident**部署在**Kubbernetes**集群上

三项控制器**Pod**

三端控制器Pod是一个运行CSI控制器插件的Pod。

- 负责配置和管理NetApp存储中的卷
- 由Kubbernetes部署管理
- 可以在控制面板或工作节点上运行、具体取决于安装参数。

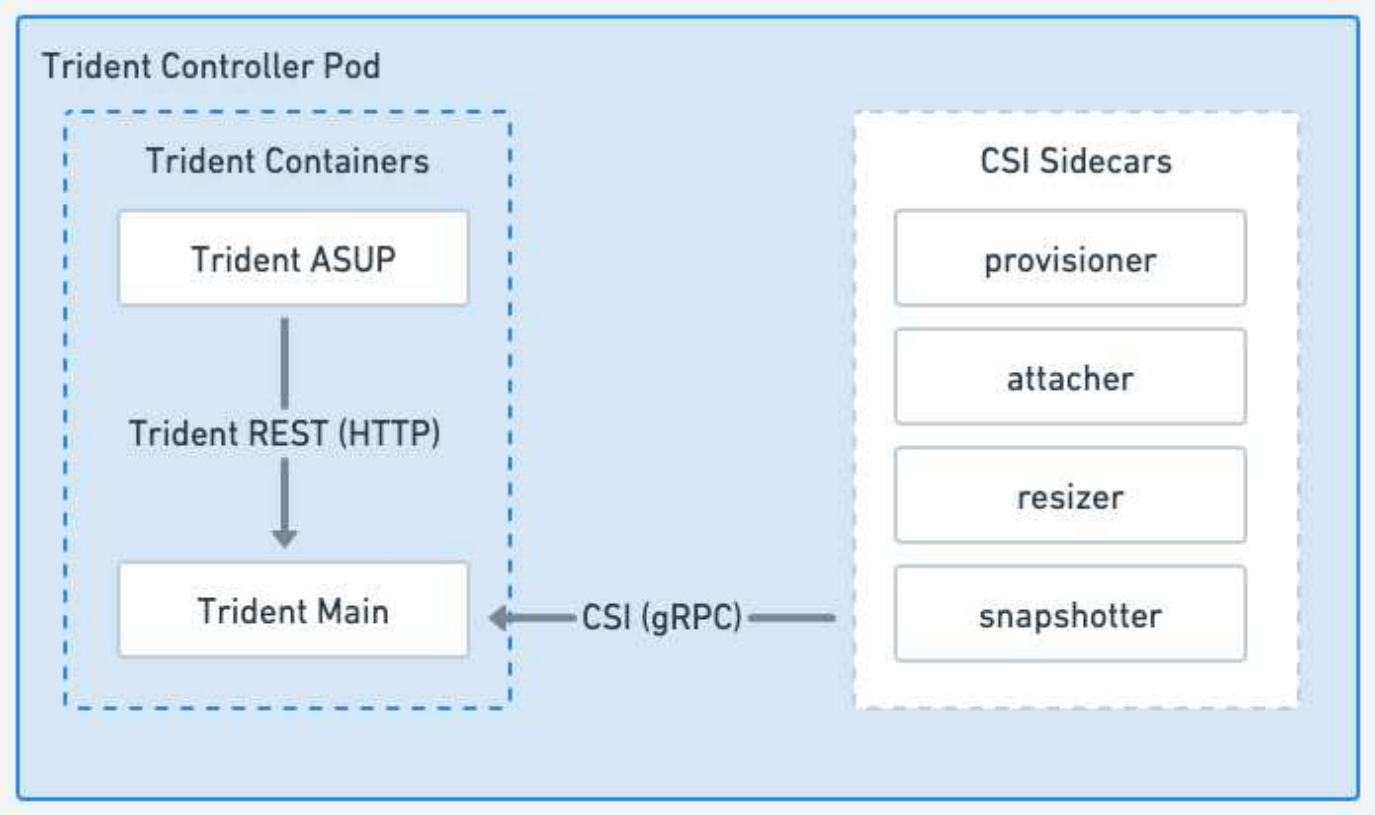


图 **2. TRIdent**控制器**Pod**示意图

三级节点块

三端节点块是运行CSI节点插件的有权限的节点。

- 负责挂载和卸载主机上运行的Pos的存储
- 由Kubbernetes DemonSet管理
- 必须在要挂载NetApp存储的任何节点上运行

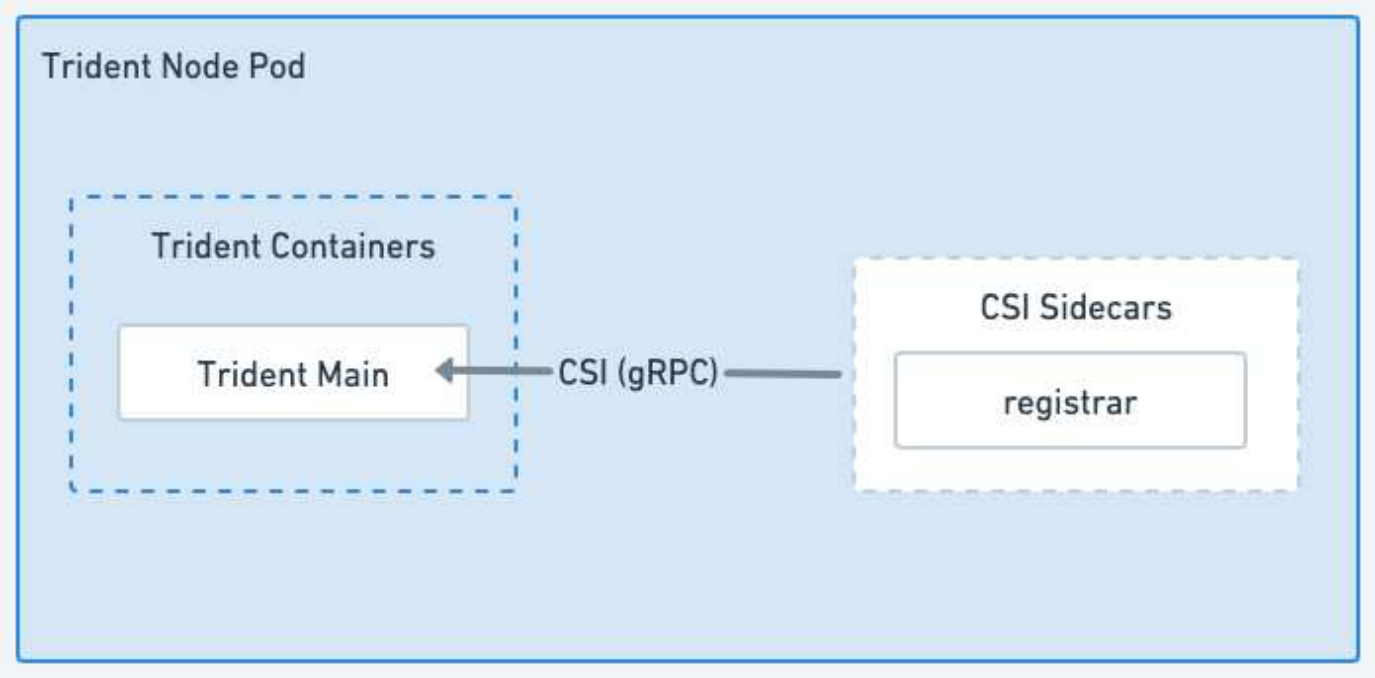


图 **3. TRIdent**节点**Pod**图

支持的 **Kubernetes** 集群架构

以下Kubenetes架构支持Trident：

Kubernetes 集群架构	**supported**	默认安装
单个主节点，计算节点	是的。	是的。
多主机，计算	是的。	是的。
主， `etcd` ，计算	是的。	是的。
主机，基础架构，计算	是的。	是的。

概念

Trident 卷配置流程

Trident中的配置分为两个主要阶段。第一阶段会将存储类与一组合适的后端存储池相关联，并在配置之前进行必要的准备。第二阶段包括卷创建本身、需要从与待定卷的存储类关联的存储池中选择一个存储池。

存储类关联

将后端存储池与存储类相关联取决于存储类请求的属性及其 `storagePools`、 `additionalStoragePools`和`excludeStoragePools`列表。创建存储类时， Trident 会将其每个后端提供的属性和池与存储类请求的属性和池进行比较。如果某个存储池的属性和名称与请求的所有属性和池名称匹配、则Trident会将该存储池添加到该存储类的一组合适存储池中。此外、Trident还会将列表中列出的所有存储池添加到该池 `additionalStoragePools`集中、即使其属性不满足存储类请求的全部或任何属性也是如此。您应使用此 `excludeStoragePools`列表覆盖存储池并将其从存储类中删除。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、并删除标记为已排除的任何。

卷创建

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。创建卷时、Trident会首先获取该卷的存储类的存储池集、如果为该卷指定协议、则Trident会删除无法提供所请求协议的存储池(例如、NetApp HCI或SolidFire后端无法提供基于文件的卷、而ONTAP后端无法提供基于块的卷)。Trident会随机排列此结果集的顺序、以便于均匀分布卷、然后在其中进行迭代、尝试依次在每个存储池上配置卷。如果在一个上成功，则它会成功返回，并记录在此过程中遇到的任何故障。只有当*无法在*所有*上配置可供请求的存储类和协议使用的存储池时、Trident才会返回故障。

Trident 卷快照创建

详细了解Trident如何为其驱动程序创建卷快照。

了解如何创建卷快照

- 对于 `ontap-nas`， `ontap-san` ， 和 `azure-netapp-files`驱动程序中，每个持久卷 (PV) 都映射到一个FlexVol volume。因此，卷快照被创建为NetApp快照。NetApp快照技术比同类快照技术具有更高的稳定性、可扩展性、可恢复性和性能。这些快照副本在创建所需时间和存储空间方面都非常高效。
- 。 `ontap-nas-flexgroup` 驱动程序、每个永久性卷(PV)都会映射到一个FlexGroup。因此，卷快照会创建为 NetApp FlexGroup 快照。与竞争对手的快照技术相比、NetApp快照技术可提供更高的稳定性、可扩展性、可恢复性和性能。无论是在创建 Snapshot 副本所需的时间还是在存储空间中，这些 Snapshot 副本都极为高效。
- 对于 `ontap-san-economy`驱动程序、PVs映射到在共享FlexVol卷上创建的LUN PVs的Snapshot可通过对关联LUN执行FlexClone来实现。借助ONTAP FlexClone技术、即使是最大的数据集、也可以近乎瞬时地创建副本。副本与其父级共享数据块，除了元数据所需的存储之外，不会占用任何存储。
- 对于 `solidfire-san` 驱动程序，每个 PV 都会映射到在 NetApp Element 软件 /NetApp HCI 集群上创建的 LUN 。VolumeSnapshot 由底层 LUN 的 Element Snapshot 表示。这些快照是时间点副本，只占用少量系统资源和空间。
- 使用和 `ontap-san`驱动程序时 `ontap-nas`、ONTAP快照是FlexVol的时间点副本、会占用FlexVol本身的空间。这样，在创建 / 计划快照时，卷中的可写空间量会随着时间的推移而减少。解决此问题的一个简单方法是，通过 Kubernetes 调整大小来增大卷的大小。另一个选项是删除不再需要的快照。删除通过KubeNet创建的卷快照后、Trident将删除关联的ONTAP快照。也可以删除未通过 Kubernetes 创建的 ONTAP 快照。

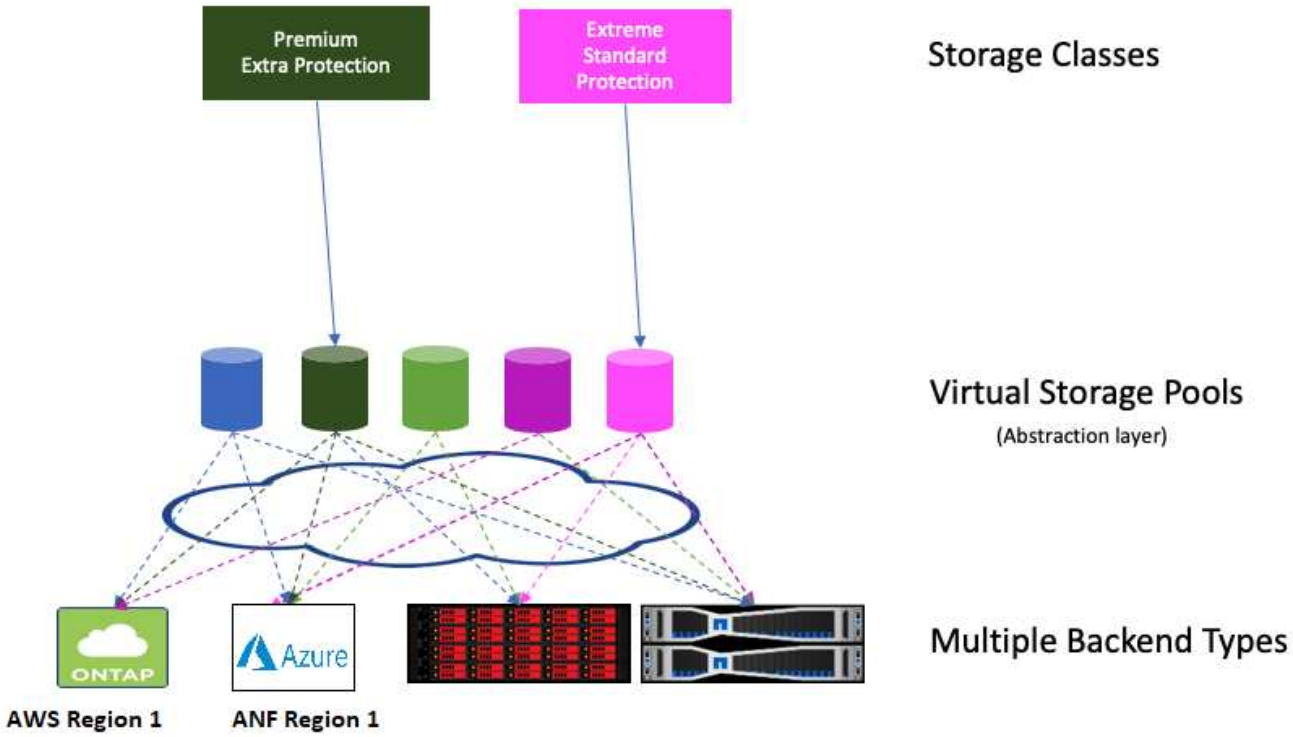
使用Trident，您可以利用 VolumeSnapshots 从中创建新的 PV。使用FlexClone技术从这些快照创建 PV（适用于受支持的ONTAP后端）执行。从快照创建 PV 时，后备卷是快照父卷的FlexClone 。这 `solidfire-san`驱动程序使用 Element 软件卷克隆从快照创建 PV。它在这里从 Element 快照创建一个克隆。

Trident 虚拟存储池配置

虚拟池在Trident存储后端和Kubnetes之间提供了一个抽象层 `StorageClasses`。管理员可以通过它们以一种不受后端限制的通用方式定义各个方面、例如每个后端的位置、性能和保护、而无需 `StorageClass`指定要用于满足所需条件的物理后端、后端池或后端类型。

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后端定义虚拟池。



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

- 定义虚拟池时、请勿尝试在后端定义中重新排列现有虚拟池的顺序。
- 建议不要修改现有虚拟池的属性。您应定义一个新的虚拟池以进行更改。

大多数方面都以后端特定术语来指定。重要的是、宽高值不会公开在后端驱动程序之外、也不能在中进行匹配 `StorageClasses`。而是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，标签可能在唯一的后端通用。与其他方面一样，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，管理员可以根据需要全权定义标签键和值。为了方便起见、存储管理员可以按标签为每个虚拟池和组卷定义标签。

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 `-`

答 `StorageClass` 通过引用选择器参数中的标签来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！ = 至高	不匹配
在 中	位置（东部，西部）	位于一组值中
注释	性能注释（银牌，铜牌）	不在值集内
`<key>`	protection	存在任何值
`!<key>`	！保护	不存在

Trident 卷访问组管理

详细了解Trident如何使用 "卷访问组"。

> 如果使用的是 CHAP ，请忽略此部分，建议使用此部分来简化管理并避免下面所述的扩展限制。此外、如果您在CSI模式下使用Trident、则可以忽略此部分。作为增强型CSI配置程序安装时、Trident使用CHAP。

了解卷访问组

Trident可以使用卷访问组来控制对其配置的卷的访问。如果禁用了CHAP、则它会查找名为的访问组 `trident`、除非您在配置中指定一个或多个访问组ID。

虽然Trident会将新卷与配置的访问组关联起来、但不会自行创建或管理访问组。在将存储后端添加到Trident之前、这些访问组必须存在、并且它们必须包含可能会挂载由该后端配置的卷的Kubbernetes集群中每个节点的iSCSI IQN。在大多数安装中，包括集群中的每个工作节点。

对于节点数超过 64 个的 Kubernetes 集群，您应使用多个访问组。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。在最多配置四个访问组的情况下，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、请参见 "此处"。

如果要从使用默认值的配置修改配置 `trident` 访问组到也使用其他的访问组、并包括的ID `trident` 列表中的访问组。

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。开始之前，请查看"Trident要求"。

> 对于Docker，请参见"适用于Docker的Trident"。

1 准备工作节点

Kubernetes集群中的所有工作节点都必须能够挂载为Pod配置的卷。

"准备工作节点"

2 安装**Trident**

Trident提供了多种针对各种环境和组织进行了优化的安装方法和模式。

"安装 Trident"

3 创建后端

后端用于定义Trident与存储系统之间的关系。它会告诉Trident如何与该存储系统通信、以及Trident如何从该存储系统配置卷。

"配置后端" 存储系统

4 创建**Kubbernetes**存储类

Kubnetes 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会为指定Trident配置程序的Kubbernetes对象创建匹配的存储类。

"创建存储类。"

5 配置卷

永久性卷(PV)是由集群管理员在Kubernetes集群上配置的物理存储资源。*PersentVolumeClaim*(PVC)是对集群上的PersentVolume的访问请求。

创建一个使用已配置的Kubernetes StorageClass来请求对PV的访问的永久性卷(PV)和永久性卷克莱姆(PVC)。然后、您可以将PV挂载到POD。

"配置卷"

下一步是什么?

现在、您可以添加其他后端、管理存储类、管理后端以及执行卷操作。

Trident要求

在安装Trident之前、您应查看这些常规系统要求。特定后端可能有其他要求。

有关**Trident**的关键信息

您必须阅读以下有关**Trident**的重要信息。

▼ ****与**Trident **相关的信息

- Trident 26.10 支持 Kubernetes 1.27 到 1.37。在升级 Kubernetes 之前，请检查下面的特定版本映像列表。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值

`find_multipaths: no`。

使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

支持的前端（编排程序）

Trident支持多个容器引擎和流程编排程序、其中包括：

- Anthos On－Prem (VMware)和Anthos on Bare metal 1.16
- Kubernetes 1.27 - 1.37
- OpenShift 5.0
- OpenShift 4.12、4.14 - 4.22 （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继续支持旧版 OpenShift 版本，与"Red Hat 扩展更新支持 (EUS) 发布生命周期"，即使它们依赖上游不再受官方支持的 Kubernetes 版本。在这种情况下安装Trident时，您可以放心地忽略有关 Kubernetes 版本的任何警告消息。

- SUSE Rancher Kubernetes Engine 2（RKE2）v1.28.x - 1.37.x
- SUSE 虚拟化（Harvester）1.8.x - 1.9.x（NFS、iSCSI、NVMe/TCP 和 FC）
- 适用于 NFSv4、iSCSI 和 NVMe/TCP 的 Talos OS。每种协议都需要以下 Talos 系统扩展；请与 Sidero Labs 确认这些要求。
  - NFSv4： `siderolabs/nfs-utils`
  - iSCSI： `siderolabs/iscsi-tools`、 `siderolabs/multipath-tools`和 `siderolabs/trident-iscsi-tools`
  - NVMe/TCP： `siderolabs/nvme-cli`、 `siderolabs/util-linux-tools`和 `siderolabs/trident-iscsi-tools`

Trident还可以与许多其他完全托管和自行管理的Kubernetes产品配合使用、包括Google Kubernetes Engine (GKE)、Amazon Elatic Kubernetes Services (EKS)、Azure Kubernetes Service (AKS)、MiranT斯Kubernetes Engine (MKE)和VMware Tanzu产品组合。

Trident和ONTAP可用作的存储提供程序"KubeVirt"。

> 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，请参见"升级Helm安装"。

支持的后端（存储）

要使用Trident、您需要使用以下一个或多个受支持的后端：

- 适用于 NetApp ONTAP 的 Amazon FSX
- Azure NetApp Files

- Cloud Volumes ONTAP
- Google Cloud NetApp卷
- 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）。看 "软件版本支持"。
- NetApp HCI/ Element软件11或更高版本

Trident 支持 KubeVirt、OpenShift 虚拟化和 SUSE 虚拟化

支持的存储驱动程序：

Trident 支持以下用于 KubeVirt、OpenShift 虚拟化和 SUSE 虚拟化的 ONTAP 驱动程序：

- ontap-NAS
- NVMe-SAN (iSCSI、FCP、基于的ONTAP)
- ONTAP—SAN经济型(仅限iSCSI)

SUSE 虚拟化认证报告

使用 SUSE Virtualization 验证了以下 Trident 26.10 配置。测试由 Suresh Thoppay 提供：

- "采用 SUSE Virtualization 1.8.2 的 Trident 26.10"
- "使用 SUSE Virtualization 1.9.1 的 Trident 26.10"

注意事项：

- 在OpenShift虚拟化环境中更新存储类以使用 `fsType`参数（例如： `fsType: "ext4"`）。如果需要、可使用中的参数 `dataVolumeTemplates`将卷模式明确设置为阻止 `volumeMode=Block`、以通知数据卷创建者。
- *rwx*块存储驱动程序访问模式：只有"volumemode：block"(原始设备)才支持ONTAP SAN (iSCSI、NVMe/TCP、FC)和ONTAP SAN经济型(iSCSI)驱动程序。对于这些驱动程序、 `fstype`无法使用参数、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)

Trident 支持 Kubernetes 上的 Kubernetes 和 OpenShift 托管控制平面

Trident 完全支持 Kubernetes on Kubernetes 拓扑。Kubernetes 集群，其节点作为 KubeVirt 虚拟机在基础架构（主机）集群上运行。这种拓扑结构是 OpenShift 虚拟化上的 OpenShift 托管控制平面（HCP）等托管控制平面产品的基础。

此配置在早期版本中可用，但全面认证从 26.06 开始。

来宾（租户）集群使用 KubeVirt CSI 驱动程序(`kubevirt-csi`）作为其存储提供程序。当来宾集群中的工作负载请求 `PersistentVolumeClaim`时，KubeVirt CSI 驱动程序会在基础架构集群中创建相应的卷。Trident 从 ONTAP 配置卷，并将卷热插入来宾节点虚拟机。Trident 仅在基础架构集群中运行。来宾集群通过 KubeVirt CSI 驱动程序间接使用 ONTAP 存储。

要求：

- Trident 已安装在具有受支持的 ONTAP 后端的基础架构集群上。
- 基础设施集群中存在 Trident 存储类，供 KubeVirt CSI 驱动程序将来宾集群卷映射到其上。
- KubeVirt 或 OpenShift Virtualization 部署在基础架构集群上，KubeVirt CSI 驱动程序安装在每个访客集群中。

组件	**version**
Trident	26.10
Kubernetes（基础设施和来宾集群）	1.37
KubeVirt	1.9.0
容器化数据导入器（CDI）	1.65

注意事项：

- 在来宾集群中，卷在来宾节点虚拟机上显示为热插拔块设备。
- KubeVirt CSI 驱动程序工作流不需要在访客集群中使用 Trident。您可以在基础架构集群中配置 ONTAP 后端和存储类。

直接在来宾集群中安装 **Trident**：

您也可以直接在访客（租户）集群中安装 Trident。当您需要完整的 Trident 功能集（例如快照、克隆、卷调整大小和导入）时，请在访客集群中安装 Trident，这些功能 `kubevirt-csi`本身并不提供。

在来宾群集中，使用基于 IP 的协议：NFS、iSCSI 和 NVMe/TCP。来宾群集不支持光纤通道 (FC)，因为来宾节点是没有直接 FC 连接的虚拟机。

功能要求

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

功能	**Kubernetes** 版本	是否需要功能安全门?
Trident	1.27 - 1.37	否
卷快照	1.27 - 1.37	否
卷快照中的 PVC	1.27 - 1.37	否
iSCSI PV 调整大小	1.27 - 1.37	否
ONTAP 双向 CHAP	1.27 - 1.37	否
动态导出策略	1.27 - 1.37	否
Trident 运算符	1.27 - 1.37	否
CSI 拓扑	1.27 - 1.37	否

已测试主机操作系统

虽然Trident不正式支持特定操作系统、但已知以下操作系统可以正常工作：

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

> NVMe/TCP需要RHEL 9或更高版本。

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

默认情况下、Trident在容器中运行、因此将在任何Linux工作器上运行。但是、根据您使用的后端、这些员工需要能够使用标准NFS客户端或iSCSI启动程序挂载Trident提供的卷。

`tridentctl` 实用程序也可在 Linux 的任何这些分发版上运行。

主机配置

Kubernetes集群中的所有工作节点都必须能够挂载为Pod配置的卷。要准备工作节点、必须根据您选择的驱动程序安装NFS、iSCSI或NVMe工具。

"准备工作节点"

存储系统配置：

Trident可能需要先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需要访问特定端口才能进行通信。

"Trident端口"

容器映像以及相应的 **Kubernetes** 版本

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

下面的 Trident 26.10 映像列表包括 Kubernetes 1.37 的映像。

Trident 26.10.1 所需的容器镜像

Kubernetes版本	容器映像
v1.37.0	• docker.io/netapp/trident:26.10.1 • docker.io/netapp/trident-autosupport:26.10 • docker.io/netapp/trident-operator:26.10.1（可选） • registry.k8s.io/sig-storage/csi-attacher:v4.11.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.17.0 • registry.k8s.io/sig-storage/csi-provisioner:v6.3.0 • registry.k8s.io/sig-storage/csi-resizer:v2.1.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.6.0

Trident 26.06.2 所需的容器镜像

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, v1.35.0, v1.36.0	• docker.io/netapp/trident:26.06.2 • docker.io/netapp/trident-autosupport:26.06 • docker.io/netapp/trident-operator:26.06.2（可选） • registry.k8s.io/sig-storage/csi-attacher:v4.11.0 • registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.18.0 • registry.k8s.io/sig-storage/csi-provisioner:v6.3.0 • registry.k8s.io/sig-storage/csi-resizer:v2.1.0 • registry.k8s.io/sig-storage/csi-snapshotter:v8.6.0

安装 Trident

了解Trident安装

为了确保Trident可以安装在各种环境和组织中、NetApp提供了多种安装选项。您可以使用Trident操作员(手动或使用Helm)或与一起安装Trident `tridentctl`。本主题提供了有关为您选择正确安装过程的重要信息。

有关 Trident 26.10 的关键信息

您必须阅读以下有关**Trident**的重要信息。

▼ **与Trident **相关的信息

- Trident 现在支持 Kubernetes 1.37。在升级 Kubernetes 之前，请先升级 Trident。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

开始之前

无论您的安装路径如何、您都必须：

- 运行受支持的 Kubernetes 版本的受支持的 Kubernetes 集群的完整权限，并启用了功能要求。查看 "要求" 了解详情。
- 访问受支持的NetApp存储系统。
- 能够从所有Kubernetes工作节点挂载卷。
- 具有的Linux主机 `kubectl` (或 `oc`(如果您使用的是OpenShift)已安装并配置为管理要使用的Kubernetes集群。
- 。 `KUBECONFIG` 环境变量设置为指向您的Kubernetes集群配置。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。
- 集群必须支持有权限的工作负载。

如果您还没有熟悉"基本概念"，现在是一个好时机。

选择安装方法

选择适合您的安装方法。您还应该在做出决定之前查看 "在方法之间移动" 的注意事项。

使用**Trident**运算符

无论是手动部署还是使用 Helm，Trident 操作员都是简化安装和动态管理 Trident 资源的好方法。您甚至可以"自定义Trident操作员部署"使用 `TridentOrchestrator`自定义资源 (CR) 中的属性。

使用Trident运算符的优势包括：

▼ ** Trident对象**

Trident操作符会自动为您的Kubernetes版本创建以下对象。

- 操作员的ServiceAccount
- ClusterRole和ClusterRoleBindingto the ServiceAccount
- 专用PodSecurityPolicy (适用于Kubernetes 1.25及更早版本)
- 运算符本身

▼ **帐户**

集群范围的Trident操作员在集群级别管理与Trident安装关联的资源。这可减少使用命名空间范围的操作符维护集群范围的资源时可能导致的错误。这对于自我修复和修补至关重要。

▼ ** —修复功能—**

操作员监控Trident安装并主动采取措施解决问题、例如何时删除部署或意外修改部署。此时将创建一个`trident-operator-<generated-id>`POD、将CR与Trident安装关联起来`TridentOrchestrator`。这样可以确保集群中只有一个Trident实例、并控制其设置、从而确保安装有效。对安装进行更改（例如删除部署或节点取消设置）时，操作员会识别这些更改并逐个修复它们。

▼ **** 更新了现有安装的****

您可以使用操作员轻松更新现有部署。您只需编辑 `TridentOrchestrator` cr以更新安装。

例如、假设您需要启用Trident以生成调试日志。要执行此操作，请修补 `TridentOrchestrator`以将设置`spec.debug`为 `true`：

```
kubectl patch torc <trident-orchestrator-name> -n trident --type=merge
-p '{"spec":{"debug":true}}'
```

之后 `TridentOrchestrator` 更新后、操作员将处理更新并修补现有安装。这可能会触发创建新Pod以相应地修改安装。

▼ ** Clean**重新安装****

通过集群范围的三端技术参数操作符、可以完全删除集群范围的资源。用户可以完全卸载Trident并轻松重新安装。

▼ ** Kubernetes**升级****

将集群的Kubnetes版本升级到受支持的版本后、操作员会自动更新现有Trident安装并进行更改、以确保满足Kubnetes版本的要求。

如果集群升级到不受支持的版本、则操作员将阻止安装Trident。如果已使用操作员安装Trident、则会显示一条警告、指示Trident安装在不受支持的Kubnetes版本上。

使用 **tridentctl**

如果您的现有部署必须升级，或者您希望高度自定义您的部署，则应考虑。这是部署Trident的传统方法。

您可以为Trident资源生成清单。其中包括Trident在安装过程中创建的部署、守护进程、服务帐户和集群角色。

> 从22.04版开始、每次安装Trident时、不会再重新生成AES密钥。在此版本中、Trident将安装一个新的机密对象、此对象会在安装期间持续存在。这意味着、 `tridentctl`在22.04中可以卸载早期版本的Trident、但早期版本无法卸载22.04安装。选择适当的安装_method。

选择安装模式

根据您的组织所需的_installation mode"(标准)、"Offline"(脱机)或"Remote"(远程)来确定部署过程。

标准安装

这是安装Trident的最简单方法、适用于不施加网络限制的大多数环境。标准安装模式使用默认注册表来存储所需的Trident (`docker.io`()和CSI (`registry.k8s.io`()图像。

使用标准模式时、Trident安装程序将：

- 通过Internet提取容器映像
- 创建部署或节点守护进程、以便在Kubornetes集群中所有符合条件的节点上启动Trident Pod

脱机安装

在带风口或安全位置可能需要脱机安装模式。在这种情况下、您可以创建一个专用的镜像注册表或两个镜像注册表来存储所需的Trident和CSI映像。

> 无论注册表配置如何、CSI映像都必须驻留在一个注册表中。

远程安装

下面简要概述了远程安装过程：

- 在要从中部署Trident的远程计算机上部署适当版本的 `kubectl`。
- 从 Kubernetes 集群复制配置文件，并在远程计算机上设置 `KUBECONFIG` 环境变量。
- 启动 `kubectl get nodes` 命令，验证您是否可以连接到所需的 Kubernetes 集群。
- 使用标准安装步骤从远程计算机完成部署。

根据您的方法和模式选择过程

做出决定后、请选择相应的流程。

方法	安装模式
Trident运算符(手动)	"标准安装" "脱机安装"

方法	安装模式
Trident运算符(Helm)	"标准安装" "脱机安装"
`tridentctl`	"标准或脱机安装"

在安装方法之间移动

您可以决定更改安装方法。在执行此操作之前、请考虑以下事项：

- 请始终使用相同的方法安装和卸载Trident。如果已使用部署 `tridentctl`，则应使用适当版本的 `tridentctl` 二进制文件卸载Trident。同样、如果使用操作员进行部署、则应编辑 `TridentOrchestrator`CR并设置 `spec.uninstall=true`为卸载Trident。
- 如果要删除基于操作员的部署并使用它 `tridentctl`来部署`Trident`、则应首先编辑并设置 `spec.uninstall=true`为卸载`Trident`。 `TridentOrchestrator`然后删除 `TridentOrchestrator`和操作员部署。然后，您可以使用安装 `tridentctl`。
- 如果您有一个基于手动操作员的部署，并且想要使用基于 Helm 的Trident操作员部署，则应先手动卸载操作员，然后执行 Helm 安装。这使得 Helm 能够使用所需的标签和注释来部署Trident操作符。如果不这样做，基于 Helm 的Trident操作员部署将因标签验证错误和注释验证错误而失败。
- 如果你有一个 `tridentctl`基于部署，无需卸载Trident即可进行基于 Helm 或基于 Operator 的部署。

其他已知配置选项

在VMware Tanzu产品组合上安装Trident时：

- ` -kubelet-dir` 标志应设置为 kubelet 目录的位置。默认情况下，此值为 ` /var/vcap/data/kubelet` 。

  已知使用 ` -kubelet-dir` 指定 kubelet 位置适用于 Trident Operator ， Helm 和 `tridentctl` 部署。

使用Trident操作员安装

手动部署Trident操作员(标准模式)

您可以手动部署Trident操作员以安装Trident。此过程适用于Trident所需容器映像未存储在私有注册表中的安装。如果您有私有映像注册表，请使用"脱机部署过程"。

有关 Trident 26.10 的关键信息

您必须阅读以下有关Trident的重要信息。

▼ 与Trident 相关的信息

- Trident 26.10 支持 Kubernetes 1.27 到 1.37。在升级 Kubernetes 之前，请检查 "Trident要求" 中的特定版本镜像列表。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

手动部署Trident操作员并安装Trident

请检查 "安装概述" 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

在开始安装之前，请登录到 Linux 主机并验证其是否正在管理工作，"支持的 Kubernetes 集群"以及您是否具有必要的权限。

使用 OpenShift 时，在以下所有示例中使用 `oc` 而不是 `kubectl` ，并首先以 * 系统： admin* 的身份运行 `oc login -u system ： admin` 或 `oc login -u Kube-admin` 进行登录。

▼ **Details**

1. 验证Kubernetes版本：

```
kubectl version
```

2. 验证集群管理员权限：

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
  ping <management IP>
```

第1步：下载Trident安装程序包

Trident安装程序包包含部署Trident Operator和安装Trident所需的一切。从下载并解压缩最新版本的Trident安装程序"GitHub上的_assets_部分"。

```
wget https://github.com/NetApp/trident/releases/download/v26.10.1/trident-
installer-26.10.1.tar.gz
tar -xf trident-installer-26.10.1.tar.gz
cd trident-installer
```

第2步：创建 `TridentOrchestrator` CRD

创建 `TridentOrchestrator`自定义资源定义（CRD）。您将创建一个 `TridentOrchestrator`稍后自定义资源。使用适当的 CRD YAML 版本 `deploy/crds`创建 `TridentOrchestrator`CRD。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

第3步：部署Trident操作员

Trident安装程序提供了一个可用于安装操作员和创建关联对象的捆绑包文件。通过捆绑包文件、可以使用默认配置轻松部署操作员并安装Trident。

- 对于运行Kubbernetes 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

- 默认情况下、通过使用三端安装程序、可以在中部署操作员 `trident` 命名空间。如果 `trident` 命名空间不存在、请使用以下命令创建命名空间：

```
kubectl apply -f deploy/namespace.yaml
```

- 在非命名空间中部署操作员 `trident` 命名空间、更新 `serviceaccount.yaml`，`clusterrolebinding.yaml` 和 `operator.yaml` 并使用生成捆绑包文件 `kustomization.yaml`。

  a. 创建 `kustomization.yaml` 使用以下命令、其中_<bundle.yaml>_为 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml` 根据您的Kubbernetes版本。

  ```
  cp deploy/kustomization_<bundle.yaml> deploy/kustomization.yaml
  ```

  b. 使用以下命令编译分发包、其中_Data <bundle.yaml>是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml` 根据您的Kubbernetes版本。

  ```
  kubectl kustomize deploy/ > deploy/<bundle.yaml>
  ```

步骤

1. 创建资源并部署操作员：

```
kubectl create -f deploy/<bundle.yaml>
```

2. 验证是否已创建操作员、部署和副本集。

```
kubectl get all -n <operator-namespace>
```

> 在 Kubernetes 集群中只能有 * 一个操作符实例 * 。请勿创建 Trident 操作员的多个部署。

第4步：创建 `TridentOrchestrator` 并安装Trident

现在、您可以创建 `TridentOrchestrator`并安装Trident。您也可以选择"自定义Trident安装"使用规范中的属性 `TridentOrchestrator。

```
kubectl create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

kubectl describe torc trident

Name:         trident
Namespace:
Labels:       <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
  nodePrep:
  - iscsi
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:26.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:26.10.1
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v26.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

验证安装。

可以通过多种方法验证您的安装。

使用 `TridentOrchestrator` status

的状态 `TridentOrchestrator` 指示安装是否成功、并显示已安装的Trident版本。在安装期间、的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态、并且操作员无法自行恢复、 "检查日志"。

Status	Description
安装	操作员正在使用此CR安装Trident `TridentOrchestrator`。
已安装	已成功安装Trident。
正在卸载	操作员正在卸载Trident，因为 `spec.uninstall=true`。
已卸载	Trident已卸载。
失败	操作员无法安装、修补、更新或卸载Trident；操作员将自动尝试从此状态中恢复。如果此状态仍然存在，则需要进行故障排除。
正在更新	操作员正在更新现有安装。
error	不使用 `TridentOrchestrator` 。另一个已存在。

正在使用POD创建状态

您可以通过查看已创建Pod的状态来确认Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS
AGE
trident-controller-7d466bf5c7-v4cpw        6/6     Running   0
1m
trident-node-linux-mr6zc                   2/2     Running   0
1m
trident-node-linux-xrp7w                   2/2     Running   0
1m
trident-node-linux-zh2jt                   2/2     Running   0
1m
trident-operator-766f7b8658-ldzsv          1/1     Running   0
3m
```

使用 `tridentctl`

您可以使用 `tridentctl`检查安装的Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.1        | 26.10.1        |
+----------------+----------------+
```

手动部署**Trident**操作员(脱机模式)

您可以手动部署Trident操作员以安装Trident。此过程适用于将Trident所需的容器映像存储在私有注册表中的安装。如果没有专用映像注册表，请使用"标准部署流程"。

有关**Trident**的关键信息

您必须阅读以下有关**Trident**的重要信息。

▼ **与Trident **相关的信息

- Trident 26.10 支持 Kubernetes 1.27 到 1.37。在升级 Kubernetes 之前，请检查 "Trident要求" 中的特定版本镜像列表。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

手动部署**Trident**操作员并安装**Trident**

请检查 "安装概述" 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

登录到 Linux 主机并验证其是否正在管理工作和"支持的 Kubernetes 集群"并且您具有必要的权限。

> 使用 OpenShift 时，在以下所有示例中使用 `oc` 而不是 `kubectl` ，并首先以 * 系统： admin* 的身份运行 `oc login -u system ： admin` 或 `oc login -u Kube-admin` 进行登录。

▼ **Details**

1. 验证Kubernetes版本：

   ```
   kubectl version
   ```

2. 验证集群管理员权限：

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
  ping <management IP>
```

第1步：下载Trident安装程序包

Trident安装程序包包含部署Trident Operator和安装Trident所需的一切。从下载并解压缩最新版本的Trident安装程序"GitHub上的_assets_部分"。

```
wget https://github.com/NetApp/trident/releases/download/v26.10.1/trident-
installer-26.10.1.tar.gz
tar -xf trident-installer-26.10.1.tar.gz
cd trident-installer
```

第2步：创建 `TridentOrchestrator` CRD

创建 `TridentOrchestrator`自定义资源定义（CRD）。您将创建一个 `TridentOrchestrator`稍后自定义资源。使用适当的 CRD YAML 版本 `deploy/crds`创建 `TridentOrchestrator`CRD：

```
kubectl create -f deploy/crds/<VERSION>.yaml
```

第3步：更新操作符中的注册表位置

在 `/deploy/operator.yaml`中，更新 `image: docker.io/netapp/trident-operator:26.10.1`以反映图像注册表的位置。您的 "Trident和CSI映像" 可以位于一个注册表或不同的注册表中，但所有 CSI 映像必须位于同一注册表中。例如：

- `image: <your-registry>/trident-operator:26.10.1` 如果您的镜像都位于一个注册表中。
- `image: <your-registry>/netapp/trident-operator:26.10.1` 如果您的 Trident 镜像位于与 CSI 镜像不同的注册表中。

第4步：部署Trident操作员

Trident安装程序提供了一个可用于安装操作员和创建关联对象的捆绑包文件。通过捆绑包文件、可以使用默认配置轻松部署操作员并安装Trident。

- 对于运行Kubbernetes 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

- 默认情况下、通过使用三端安装程序、可以在中部署操作员 `trident` 命名空间。如果 `trident` 命名空间

不存在、请使用以下命令创建命名空间：

```
kubectl apply -f deploy/namespace.yaml
```

- 在非命名空间中部署操作员 `trident` 命名空间、更新 `serviceaccount.yaml`，`clusterrolebinding.yaml` 和 `operator.yaml` 并使用生成捆绑包文件 `kustomization.yaml`。

  a. 创建 `kustomization.yaml` 使用以下命令、其中_<bundle.yaml>_为 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml` 根据您的Kubbernetes版本。

  ```
  cp deploy/kustomization_<bundle.yaml> deploy/kustomization.yaml
  ```

  b. 使用以下命令编译分发包、其中_Data <bundle.yaml>是 `bundle_pre_1_25.yaml` 或 `bundle_post_1_25.yaml` 根据您的Kubbernetes版本。

  ```
  kubectl kustomize deploy/ > deploy/<bundle.yaml>
  ```

步骤

1. 创建资源并部署操作员：

```
kubectl create -f deploy/<bundle.yaml>
```

2. 验证是否已创建操作员、部署和副本集。

```
kubectl get all -n <operator-namespace>
```

在 Kubernetes 集群中只能有 * 一个操作符实例 * 。请勿创建 Trident 操作员的多个部署。

第5步：更新中的映像注册表位置 `TridentOrchestrator`

Your "Trident和CSI映像" 可以位于一个注册表或不同的注册表中、但所有CSI映像都必须位于同一注册表中。更新 `deploy/crds/tridentorchestrator_cr.yaml` 根据注册表配置添加其他位置规格。

一个注册表中的映像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.10"
tridentImage: "<your-registry>/trident:26.10.1"
```

不同注册表中的映像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:26.10"
tridentImage: "<your-registry>/trident:26.10.1"
```

第6步：创建 `TridentOrchestrator` 并安装Trident

现在、您可以创建 `TridentOrchestrator`并安装Trident。您也可以进一步"自定义Trident安装"使用规范中的属性 `TridentOrchestrator。以下示例显示了Trident和CSI映像位于不同注册表中的安装。

```
kubectl create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

kubectl describe torc trident

Name:         trident
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
...
Spec:
  Autosupport Image:  <your-registry>/trident-autosupport:26.10
  Debug:              true
  Image Registry:     <your-registry>
  Namespace:          trident
  Trident Image:      <your-registry>/trident:26.10.1
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         <your-registry>/trident-autosupport:26.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Http Request Timeout:      90s
    Image Pull Secrets:
    Image Registry:       <your-registry>
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
```

```
    Trident Image:          <your-registry>/trident:26.10.1
  Message:                  Trident installed
  Namespace:                trident
  Status:                   Installed
  Version:                  v26.10.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

验证安装。

可以通过多种方法验证您的安装。

使用 `TridentOrchestrator` status

的状态 `TridentOrchestrator` 指示安装是否成功、并显示已安装的Trident版本。在安装期间、的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态、并且操作员无法自行恢复、 "检查日志"。

Status	Description
安装	操作员正在使用此CR安装Trident `TridentOrchestrator`。
已安装	已成功安装Trident。
正在卸载	操作员正在卸载Trident，因为 `spec.uninstall=true`。
已卸载	Trident已卸载。
失败	操作员无法安装、修补、更新或卸载Trident；操作员将自动尝试从此状态中恢复。如果此状态仍然存在，则需要进行故障排除。
正在更新	操作员正在更新现有安装。
error	不使用 `TridentOrchestrator` 。另一个已存在。

正在使用POD创建状态

您可以通过查看已创建Pod的状态来确认Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS
AGE
trident-controller-7d466bf5c7-v4cpw        6/6     Running   0
1m
trident-node-linux-mr6zc                   2/2     Running   0
```

```
1m
trident-node-linux-xrp7w                    2/2     Running   0
1m
trident-node-linux-zh2jt                    2/2     Running   0
1m
trident-operator-766f7b8658-ldzsv           1/1     Running   0
3m
```

使用 **tridentctl**

您可以使用 `tridentctl`检查安装的Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.1        | 26.10.1        |
+----------------+----------------+
```

使用Helm部署Trident操作员(标准模式)

您可以使用Helm部署Trident操作员并安装Trident。此过程适用于Trident所需容器映像未存储在私有注册表中的安装。如果您有私有映像注册表，请使用"脱机部署过程"。

有关 Trident 26.10 的关键信息

您必须阅读以下有关Trident的重要信息。

▼ 与Trident 相关的信息

- Trident 26.10 支持 Kubernetes 1.27 到 1.37。在升级 Kubernetes 之前，请检查 "Trident要求" 中的特定版本镜像列表。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

部署Trident操作员并使用Helm安装Trident

使用Trident "Helm图表" 您可以一步部署Trident操作员并安装Trident。

请检查 "安装概述" 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

除了 "部署前提条件" 之外，您还需要 "Helm 版本 3"。

步骤

1. 添加Trident Helm存储库：

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

2. 使用 `helm install`并指定部署的名称，如下例所示，其中 `100.2610.0`是您要安装的 Trident 版本。

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace <trident-namespace>
```

如果已为Trident创建命名空间、则`-create-namespace`参数不会创建额外的命名空间。

您可以使用 `helm list` 查看安装详细信息、例如名称、命名空间、图表、状态、应用程序版本、和修订版号。

在安装期间传递配置数据

在安装期间，可以通过两种方式传递配置数据：

选项	Description
`--values` (或 `-f`)	指定包含覆盖的YAML文件。可以多次指定此值，最右侧的文件将优先。
`--set`	在命令行上指定覆盖。

例如，要更改 `debug`的默认值，请运行以下命令，其中 `100.2610.0`是您要安装的 Trident 版本：

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace trident --set tridentDebug=true
```

配置选项

此表和 `values.yaml` 文件(属于Helm图表的一部分)提供了键列表及其默认值。

选项	Description	Default
`nodeSelector`	用于POD分配的节点标签	
`podAnnotations`	POD标注	
`deploymentAnnotations`	部署标注	

选项	Description	Default
`tolerations`	POD分配的差值	

`affinity`	用于Pod分配的相关性	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedulingIgnoredDuringExecution:`
		`  nodeSelectorTerms:`
		`  - matchExpressions:`
		`  - key: kubernetes.io/arch`
		`  operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key: kubernetes.io/os`
		`  operator: In`
		`  values:`
		`  - linux`
		
		请勿从values.yaml文件中删除默认相关性。如果要提供自定义相关性、请扩展默认相关性。

`tridentControllerPluginNodeSelector`	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
`tridentControllerPluginTolerations`	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
`tridentNodePluginNodeSelector`	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
`tridentNodePluginTolerations`	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
`imageRegistry`	标识、 `trident`和其它图像的注册表 `trident-operator`。留空以接受默认值。重要信息：在私有存储库中安装Trident时、如果`imageRegistry`使用交换机指定存储库位置、请勿在存储库路径中使用 `/netapp/`。	""

选项	Description	Default
imagePullPolicy	设置的映像提取策略 trident-operator。	IfNotPresent
imagePullSecs	设置的映像提取密钥 trident-operator， `trident`和其他图像。	
kubeletDir	允许覆盖kubelet内部状态的主机位置。	"/var/lib/kubelet"
operatorLogLevel	允许将Trident操作符的日志级别设置为： trace， debug， info， warn， error`或 `fatal。	"info"
operatorDebug	允许将Trident操作符的日志级别设置为DEBUG。	true
operatorImage	允许完全覆盖的映像 trident-operator。	""
operatorImageTag	允许覆盖的标记 trident-operator 图像。	""
tridentIPv6	允许在IPv6集群中启用Trident。	false
tridentK8sTimeout	覆盖大多数Kubernetes API操作的默认30秒超时(如果不为零、则以秒为单位)。	0
tridentHttpRequestTimeout	使用覆盖HTTP请求的默认90秒超时 0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutosupport	允许禁用Trident定期AutoSupport报告。	false
tridentExcludeAutosupport	当设置为 `true`时，Trident 在安装期间不会设置 AutoSupport 容器。	false
tridentAutosupportImageTag	允许覆盖Trident AutoSupport容器的映像标记。	<version>
tridentAutosupportProxy	允许Trident AutoSupport容器通过HTTP代理回拨。	""
tridentLogFormat	设置Trident日志记录格式(text`或 `json)。	"text"
tridentDisableAuditLog	禁用Trident审核日志程序。	true
tridentLogLevel	允许将Trident的日志级别设置为： trace、 debug info、、、 warn error`或 `fatal。	"info"

选项	Description	Default
tridentDebug	允许将Trident的日志级别设置为debug。您可以通过与节点健康检查 (NHC) 操作符集成来自动执行强制分离过程。有关信息，请参阅"使用Trident实现有状态应用程序的故障转移自动化"。	false
tridentLogWorkflows	允许为跟踪日志记录或日志抑制启用特定的Trident工作流。	""
tridentLogLayers	允许为跟踪日志记录或日志抑制启用特定的Trident层。	""
TridentImage	允许完全覆盖Trident的图像。	""
tridentImageTag	允许覆盖Trident的映像标记。	""
tridentProbePort	允许覆盖用于Kubernetes活动/就绪性探测的默认端口。	""
windows	使Trident能够安装在Windows工作节点上。	false
enableForceDetach	允许启用强制分离功能。	false
enableDataLIFRefresh	当 LIF 被添加、删除或分配新地址时，刷新已发布卷的 NVMe 数据 LIF。适用于 ontap-san（含 sanType: nvme）（在统一 ONTAP 上）以及 ASA r2 上的 NVMe/TCP。有关更多信息，请参见"刷新 NVMe 数据 LIF"。	false
excludePodSecurityPolicy	从创建过程中排除操作员POD安全策略。	false
cloudProvider	在 AKS 集群上使用托管标识或云标识时设置为 "Azure"，在 EKS 集群上使用云标识时设置为 "AWS"，或者在 GKE 上使用云标识或在 OpenShift 上使用 Workload Identity Federation 时设置为 "GCP"。	""

选项	Description	Default
`cloudIdentity`	在 AKS 集群上使用云标识时，设置为工作负载标识（"azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"）。在 EKS 集群上使用云标识时，设置为 AWS IAM 角色（'"eks.amazonaws.com/role-arn: arn:aws:iam::123456:role/trident-role"'）。在 GKE 上使用云标识时，设置为 GKE 云标识注释。请勿为 OpenShift Workload Identity Federation 设置。	""
`gcpWIFAudience`	在 OpenShift 上与 Google Cloud Workload Identity Federation 一起使用的投影服务帐户令牌的受众。使用提供商的规范资源名称。有关详细信息，请参见 "使用 Workload Identity Federation 配置 Google Cloud NetApp Volumes"。	""
`iscsiSelfHealingInterval`	调用iSCSI自我修复的间隔。	`5m0s`
`iscsiSelfHealingWaitTime`	iSCSI自我修复通过执行注销和后续登录来尝试解决陈旧会话之前经过的持续时间。	`7m0s`
`nodePrep`	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。 *现在， `iscsi`是唯一支持的值。*注意：从 OpenShift 4.19 开始，此功能支持的最低Trident版本为 25.06.1。	
`enableConcurrency`	启用并发 Trident 控制器操作，以提高吞吐量。有关支持的驱动程序的详细信息，请参见链接：https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability[Controller Scalability documentation]。	false
`k8sAPIQPS`	控制器与 Kubernetes API 服务器通信时使用的每秒查询数 (QPS) 限制。 Burst 值根据 QPS 值自动设置。	`100`; 选修的

选项	Description	Default

`resources`	设置 Kubernetes 对Trident控制器、节点和操作员 pod 的资源限制和请求。您可以为每个容器和边车配置 CPU 和内存，以管理 Kubernetes 中的资源分配。
	
	有关配置资源请求和限制的更多信息，请参阅"Pod 和容器的资源管理"。
	
	• 请勿更改任何容器或字段的名称。
	• 请勿更改缩进 - YAML 缩进对于正确解析至关重要。
	
	• 默认情况下不应用任何限制——只有请求才有默认值，如果未指定则自动应用。
	• 容器名称按其在 pod 规范中的显示方式列出。
	• 边车列在每个主集装箱下方。
	• 检查TORC的 `status.CurrentInstallationParams `用于查看当前应用值的字段。
		`resources:`
		`  controller:`
		`  trident-main:`
		`  requests:`
		`  cpu: 10m`
		`  memory: 80Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-provisioner:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-attacher:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-resizer:`
		`  requests:`
		`  cpu: 3m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  csi-snapshotter:`
		`  requests:`
		`  cpu: 2m`
		`  memory: 20Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  trident-autosupport:`
		`  requests:`
		`  cpu: 1m`
		`  memory: 30Mi`
		`  limits:`
		`  cpu:`
		`  memory:`
		`  node:`

选项	Description	Default
`httpsMetrics`	为 Prometheus 指标端点启用 HTTPS。	false
`hostNetwork`	为Trident控制器启用主机联网功能。在多宿主网络中，当需要分离前端和后端流量时，这非常有用。	false

了解控制器**Pod**和节点**Pod**

Trident作为单个控制器Pod以及集群中每个工作节点上的一个节点Pod运行。节点Pod必须运行在可能要挂载Trident卷的任何主机上。

Kubernetes "节点选择器" 和 "容忍和损害" 用于限制Pod在特定节点或首选节点上运行。使用`ControllerPlugin`和 `NodePlugin`、您可以指定约束和覆盖。

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

使用**Helm**部署**Trident**操作员(脱机模式)

您可以使用Helm部署Trident操作员并安装Trident。此过程适用于将Trident所需的容器映像存储在私有注册表中的安装。如果没有专用映像注册表，请使用"标准部署流程"。

有关 **Trident 26.10** 的关键信息

您必须阅读以下有关**Trident**的重要信息。

▼ ****与**Trident **相关的信息

- Trident 26.10 支持 Kubernetes 1.27 到 1.37。在升级 Kubernetes 之前，请检查 "Trident要求" 中的特定版本镜像列表。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

部署**Trident**操作员并使用**Helm**安装**Trident**

使用Trident "Helm图表" 您可以一步部署Trident操作员并安装Trident。

请检查 "安装概述" 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

除了 "部署前提条件" 之外，您还需要 "Helm 版本 3"。

在私有存储库中安装Trident时、如果 `imageRegistry`使用交换机指定存储库位置、请勿在存

储库路径中使用 `/netapp/。

步骤

1. 添加Trident Helm存储库：

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

2. 使用 `helm install`并指定部署和映像注册表位置的名称。您的 "Trident和CSI映像" 可以位于一个注册表或不同的注册表中，但所有 CSI 映像必须位于同一注册表中。在示例中， `100.2610.0`是您要安装的 Trident 版本。

一个注册表中的映像

```
helm install <name> netapp-trident/trident-operator --version
100.2610.0 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace> --set nodePrep={iscsi}
```

不同注册表中的映像

```
helm install <name> netapp-trident/trident-operator --version
100.2610.0 --set imageRegistry=<your-registry> --set operatorImage
=<your-registry>/trident-operator:26.10.1 --set
tridentAutosupportImage=<your-registry>/trident-autosupport:26.10
--set tridentImage=<your-registry>/trident:26.10.1 --create
-namespace --namespace <trident-namespace> --set nodePrep={iscsi}
```

如果已为Trident创建命名空间、则`-create-namespace`参数不会创建额外的命名空间。

您可以使用 `helm list` 查看安装详细信息、例如名称、命名空间、图表、状态、应用程序版本、和修订版号。

在安装期间传递配置数据

在安装期间，可以通过两种方式传递配置数据：

选项	Description
`--values` (或 `-f`)	指定包含覆盖的YAML文件。可以多次指定此值，最右侧的文件将优先。
`--set`	在命令行上指定覆盖。

例如，要更改 `debug`的默认值，请运行以下命令，其中 `100.2610.0`是您要安装的 Trident 版本：

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
```

```
--create-namespace --namespace trident --set tridentDebug=true
```

要添加nodePrep值、请运行以下命令：

```
helm install <name> netapp-trident/trident-operator --version 100.2610.0
--create-namespace --namespace trident --set nodePrep={iscsi}
```

配置选项

此表和 `values.yaml` 文件(属于Helm图表的一部分)提供了键列表及其默认值。

请勿从values.yaml文件中删除默认相关性。如果要提供自定义相关性、请扩展默认相关性。

选项	Description	Default
`nodeSelector`	用于POD分配的节点标签	
`podAnnotations`	POD标注	
`deploymentAnnotations`	部署标注	
`tolerations`	POD分配的差值	

选项	Description	Default

`affinity`	用于Pod分配的相关性	`affinity:`
		`  nodeAffinity:`
		
		`requiredDuringSchedul`
		`ingIgnoredDuringExecu`
		`tion:`
		
		`nodeSelectorTerms:`
		`  -`
		`matchExpressions:`
		`  - key:`
		`kubernetes.io/arch`
		
		`operator: In`
		`  values:`
		`  - arm64`
		`  - amd64`
		`  - key:`
		`kubernetes.io/os`
		
		`operator: In`
		`  values:`
		`  - linux`
		
		请勿从values.yaml文件中删除默认相关性。如果要提供自定义相关性、请扩展默认相关性。

`tridentControllerPluginNodeSelector`	Pod的其他节点选择器。请参见 "了解控制器Pod和节点Pod" 了解详细信息。	
`tridentControllerPluginTolerations`	覆盖Kubernetes对Pod的容错。请参见 "了解控制器Pod和节点Pod" 了解详细信息。	
`tridentNodePluginNodeSelector`	Pod的其他节点选择器。请参见 "了解控制器Pod和节点Pod" 了解详细信息。	
`tridentNodePluginTolerations`	覆盖Kubernetes对Pod的容错。请参见 "了解控制器Pod和节点Pod" 了解详细信息。	

选项	Description	Default
imageRegistry	标识、 trident`和其它图像的注册表 `trident-operator。留空以接受默认值。重要信息：在私有存储库中安装Trident时、如果imageRegistry`使用交换机指定存储库位置、请勿在存储库路径中使用 `/netapp/。	""
imagePullPolicy	设置的映像提取策略 trident-operator。	IfNotPresent
imagePullSecs	设置的映像提取密钥 trident-operator， `trident`和其他图像。	
kubeletDir	允许覆盖kubelet内部状态的主机位置。	"/var/lib/kubelet"
operatorLogLevel	允许将Trident操作符的日志级别设置为： trace， debug， info， warn， error`或 `fatal。	"info"
operatorDebug	允许将Trident操作符的日志级别设置为DEBUG。	true
operatorImage	允许完全覆盖的映像 trident-operator。	""
operatorImageTag	允许覆盖的标记 trident-operator 图像。	""
tridentIPv6	允许在IPv6集群中启用Trident。	false
tridentK8sTimeout	覆盖大多数 Kubernetes API 操作的默认 180 秒超时（如果非零，则以秒为单位）。 这 `tridentK8sTimeout` 参数仅适用于Trident安装。	180
tridentHttpRequestTimeout	使用覆盖HTTP请求的默认90秒超时 0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutosupport	允许禁用Trident定期AutoSupport报告。	false
tridentExcludeAutosupport	当设置为 `true`时，Trident 在安装期间不会设置 AutoSupport 容器。	false
tridentAutosupportImageTag	允许覆盖Trident AutoSupport容器的映像标记。	<version>
tridentAutosupportProxy	允许Trident AutoSupport容器通过HTTP代理回拨。	""

选项	Description	Default
tridentLogFormat	设置Trident日志记录格式(text`或`json)。	"text"
tridentDisableAuditLog	禁用Trident审核日志程序。	true
tridentLogLevel	允许将Trident的日志级别设置为：trace、 debug info、、、 warn error`或 `fatal。	"info"
tridentDebug	允许将Trident的日志级别设置为debug。	false
tridentLogWorkflows	允许为跟踪日志记录或日志抑制启用特定的Trident工作流。	""
tridentLogLayers	允许为跟踪日志记录或日志抑制启用特定的Trident层。	""
TridentImage	允许完全覆盖Trident的图像。	""
tridentImageTag	允许覆盖Trident的映像标记。	""
tridentProbePort	允许覆盖用于Kubernetes活动/就绪性探测的默认端口。	""
windows	使Trident能够安装在Windows工作节点上。	false
enableForceDetach	允许启用强制分离功能。您可以通过与节点健康检查 (NHC) 操作符集成来自动执行强制分离过程。有关信息，请参阅"使用Trident实现有状态应用程序的故障转移自动化"。	false
enableDataLIFRefresh	当 LIF 被添加、删除或分配新地址时，刷新已发布卷的 NVMe 数据 LIF。适用于 ontap-san（含 sanType： nvme）（在统一 ONTAP 上）以及 ASA r2 上的 NVMe/TCP。有关更多信息，请参见"刷新 NVMe 数据 LIF"。	false
excludePodSecurityPolicy	从创建过程中排除操作员POD安全策略。	false
nodePrep	使Trident能够使Kubbernetes集群的节点做好准备、以便使用指定的数据存储协议管理卷。目前， **`iscsi`** 是唯一支持的值。 从 OpenShift 4.19 开始，此功能支持的最低Trident版本为25.06.1。	

选项	Description	Default

`resources`	设置 Kubernetes 对Trident控制器、节点和操作员 pod 的资源限制和请求。您可以为每个容器和边车配置 CPU 和内存，以管理 Kubernetes 中的资源分配。
	
	有关配置资源请求和限制的更多信息，请参阅"Pod 和容器的资源管理"。
	
	• 请勿更改任何容器或字段的名称。
	• 请勿更改缩进 - YAML 缩进对于正确解析至关重要。
	
	• 默认情况下不应用任何限制——只有请求才有默认值。
	• 容器名称按其在 pod 规范中的显示方式列出。
	• 边车列在每个主集装箱下方。
	• 检查TORC的 `status.CurrentInstallationParams `用于查看当前应用值的字段。
		`resources:`
		`  controller:`
		`    trident-main:`
		`      requests:`
		`        cpu: 10m`
		`        memory: 80Mi`
		`      limits:`
		`          cpu:`
		`          memory:`
		`    csi-provisioner:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-attacher:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-resizer:`
		`      requests:`
		`        cpu: 3m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    csi-snapshotter:`
		`      requests:`
		`        cpu: 2m`
		`        memory: 20Mi`
		`      limits:`
		`        cpu:`
		`        memory:`
		`    trident-autosupport:`
		`      requests:`
		`        cpu: 1m`
		`        memory: 30Mi`
		`      limits:`
		`        cpu:`
		`        memory:`

自定义Trident操作员安装

Trident操作员允许您使用规范中的属性自定义Trident安装 `TridentOrchestrator`。如果要在允许的参数之外自定义安装 `TridentOrchestrator`、请考虑使用生成自定义YAML清单、 `tridentctl`以便根据需要进行修改。

了解控制器Pod和节点Pod

Trident在集群中的每个工作节点上以单个控制器 pod 和节点 pod 的形式运行。节点 pod 必须运行在您希望挂载Trident卷的任何主机上。

Kubernetes "节点选择器" 和 "容忍和损害" 用于限制Pod在特定节点或首选节点上运行。使用`ControllerPlugin`和 `NodePlugin`、您可以指定约束和覆盖。

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

配置选项

> `spec.namespace`在中指定 `TridentOrchestrator`以表示安装`Trident`的命名空间。安装`Trident`后无法更新此参数*。尝试执行此操作会导致 `TridentOrchestrator`状态更改为 `Failed`。Trident不应跨多个名称卷进行迁移。

此表详细介绍了相关信息 `TridentOrchestrator` 属性。

参数	Description	Default
`namespace`	要安装Trident的命名空间	`"default"`
`debug`	为Trident启用调试	`false`
`enableForceDetach`	`ontap-san`， `ontap-san-economy` ， `ontap-nas` ， 和 `ontap-nas-economy`仅有的。与 Kubernetes 非优雅节点关闭 (NGNS) 配合使用，使集群管理员能够在节点出现故障时安全地将挂载卷的工作负载迁移到新节点。有关信息，请参阅"使用Trident实现有状态应用程序的故障转移自动化"。	`false`
`enableDataLIFRefresh`	当添加、删除 LIF 或为 LIF 分配新地址时，支持刷新已发布卷的 NVMe 数据 LIF。适用于 `ontap-san`（具有 `sanType: nvme`）、统一 ONTAP 上的 NVMe/TCP 以及 ASA r2 上的 NVMe/TCP。有关详细信息，请参见"刷新 NVMe 数据 LIF"。	`false`
`windows`	设置为 `true` 启用在Windows工作节点上的安装。	`false`
`cloudProvider`	如果在AKS集群上使用托管身份或云身份、请设置为`"Azure"`。如果在EKS集群上使用云标识、请设置为`"AWS"`。在GKE集群上使用云标识时、将设置为`"GCP"`。	`""`

参数	Description	Default
cloudIdentity	在AKS集群上使用云标识时、设置为工作负载标识("azure.Workload .Identity /client-id：XXXXXX-xxxx-xxxx-xxxx-xxxxxxxxxxx")。在Trident集群上使用云身份时、设置为eks.amazonaws.com/role-arn: AWS IAM角色("eks arn：aws：iam：：：12356：roe/EAM-roe"")。在GKE集群上使用云标识时、请设置为云标识("iam.GKE.IO/GCP-service-account：xxxx@mygcpproject.iam.gserviceaccount.com ")。	""
gcpWIFAudience	在 OpenShift 上与 Google Cloud Workload Identity Federation 一起使用的投影服务帐户令牌的受众。使用提供商的规范资源名称。有关详细信息，请参见 "使用 Workload Identity Federation 配置 Google Cloud NetApp Volumes"。	""
IPv6	安装基于IPv6的Trident	false
k8sTimeout	Kubernetes 操作超时。 这 `k8sTimeout`参数仅适用于Trident安装。	180sec
silenceAutosupport	不要向NetApp发送AutoSupport捆绑包自动	false
autosupport 映像	AutoSupport 遥测的容器映像	"netapp/trident-autosupport:26.10"
autosupport 代理	用于发送AutoSupport的代理的地址/端口遥测	"http://proxy.example.com:8888"
卸载	用于卸载Trident的标志	false
logFormat	要使用的Trident日志记录格式[text,json]	"text"
TridentImage	要安装的Trident映像	"netapp/trident:26.10.1"
imageRegistry	内部注册表的路径、格式 <registry FQDN>[:port][/subpath]	"registry.k8s.io"
kubeletDir	主机上的 kubelet 目录的路径	"/var/lib/kubelet"
wipeout	要删除以执行Trident完全删除的资源列表	
imagePullSecs	从内部注册表中提取映像的机密信息	
imagePullPolicy	设置Trident运算符的映像提取策略。有效值为：Always 以始终提取映像。IfNotPresent 仅当节点上尚不存在映像时才提取该映像。Never 从不提取映像。	IfNotPresent
控制器插件节点选择器	Pod的其他节点选择器。 遵循与相同的格式 pod.spec.nodeSelector。	无默认值；可选

参数	Description	Default
控制器插件公差	覆盖Kubernetes对Pod的容错。遵循与相同的格式 `pod.spec.Tolerations`。	无默认值；可选
`nodePluginNodeSelector`	Pod的其他节点选择器。遵循与相同的格式 `pod.spec.nodeSelector`。	无默认值；可选
`nodePluginTholeations`	覆盖Kubernetes对Pod的容错。遵循与相同的格式 `pod.spec.Tolerations`。	无默认值；可选
`nodePrep`	使Trident能够使Kubbernetes集群的节点做好准备、以便使用指定的数据存储协议管理卷。目前， **`iscsi`**是唯一支持的值。 从 OpenShift 4.19 开始，此功能支持的最低Trident版本为 25.06.1。	
`k8sAPIQPS`	控制器与 Kubernetes API 服务器通信时使用的每秒查询数 (QPS) 限制。 Burst 值根据 QPS 值自动设置。	`100`; 选修的
`enableConcurrency`	启用并发 Trident 控制器操作，以提高吞吐量。有关支持的驱动程序的详细信息，请参见链接：https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability[Controller Scalability documentation]。	`enableCOSI`
启用 Trident 容器对象存储接口（COSI）前端。要求将 `enableConcurrency` 设置为 `true`。Trident COSI 不能与社区 COSI 控制器一起运行。有关详细信息，请参见 "使用 COSI 配置对象存储"。	`false`	`resources`

参数	Description	Default

设置 Kubernetes 对Trident控制器和节点 pod 的资源限制和请求。您可以为每个容器和边车配置 CPU 和内存，以管理 Kubernetes 中的资源分配。
 	
有关配置资源请求和限制的更多信息，请参阅"Pod 和容器的资源管理"。
 	
• 请勿更改任何容器或字段的名称。
• 请勿更改缩进 - YAML 缩进对于正确解析至关重要。
 	
• 默认情况下不应用任何限制——只有请求才有默认值，如果未指定则自动应用。
• 容器名称按其在 pod 规范中的显示方式列出。
• 边车列在每个主集装箱下方。
	`resources:`
	`  controller:`
	`  trident-main:`
	`  requests:`
	`  cpu: 10m`
	`  memory: 80Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-provisioner:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-attacher:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-resizer:`
	`  requests:`
	`  cpu: 3m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  csi-snapshotter:`
	`  requests:`
	`  cpu: 2m`
	`  memory: 20Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  trident-autosupport:`
	`  requests:`
	`  cpu: 1m`
	`  memory: 30Mi`
	`  limits:`
	`  cpu:`
	`  memory:`
	`  node:`
		`httpsMetrics`

参数	Description	Default
为 Prometheus 指标端点启用 HTTPS。	false	`hostNetwork`

有关格式化POD参数的详细信息、请参阅 "将 Pod 分配给节点"。

配置示例

您可以使用以下属性[配置选项]在定义时 `TridentOrchestrator`自定义您的安装。

▼ 基本自定义配置

这个示例是在运行以下命令后创建的： `cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml`该命令代表一个基本的自定义安装：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
```

▼ 节点选择器

此示例使用节点选择器安装Trident 。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  controllerPluginNodeSelector:
    nodetype: master
  nodePluginNodeSelector:
    storage: netapp
```

▼ **Windows** 工作节点

这个示例是在运行以下命令后创建的： `cat deploy/crds/tridentorchestrator_cr.yaml`该命令在 Windows 工作节点上安装Trident 。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true
```

▼ **AKS** 集群上的托管身份

本示例安装Trident以在 AKS 集群上启用托管身份。

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

▼ **AKS** 集群上的云身份

本示例在 AKS 集群上安装Trident以用于云身份。

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

▼ **EKS** 集群上的云身份

本示例在 AKS 集群上安装Trident以用于云身份。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
```

```
spec:
  debug: true
  namespace: trident
  cloudProvider: "AWS"
  cloudIdentity: "'eks.amazonaws.com/role-arn:
arn:aws:iam::123456:role/trident-role'"
```

▼ 适用于**GKE**的云身份

本示例在 GKE 集群上安装Trident以用于云身份。

```
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
```

▼ **Trident**控制器和**Trident Linux** 节点 **Pod** 的 **Kubernetes** 资源请求和限制配置

本示例为Trident控制器和Trident Linux 节点 pod 配置 Kubernetes 资源请求和限制。



免责声明：本示例中提供的请求和限制值仅用于演示目的。请根据您的环境和工作负载需求调整这些值。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
```

```
        limits:
          cpu: 200m
          memory: 256Mi
      # sidecars
      csi-provisioner:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-attacher:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-resizer:
        requests:
          cpu: 3m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-snapshotter:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      trident-autosupport:
        requests:
          cpu: 1m
          memory: 30Mi
        limits:
          cpu: 50m
          memory: 128Mi
    node:
      linux:
        trident-main:
          requests:
            cpu: 10m
            memory: 60Mi
          limits:
```

```
            cpu: 200m
            memory: 256Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 1m
            memory: 10Mi
          limits:
            cpu: 50m
            memory: 32Mi
```

▼ **Kubernetes资源请求和限制配置，适用于Trident控制器以及Trident Windows和Linux节点Pod**

本示例为Trident控制器和Trident Windows 和 Linux 节点 pod 配置 Kubernetes 资源请求和限制。



免责声明：本示例中提供的请求和限制值仅用于演示目的。请根据您的环境和工作负载需求调整这些值。

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullSecrets:
  - thisisasecret
  windows: true
  resources:
    controller:
      trident-main:
        requests:
          cpu: 10m
          memory: 80Mi
        limits:
          cpu: 200m
          memory: 256Mi
        # sidecars
      csi-provisioner:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-attacher:
```

```
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-resizer:
        requests:
          cpu: 3m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      csi-snapshotter:
        requests:
          cpu: 2m
          memory: 20Mi
        limits:
          cpu: 100m
          memory: 64Mi
      trident-autosupport:
        requests:
          cpu: 1m
          memory: 30Mi
        limits:
          cpu: 50m
          memory: 128Mi
    node:
      linux:
        trident-main:
          requests:
            cpu: 10m
            memory: 60Mi
          limits:
            cpu: 200m
            memory: 256Mi
        # sidecars
        node-driver-registrar:
          requests:
            cpu: 1m
            memory: 10Mi
          limits:
            cpu: 50m
            memory: 32Mi
      windows:
        trident-main:
```

```
      requests:
        cpu: 6m
        memory: 40Mi
      limits:
        cpu: 200m
        memory: 128Mi
    # sidecars
    node-driver-registrar:
      requests:
        cpu: 6m
        memory: 40Mi
      limits:
        cpu: 100m
        memory: 128Mi
    liveness-probe:
      requests:
        cpu: 2m
        memory: 40Mi
      limits:
        cpu: 50m
        memory: 64Mi
```

使用tridentctl进行安装

使用trdentcl安装Trident

您可以使用安装Trident `tridentctl`。此过程适用于Trident所需容器映像存储在或未存储在私有注册表中的安装。要自定义 `tridentctl`部署，请参见"自定义 tridentctl 部署"。

有关 Trident 26.10 的关键信息

您必须阅读以下有关Trident的重要信息。

▼ 与Trident 相关的信息

- Trident 26.10 支持 Kubernetes 1.37。在升级 Kubernetes 之前升级 Trident。
- Trident会严格强制在SAN环境中使用多路径配置、并在Multipath.conf文件中使用建议值 `find_multipaths: no`。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

使用安装Trident `tridentctl`

请检查 "安装概述" 以确保您已满足安装先决条件，并为您的环境选择了正确的安装选项。

开始之前

在开始安装之前，请登录到 Linux 主机并验证其是否正在管理工作，"支持的 Kubernetes 集群"以及您是否具有必要的权限。

使用 OpenShift 时，在以下所有示例中使用 `oc` 而不是 `kubectl` ，并首先以 * 系统： admin* 的身份运行 `oc login -u system ： admin` 或 `oc login -u Kube-admin` 进行登录。

▼ **Details**

1. 验证Kubernetes版本：

```
kubectl version
```

2. 验证集群管理员权限：

```
kubectl auth can-i '*' '*' --all-namespaces
```

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
  ping <management IP>
```

第**1**步：下载**Trident**安装程序包

Trident安装程序包会创建Trident Pod、配置用于保持其状态的CRD对象、并初始化CSI边形以执行配置卷并将卷附加到集群主机等操作。从下载并解压缩最新版本的Trident安装程序"GitHub上的_assets_部分"。使用您选择的版本更新Trident本示例中的Trident安装程序-XX．．．．．．tar．gz>_。

```
wget https://github.com/NetApp/trident/releases/download/v26.10.1/trident-
installer-26.10.1.tar.gz
tar -xf trident-installer-26.10.1.tar.gz
cd trident-installer
```

第**2**步：安装**Trident**

执行命令、在所需的命名空间中安装Trident `tridentctl install`。您可以添加其他参数来指定映像注册表位置。

标准模式

```
./tridentctl install -n trident
```

一个注册表中的映像

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:26.10 --trident
-image <your-registry>/trident:26.10.1
```

不同注册表中的映像

```
./tridentctl install -n trident --image-registry <csi-registry>
--autosupport-image <autosupport-registry>/trident-autosupport:26.10
--trident-image <trident-registry>/trident:26.10.1
```

您的安装状态应如下所示。

```
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=26.10.1
INFO Trident installation succeeded.
....
```

验证安装。

您可以使用POD创建状态或验证安装 `tridentctl`。

正在使用**POD**创建状态

您可以通过查看已创建Pod的状态来确认Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-679648bd45-cv2mx   6/6     Running   0          5m29s
```

```
trident-node-linux-vgc8n              2/2     Running   0          5m29s
```

如果安装程序未成功完成或 `trident-controller-<generated id>`未处于 **Running** 状态，则平台未安装。使用 `-d`来"打开调试模式"并对问题进行故障排除。

使用 **`tridentctl`**

您可以使用 `tridentctl`检查安装的Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 26.10.1        | 26.10.1        |
+----------------+----------------+
```

配置示例

以下示例提供了使用安装Trident的示例配置 `tridentctl`。

▼ **Windows**节点

要使Trident能够在Windows节点上运行、请执行以下操作：

```
tridentctl install --windows -n trident
```

▼ 强制断开

有关信息，请参阅"使用Trident实现有状态应用程序的故障转移自动化"。

```
tridentctl install --enable-force-detach=true -n trident
```

▼ 启用并发**Trident**控制器操作

要启用并发 Trident 控制器操作以提高吞吐量，请在安装过程中添加 `--enable-concurrency` 选项，如本示例所示。有关支持的驱动程序的更多信息，请参见链接：https://docs.netapp.com/us-en/trident/trident-use/controller-scalability.html#controller-scalability[Controller Scalability documentation]。

```
tridentctl install --enable-concurrency -n trident
```

▼ 刷新 **NVMe** 数据 **LIF**

要在 LIF 配置更改时刷新已发布卷的 NVMe 数据 LIF，请使用 `--enable-datalif-refresh`安装 Trident。有关支持的驱动程序和配置详细信息，请参见"刷新 NVMe 数据 LIF"。

```
tridentctl install --enable-datalif-refresh -n trident
```

自定义 **Trident tridentctl** 安装

您可以使用Trident安装程序自定义安装。

了解安装程序

通过Trident安装程序、您可以自定义属性。例如，如果已将Trident映像复制到私有存储库，则可以使用指定映像名称 `--trident-image`。如果您已将Trident映像以及所需的CSI副本映像复制到私有存储库，则最好使用开关指定该存储库的位置， `--image-registry`其格式为 `<registry FQDN>[:port]`。

> 在私有存储库中安装Trident时、如果 `--image-registry`使用交换机指定存储库位置、请勿在存储库路径中使用 `/netapp/`。例如： `./tridentctl install --image-registry <image-registry> -n <namespace>`

如果您使用的是 Kubernetes 的分发版本，其中 `kubelet` 将其数据保存在通常的 `/var/lib/kubelet` 以外的路径上，则可以使用 `-kubelet-dir` 指定备用路径。

如果您需要自定义安装，使其超出安装程序参数的允许范围，则还可以自定义部署文件。使用 `-generate-custom-yaml` 参数可在安装程序的 `setup` 目录中创建以下 YAML 文件：

- `trident — clusterrolebinding 。 yaml`
- `trident 部署。 yaml`
- `trident — crds.yaml`
- `trident 集群角色。 yaml`
- `trident — demonset.yaml`
- `trident service.yaml`
- `trident 命名空间 .yaml`
- `trident 服务帐户。 yaml`
- `trident-resourcequota.yaml*`

生成这些文件后，您可以根据需要对其进行修改，然后使用 `-use-custom-yaml` 安装自定义部署。

```
./tridentctl install -n trident --use-custom-yaml
```

使用**OpenShift**认证操作员进行安装

使用**OpenShift OperatorHub**安装**Trident**

如果您使用的是Red Hat OpenShift、则可以通过Red Hat认证操作员安装NetApp

Trident。使用此过程从Red Hat OpenShift容器平台安装Trident。

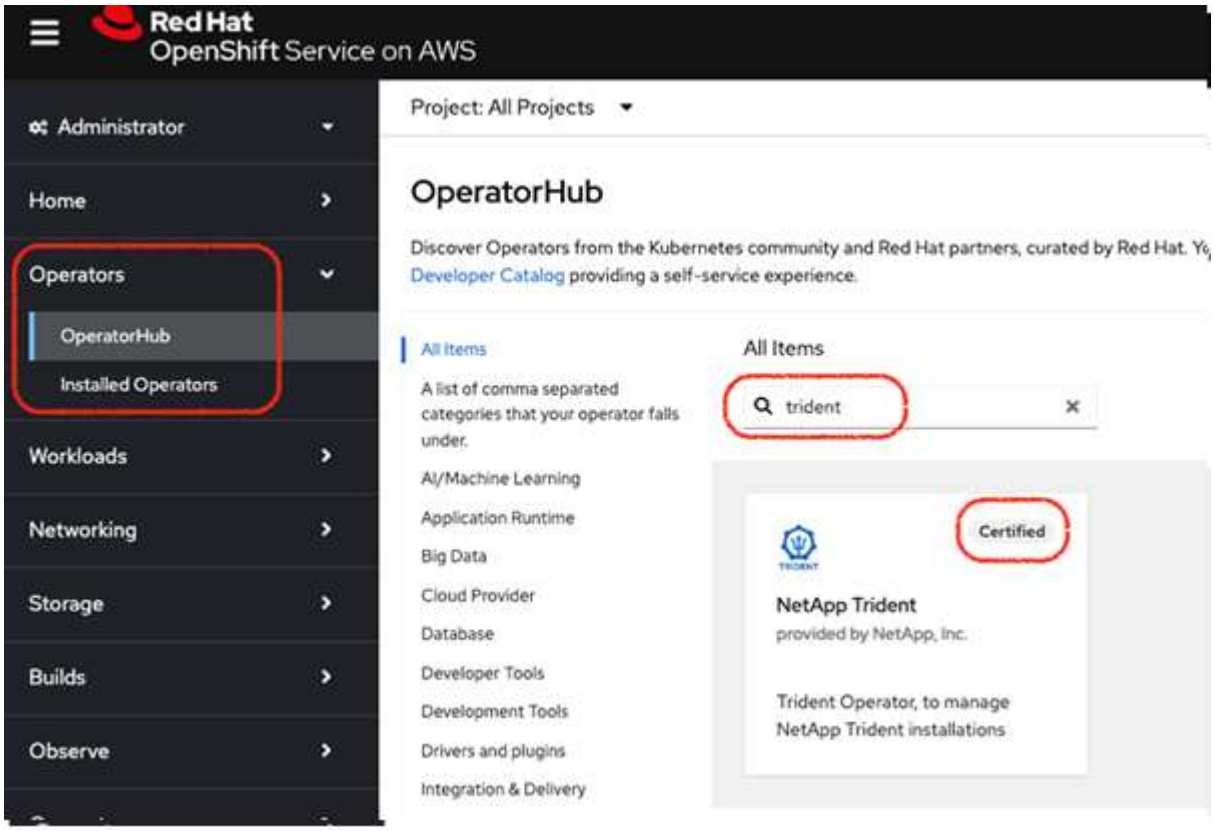
开始之前

开始安装之前，"准备安装Trident的环境"。

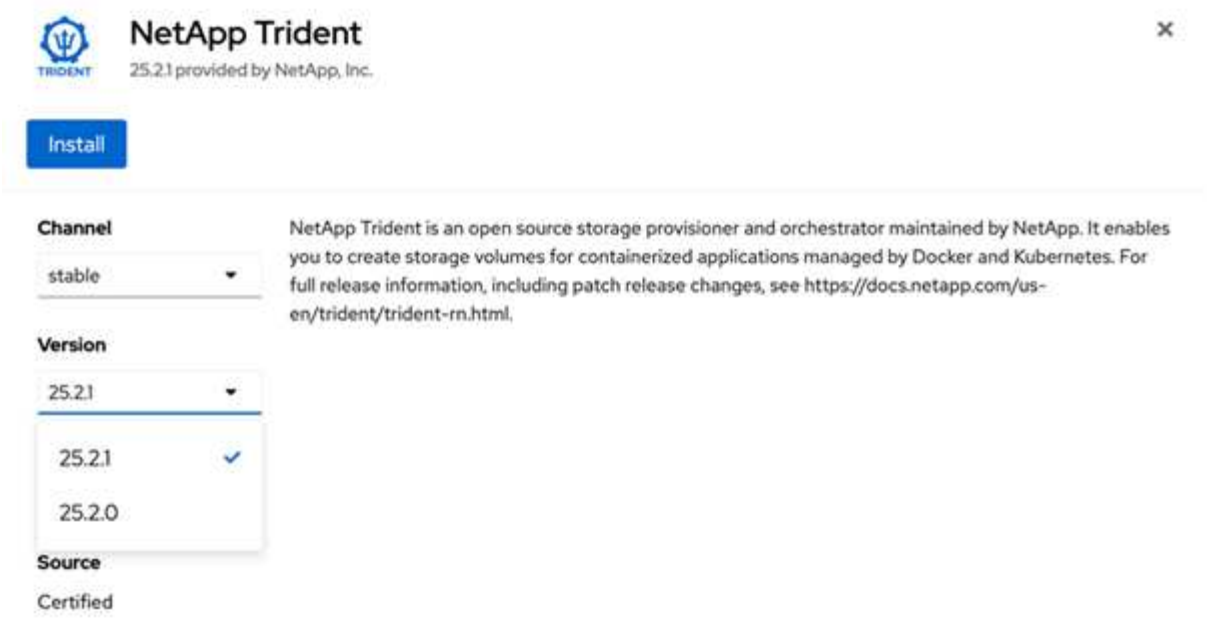
找到并安装**Trident**运算符

步骤

1. 导航到OpenShift OperatorHub并搜索NetApp Trident。

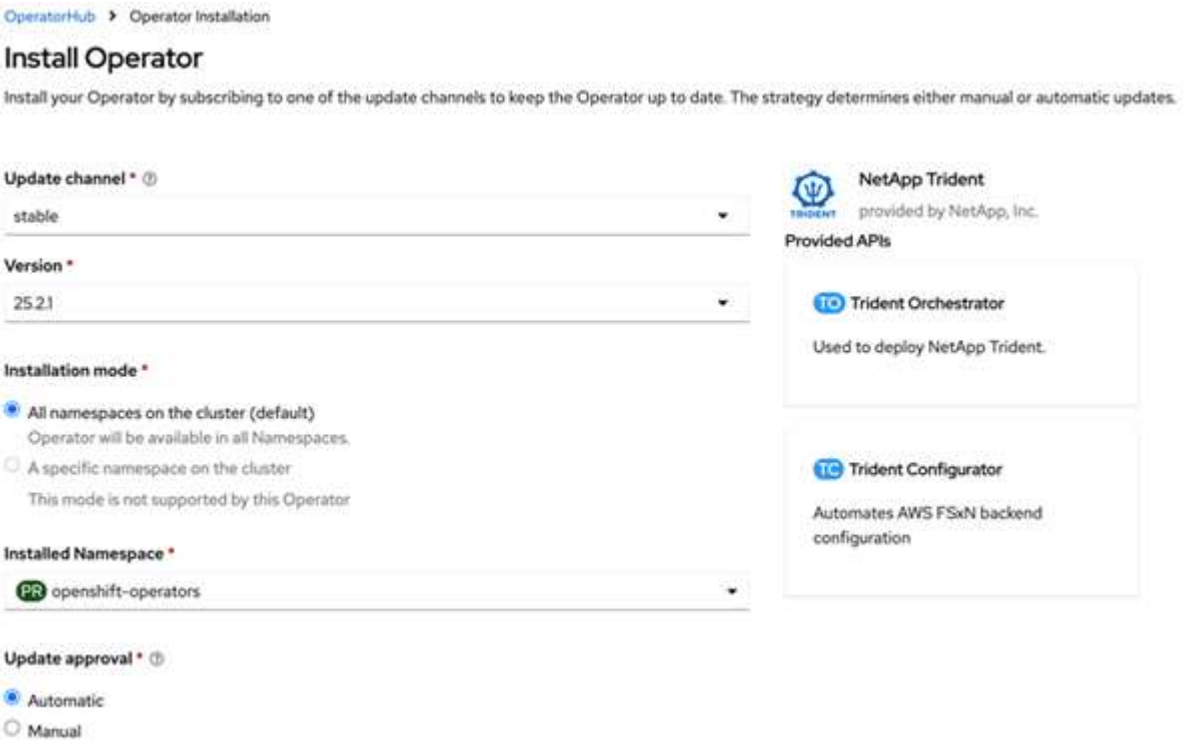


2. 单击* NetApp Trident* 打开安装设置。
3. 选择所需的选项，然后单击“安装”以打开操作员配置。

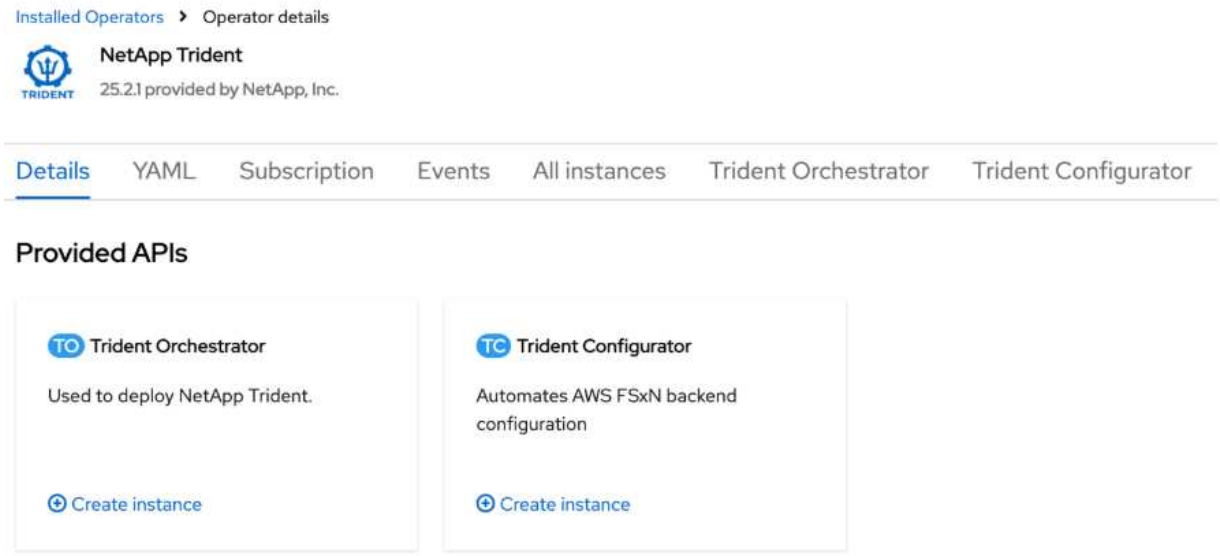


确保选择最新的 Operator 版本。

4. 保留所有参数，然后单击“安装”。



5. 单击“查看操作员”可查看操作员的详细信息。



6. 单击“YAML 视图”并将以下内容粘贴到表单中：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
  namespace: openshift-operators
spec:
  IPv6: false
  debug: false
  nodePrep:
  - iscsi
  imageRegistry: ''
  k8sTimeout: 180s
  namespace: trident
  silenceAutosupport: false
```

UI 提供了默认示例。您可以直接编辑它，而不是复制完整配置。

可选：启用并发

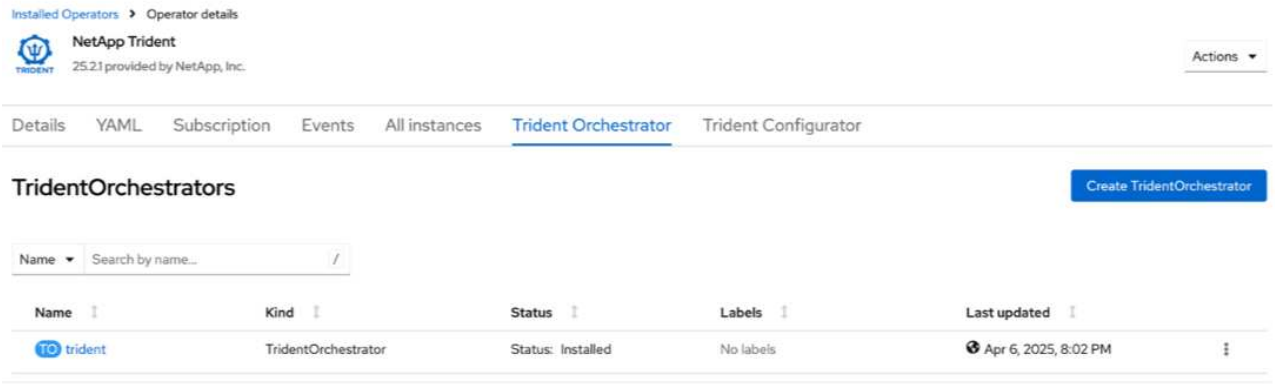
要启用并发，请将以下字段添加到规范：

```
enableConcurrency: true
```

- Red Hat Enterprise Linux CoreOS (RHCOS) 未启用和配置 iSCSI。
- 您可以添加 `nodePrep`在所有 OpenShift 工作节点上配置和启用 iSCSI 和 Multipath 服务的参数。

- 从 OpenShift 4.19 开始，此功能支持的最低Trident版本为 25.06.1。

1. 单击“创建”； Trident Orchestrator 将完全安装。



卸载**Trident**操作员

步骤

1. 从已安装操作员列表中选择Trident操作员。
2. 选择是否要从操作符中删除所有操作数实例。

> 如果未选中“从该运算符中删除所有操作数实例”复选框，则不会卸载Trident。

3. 单击 * 卸载 * 。

切换到 **OpenShift** 认证的 **Trident** 操作员

您可以从社区操作员、基于 Helm 的安装或手动部署的操作员切换到 Red Hat OpenShift 认证的 Trident 操作员。每种方法的过程包括卸载现有操作员，然后使用 OperatorHub 安装认证操作员。

开始之前

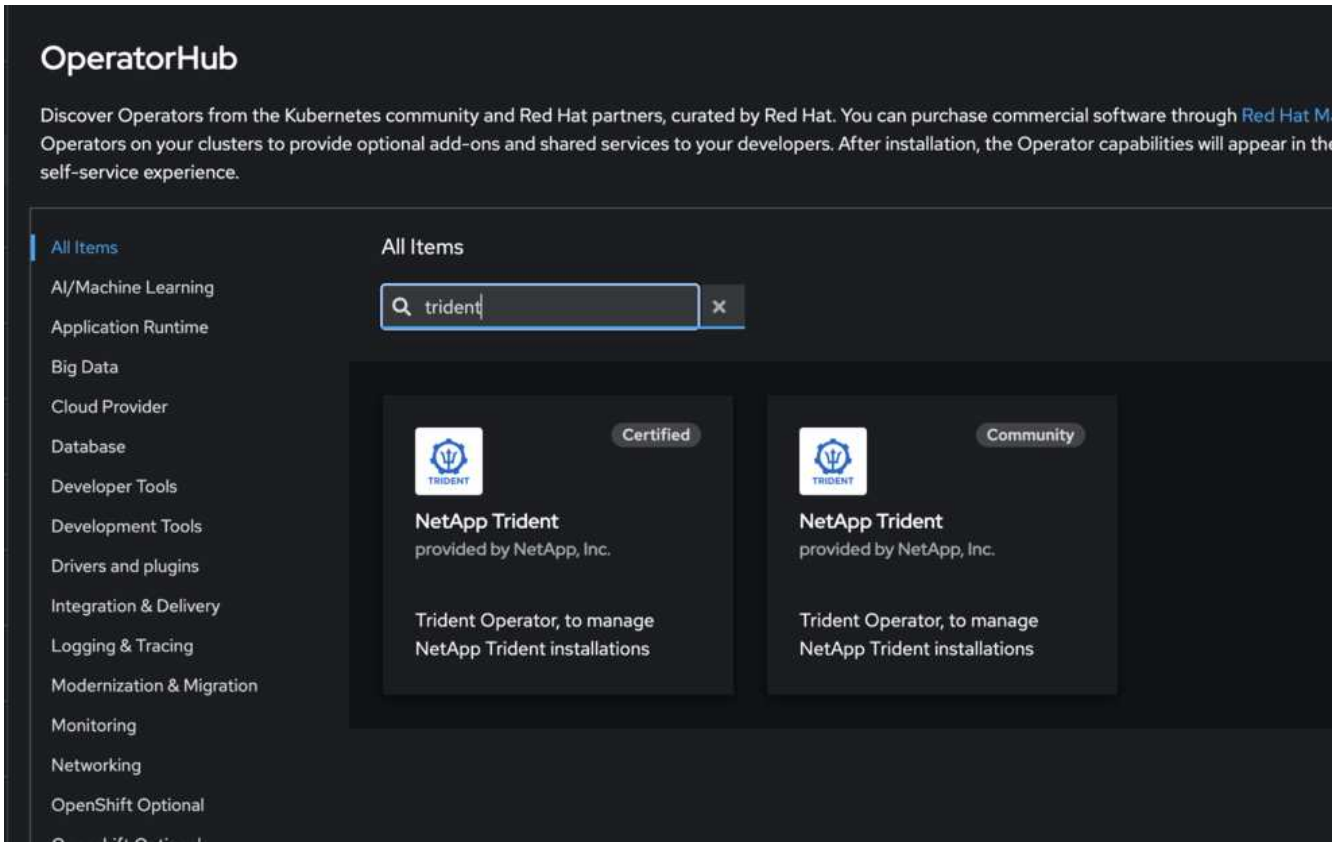
开始安装之前，"准备安装Trident的环境"。

> 请勿在卸载过程中删除 `TridentOrchestrator`自定义资源 (CR)。`TridentOrchestrator` CR 保留您的后端和存储类配置，这是在安装认证操作员后所需的配置。

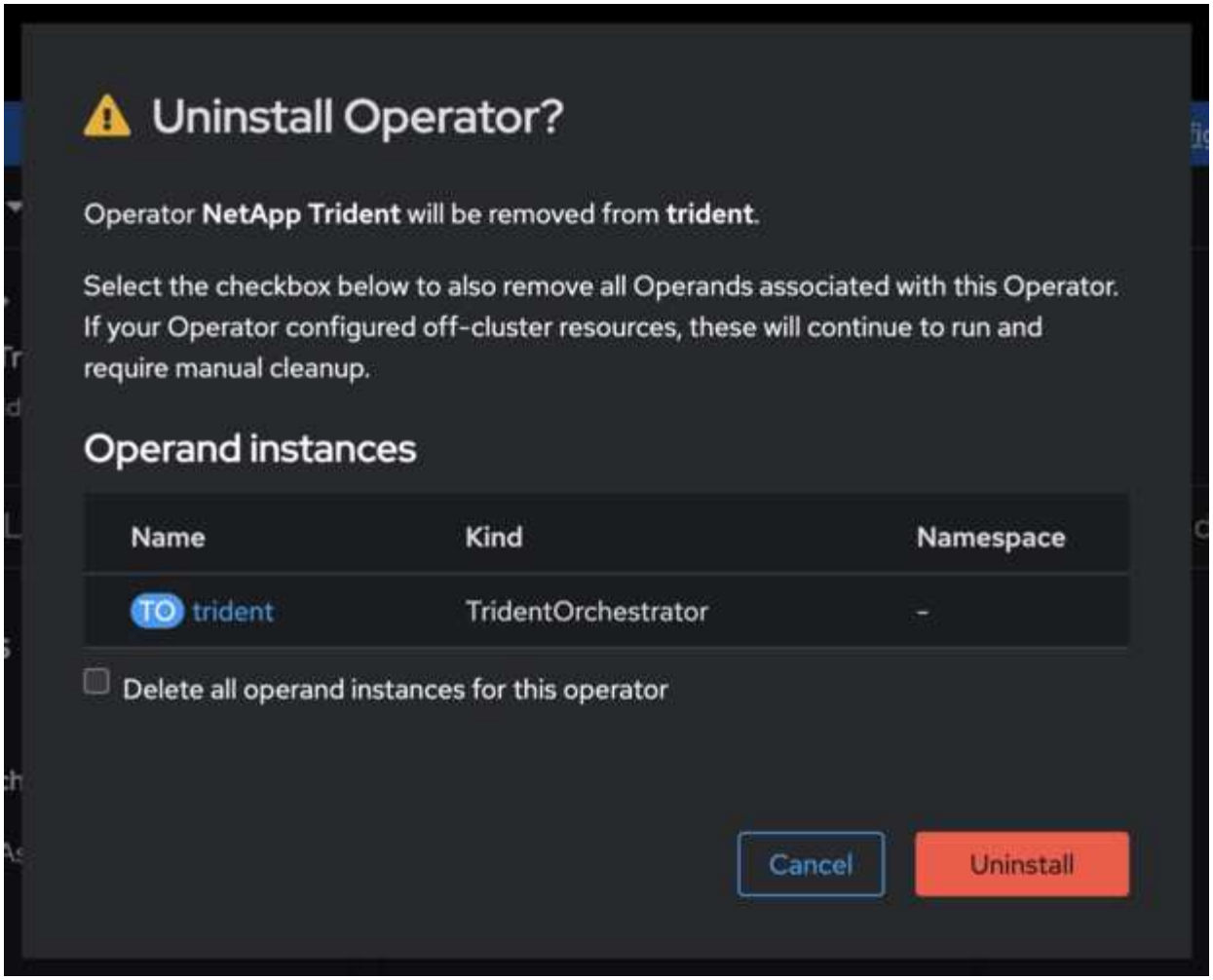
从社区运营商切换

步骤

1. 使用OpenShift控制台导航到OperatorHub。



2. 查找NetApp Trident社区操作员。



不要选择*从该操作符中删除所有操作数实例*。

3. 单击 * 卸载 * 。
4. 卸载完成后，请继续 安装 OpenShift 认证操作员。

从基于 **Helm** 的操作员安装切换

步骤

1. 列出 Trident 安装的 Helm 版本：

```
helm ls -n trident
```

2. 卸载 Helm 版本：

```
helm uninstall <release-name> -n trident
```

3. 卸载完成后，请继续 安装 OpenShift 认证操作员。

从手动部署的 operator 切换

如果您通过使用安装程序包中的 `bundle.yaml` 手动部署操作员来安装 Trident，请通过删除相同的清单来删除它。

步骤

1. 使用捆绑清单删除操作员部署：

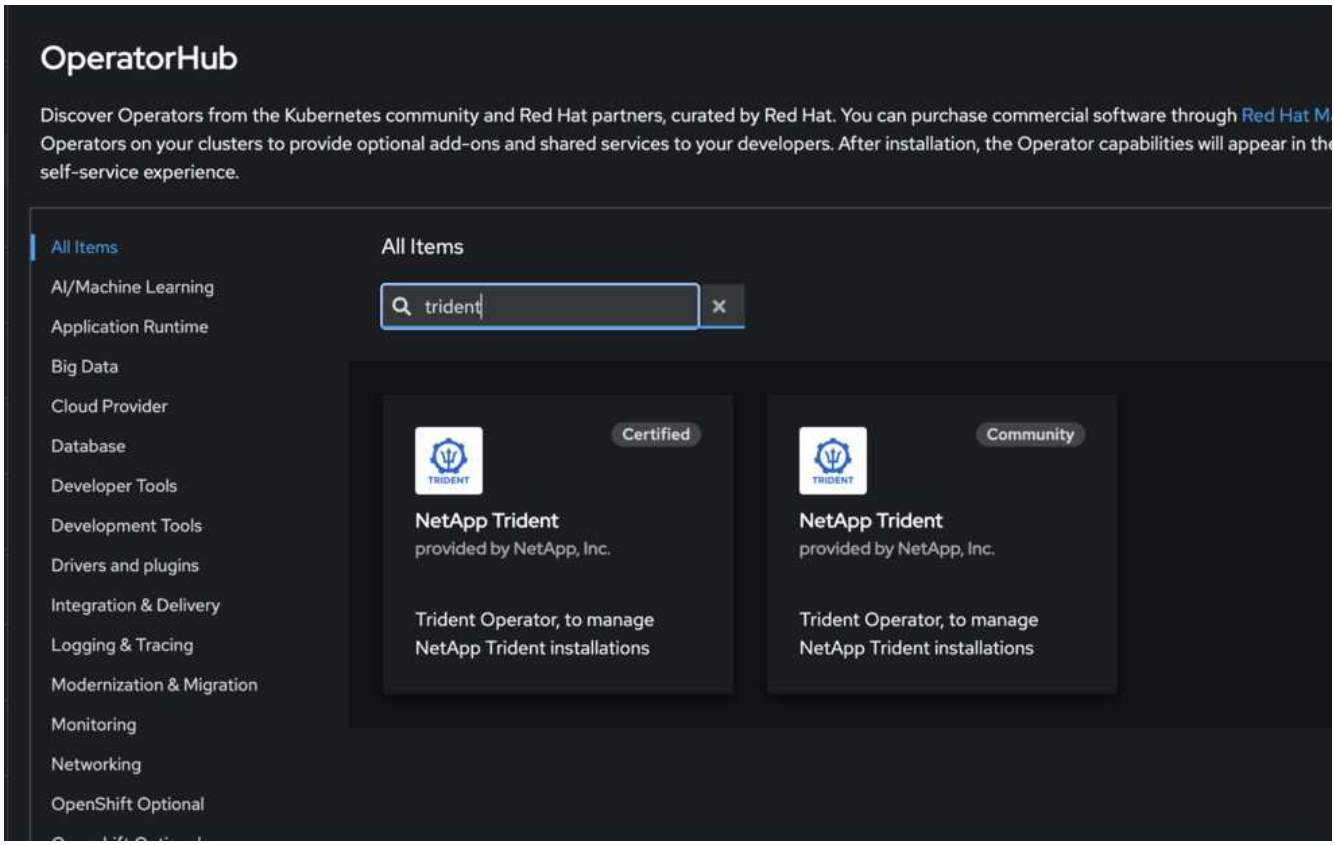
```
kubectl delete -f deploy/bundle.yaml -n trident
```

2. 卸载完成后，请继续 安装 OpenShift 认证操作员。

安装 OpenShift 认证操作员

步骤

1. 导航到Red Hat OperatorHub。
2. 搜索并选择NetApp Trident操作员。



3. 按照屏幕上的说明安装操作员。

验证

- 检查控制台中的OperatorHub、确保已成功安装新的认证操作员。

使用Trident

准备 Trident Worker 节点

Kubernetes集群中的所有工作节点都必须能够挂载为Pod配置的卷。要准备工作节点、必须根据您选择的驱动程序安装NFS、iSCSI、NVMe/TCP或FC工具。

选择合适的工具

如果您使用的是驱动程序组合、则应安装驱动程序所需的所有工具。默认情况下、最新版本的Red Hat Enterprise Linux CoreTM OS (RHCOS)已安装这些工具。

NFS工具

"安装NFS工具"如果您正在使用： `ontap-nas` ， `ontap-nas-economy` ， `ontap-nas-flexgroup` ， 或者 `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" 如果您使用的是 `ontap-san` 适用于基于TCP (NVMe/TCP)协议的非易失性内存标准(NVMe)。

NetApp建议对NVMe/TCP使用ONTAP 9.12或更高版本。

基于**FC**的**SCSI**工具

请参见"配置FC和FC-NVMe SAN主机的方式()"有关配置FC和FC-NVMe SAN主机的详细信息、。

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

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

SMB工具

"准备配置SMB卷" 如果您正在使用： `ontap-nas` 为 SMB 提供卷。

节点服务发现

Trident会尝试自动检测节点是否可以运行iSCSI或NFS服务。

节点服务发现可识别已发现的服务、但无法保证服务已正确配置。相反、如果没有发现的服务、则无法保证卷挂载将失败。

查看事件

Trident会为此节点创建事件以确定发现的服务。要查看这些事件、请运行：

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

查看发现的服务

Trident标识为Trident节点CR上的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 volumes

使用适用于您的操作系统的命令安装NFS工具。确保NFS服务已在启动期间启动。

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、以防止在将卷连接到容器时失败。

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、以便：

1. *确定*所需的iSCSI会话状态和当前的iSCSI会话状态。
2. 将所需状态与当前状态进行比较、以确定所需的修复。Trident确定修复优先级以及何时抢占修复。
3. 需要执行*修复*才能将当前iSCSI会话状态恢复为所需的iSCSI会话状态。

自我修复活动日志位于相应的Demonset Pod上的容器中 `trident-main`。要查看日志、必须在Trident安装期间将设置 `debug`为"TRUE"。

Trident iSCSI自我修复功能有助于防止：

- 在网络连接问题描述 之后可能发生的陈旧或运行不正常的iSCSI会话。如果会话陈旧、Trident将等待七分钟后再注销、以便重新建立与门户的连接。

例如、如果在存储控制器上轮换了CHAP密钥、而网络断开了连接、则旧的(*stal*) CHAP密钥可能会持续存在。自修复功能可以识别此问题、并自动重新建立会话以应用更新后的CHAP密码。

- 缺少iSCSI会话
- 缺少LUN

升级**Trident**前的注意事项

- 如果仅使用了每个节点的igroup (在23.04及更高版本中推出)、则iSCSI自我修复功能将对SCSI总线中的所有设备启动SCSI重新检查。
- 如果仅使用后端范围的igroup (自23.04起已弃用)、则iSCSI自行恢复功能将启动SCSI重新检查、以确定SCSI总线中的确切LUN ID。
- 如果混合使用了每个节点的igroub和后端范围的igroux、则iSCSI自我修复功能将对SCSI总线中的确切LUN ID启动SCSI重新检查。

安装**iSCSI**工具

使用适用于您的操作系统的命令安装iSCSI工具。

开始之前

- Kubernetes 集群中的每个节点都必须具有唯一的 IQN 。* 这是必要的前提条件 * 。
- 如果使用RHCOS 4.5或更高版本或其他与RHEL兼容的Linux分发版、请与结合使用 `solidfire-san` 驱动程序和Element OS 12：5或更早版本、请确保中的CHAP身份验证算法设置为MD5 `/etc/iscsi/iscsid.conf`。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
```

- 如果将运行RHL/Red Hat Enterprise Linux Core-OS (RHOS)的工作节点与iSCSI PVs结合使用、请在StorageClass中指定 `discard`mountOption以执行实时空间回收。请参阅 "Red Hat 文档"。
- 确保您已升级到最新版本 `multipath-tools` 。

RHEL 8+

1. 安装以下系统软件包：

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

2. 检查 iscsi-initiator-utils 版本是否为 6.2.0.877-2.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
```

确保 `/etc/multipath.conf`包含 `find_multipaths no`在下 `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 2.0.877-5ubuntu2.10 或更高版本（对于双子系统）或 2.0.877-7.1ubuntu6.1 或更高版本（对于 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
```

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

5. 确保已启用并运行 `open-iscsi` 和 `multipath-tools`：

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

对于 Ubuntu 18.04 ，您必须先使用 `iscsiadm` 发现目标端口，然后再启动 `open-iscsi` ， iSCSI 守护进程才能启动。您也可以将 `iscsi` 服务修改为自动启动 `iscsid` 。

配置或禁用**iSCSI**自我修复

您可以配置以下Trident iSCSI自我修复设置来修复陈旧会话：

- **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)。您可以将其配置为通过设置较小的数字来提高运行频率、也可以通过设置较大的数字来降低运行频率。

将iSCSI自我修复间隔设置为0可完全停止iSCSI自我修复。建议不要禁用iSCSI自我修复；只有在iSCSI自我修复功能无法正常工作或出于调试目的时、才应禁用它。

- **iSCSI**自我修复等待时间：确定在注销运行状况不正常的会话并尝试重新登录之前iSCSI自我修复等待的时间(默认值：7分钟)。您可以将其配置为较大的数字、以便确定为运行状况不正常的会话必须等待较长的时间才能注销、然后再尝试重新登录、或者配置为较小的数字以较早地注销和登录。

掌舵

要配置或更改iSCSI自我修复设置、请传递 `iscsiSelfHealingInterval` 和 `iscsiSelfHealingWaitTime` 舵安装或舵更新期间的参数。

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

```
helm install trident trident-operator-100.2610.0.tgz --set
iscsiSelfHealingInterval=3m0s --set iscsiSelfHealingWaitTime=6m0s -n
trident
```

Tridentctl

要配置或更改iSCSI自我修复设置、请传递 `iscsi-self-healing-interval` 和 `iscsi-self-healing-wait-time` 在安装或更新TRDentcdr期间的参数。

以下示例将iSCSI自我修复间隔设置为3分钟、并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

使用适用于您的操作系统的命令安装NVMe工具。

- NVMe需要RHEL 9或更高版本。
- 如果Kubelnetes节点的内核版本太旧、或者NVMe软件包不适用于您的内核版本、您可能需要将节点的内核版本更新为具有NVMe软件包的版本。

RHEL 9

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

Ubuntu

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

验证安装

安装后、使用命令验证Kubenetes集群中的每个节点是否都具有唯一的NQN：

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

Trident会修改此 `ctrl_device_tmo`值、以确保NVMe在路径发生故障时不会放弃此路径。请勿更改此设置。

基于FC卷的SCSI

现在、您可以在Trident中使用光纤通道(Fibre Channel、FC)协议来配置和管理ONTAP系统上的存储资源。

前提条件

为FC配置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. 获取启动程序(主机)上接口的WWPN。

   请参阅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、请参见以下ONTAP文档：

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

安装**FC**工具

使用适用于您的操作系统的命令安装FC工具。

- 如果将运行RHL/Red Hat Enterprise Linux Core-OS (RHOS)的工作节点与FC PVs结合使用、请在StorageClass中指定 `discard`mountOption以执行实时空间回收。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
```

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

3. 确保 `multipathd`正在运行：

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

Ubuntu

1. 安装以下系统软件包：

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

2. 启用多路径：

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

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

3. 确保 `multipath-tools`已启用且正在运行：

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

准备配置SMB卷

您可以使用以下方式配置 SMB 卷 `ontap-nas` 司机。

您必须在SVM上同时配置NFS和SMB/CCIFS协议、才能为ONTAP内部集群创建 `ontap-nas-economy`SMB卷。如果未能配置其中任一协议、则发生原因 SMB卷创建将失败。

`autoExportPolicy`SMB卷不支持。

开始之前

在配置SMB卷之前、您必须满足以下条件。

- 一个Kubernetes集群、其中包含一个Linux控制器节点以及至少一个运行Windows Server 2022的Windows工作节点。Trident仅支持挂载到Windows节点上运行的Pod的SMB卷。
- 至少一个包含Active Directory凭据的Trident密钥。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。配置 `csi-proxy`、请参见 "GitHub：CSI代理" 或 "GitHub：适用于Windows的CSI代理" 适用于在Windows上运行的Kubernetes节点。

步骤

1. 对于内部ONTAP、您可以选择创建SMB共享、也可以选择Trident为您创建一个共享。

   Amazon FSx for ONTAP需要SMB共享。

   您可以使用以下两种方式之一创建SMB管理共享 "Microsoft管理控制台" 共享文件夹管理单元或使用ONTAP命令行界面。要使用ONTAP 命令行界面创建SMB共享、请执行以下操作：

   a. 如有必要，为共享创建目录路径结构。

      。 `vserver cifs share create` 命令会在创建共享期间检查-path选项中指定的路径。如果指定路径不存在，则命令将失败。

   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. 验证是否已创建共享：

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

      请参见 "创建 SMB 共享" 了解完整详细信息。

2. 创建后端时、必须配置以下内容以指定SMB卷。有关适用于ONTAP 后端的所有FSX配置选项、请参见 "适用于ONTAP 的FSX配置选项和示例"。

参数	Description	示例
`smbShare`	您可以指定以下选项之一：使用Microsoft管理控制台或ONTAP命令行界面创建的SMB共享的名称；允许Trident创建SMB共享的名称；或者、您可以将参数留空以防止对卷进行通用共享访问。对于内部ONTAP、此参数是可选的。此参数对于Amazon FSx for ONTAP后端为必填项、不能为空。	`smb-share`
`nasType`	*必须设置为 `smb`如果为空、则默认为 `nfs`。	`smb`
`securityStyle`	新卷的安全模式。必须设置为 **`ntfs`** 或 **`mixed`** 用于**SMB**卷。	`ntfs` 或 `mixed` 对于SMB卷
`unixPermissions`	新卷的模式。对于SMB卷、必须留空。	""

使用 COSI 配置对象存储

使用容器对象存储接口（COSI）通过 Trident 管理的 Kubernetes 资源为 Kubernetes 工作负载配置 S3 存储桶。

概述

COSI 可将对象存储提供商连接到 Kubernetes 工作负载。Trident 可提供以下资源：

- `TridentObjectStore`：将 Trident 连接到对象存储系统及其凭据。在 Trident 命名空间中创建此资源。
- `BucketClass`：定义对象存储选择和存储桶删除策略。
- `BucketClaim`：请求一个存储桶。
- `BucketAccessClass`：定义如何配置对存储桶的访问。
- `BucketAccess`：请求一个或多个存储桶的凭据。

Trident 通过以下对象存储驱动程序支持 COSI：

- StorageGRID：全面可用。
- ONTAP S3：技术预览。

Trident 支持 S3 协议。在初始版本中，仅支持路径样式 S3 寻址。

要求和限制

- 在 `TridentOrchestrator`自定义资源 （`TORC`） 中同时启用 `enableConcurrency`和 `enableCOSI`。默认情况下，COSI 未启用。
- 在启用控制器并发之前，请验证所有配置的 Trident 后端是否使用受支持的驱动程序。不支持的后端驱动程序会阻止 Trident 启动；请参见 "在您启用控制器可扩展性之前"。
- 请勿将 Trident COSI 与社区 COSI 控制器一起运行。
- 初始版本使用 `objectstorage.k8s.io/v1alpha2` COSI API。
- StorageGRID `TridentObjectStore` 资源使用长期有效的租户根凭据。每个 `TridentObjectStore` 连接到一个 StorageGRID 租户；多个资源可以连接到同一个租户。将凭据存储在 Trident 命名空间的 Kubernetes Secret 中，并根据您的安全要求对其进行保护。
- 一个 StorageGRID `BucketAccess` 最多支持八个存储桶。

有关并发控制器操作的一般信息，请参见 "控制器可扩展性"。

启用 COSI

在 `TridentOrchestrator` 规范中设置这两个选项：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

```
  enableCOSI: true
```

使用常规操作员工作流程应用 TORC 或更新现有资源。有关配置 TORC 的详细信息，请参见"自定义Trident操作员安装"。

配置 StorageGRID 对象存储

在 Trident 命名空间中使用 StorageGRID 租户凭据创建 Kubernetes Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: tenant3
  namespace: trident
type: Opaque
stringData:
  username: <tenant-root-username>
  password: <tenant-root-password>
```

创建一个引用 Secret 的 `TridentObjectStore`。将示例端点和帐户 ID 替换为您 StorageGRID 环境的值：

```
apiVersion: trident.netapp.io/v1
kind: TridentObjectStore
metadata:
  name: trident-tenant3
  namespace: trident
spec:
  accountId: "<account-id>"
  credentials:
    secretName: tenant3
  labels:
    tenantInstance: three
  managementEndpoint: https://<storagegrid-management-endpoint>
  objectStoreDriver: storage-grid
  s3Endpoint: https://<storagegrid-s3-endpoint>
```

应用这两种资源：

```
kubectl apply -f tenant-secret.yaml
kubectl apply -f trident-object-store.yaml
```

配置 ONTAP S3 对象存储

通过 COSI 提供的 ONTAP S3 支持是 Trident 26.10 中的技术预览版，不支持在生产环境中使用。在正式发布之前，配置字段和行为可能会发生变化。

在 Trident 命名空间中使用 ONTAP S3 用户凭据创建 Kubernetes Secret，然后创建一个引用该 Secret 的 `TridentObjectStore`。将占位符替换为您 ONTAP S3 环境的值：

```
apiVersion: v1
kind: Secret
metadata:
  name: ontap-s3-credentials
  namespace: trident
type: Opaque
stringData:
  username: "<user>"
  password: "<password>"
---
apiVersion: trident.netapp.io/v1
kind: TridentObjectStore
metadata:
  name: ontap-s3
  namespace: trident
spec:
  objectStoreDriver: ontap-s3
  objectStoreName: ontap-object-store
  managementEndpoint: "<management-endpoint>"
  svmName: "<svm-name>"
  storagePrefix: trident
  credentials:
    secretName: ontap-s3-credentials
```

`storagePrefix` 是可选的，省略时默认为 `trident`。应用 `Secret` 和 `TridentObjectStore`：

```
kubectl apply -f ontap-s3-object-store.yaml
```

创建 BucketClass 并请求存储桶

`BucketClass`
可以使用标签选择器、对象存储类型或两者来选择对象存储。如果没有选择参数，`Trident`

可以从所有已配置的对象存储中进行选择。

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketClass
metadata:
  name: sg-delete
spec:
  driverName: cosi.trident.netapp.io
  deletionPolicy: Delete
  parameters:
    selector: tenantInstance=three
    objectStoreType: storage-grid
```

创建一个使用该类的 `BucketClaim`：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketClaim
metadata:
  name: claim-a1
  namespace: default
spec:
  bucketClassName: sg-delete
  protocols:
    - S3
```

当 Trident 处理声明时，它会选择一个匹配的对象存储。如果多个存储匹配，Trident 会随机选择一个，以便在匹配的存储之间分配存储桶。

授予对存储桶的工作负载访问权限

创建 `BucketAccessClass`以定义访问配置：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketAccessClass
metadata:
  name: sg-access
spec:
  authenticationType: Key
  driverName: cosi.trident.netapp.io
  multiBucketAccess: MultipleBuckets
```

创建一个引用存储桶声明的 `BucketAccess`：

```
apiVersion: objectstorage.k8s.io/v1alpha2
kind: BucketAccess
metadata:
  name: sg-claim-a1
  namespace: default
spec:
  bucketAccessClassName: sg-access
  bucketClaims:
    - accessMode: ReadWrite
      accessSecretName: sg-claim-a1
      bucketClaimName: claim-a1
  protocol: S3
```

Trident 创建一个包含存储桶访问信息的 Kubernetes Secret。工作负载可以使用标准的 Kubernetes Secret 机制来使用此 Secret。生成的 Secret 包括 S3 连接详细信息，例如存储桶标识符、端点、区域和访问密钥。请勿将生成的 Secret 暴露给未经授权的用户或工作负载。

生成的 Secret 包括诸如 `COSI_S3_ACCESS_KEY_ID`、 `COSI_S3_ACCESS_SECRET_KEY`、 `COSI_S3_ADDRESSING_STYLE`、 `COSI_S3_BUCKET_ID`、 `COSI_S3_ENDPOINT`和 `COSI_S3_REGION` 等键。

配置和管理后端

配置 Trident 后端

后端用于定义Trident与存储系统之间的关系。它会告诉Trident如何与该存储系统通信、以及Trident如何从该存储系统配置卷。

Trident会自动从满足存储类定义的要求的后端提供存储池。了解如何为存储系统配置后端。

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

Azure NetApp Files

配置 Azure NetApp Files 后端

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

支持的 **Azure** 云环境

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

支持的 Azure 云包括：

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

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

查看 **Azure NetApp Files** 驱动程序支持

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

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

驱动程序	协议	卷模式	支持的访问模式	支持的文件系统
`azure-netapp-files`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	`nfs`， `smb`

查看考虑事项

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

对 **AKS** 使用托管标识

Trident 支持 "受管身份" AKS 集群。

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

要使用托管身份，必须具有：

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

Trident 运算符

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

```
apiVersion: trident.netapp.io/v1
```

```
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
```

掌舵

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

```
helm install trident trident-operator-100.2610.0.tgz --create-namespace
--namespace <trident-namespace> --set cloudProvider=$CP
```

<code>tridentctl</code>

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

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

使用 **AKS** 的工作负载标识

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

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

要使用工作负载标识，必须具有：

- 使用AKS部署的Kubbernetes集群
- 在AKS Kubelnetes集群上配置的工作负载身份和oidc-Issuer
- 已安装 Trident，其中 `cloudProvider`设置为 `"Azure"`，`cloudIdentity`设置为工作负载身份值

Trident 运算符

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

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
```

```
  namespace: trident
  imagePullPolicy: IfNotPresent
  cloudProvider: "Azure"
  cloudIdentity: 'azure.workload.identity/client-id: xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxx' # Edit
```

掌舵

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

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

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

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

<code>tridentctl</code>

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

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

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

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

准备配置**Azure NetApp Files** 后端

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

支持的 **Azure** 云环境

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

支持的 Azure 云包括：

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

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

NFS和SMB卷的前提条件

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：



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

- 一个容量池。请参见 "Microsoft：为Azure NetApp Files 创建容量池"。
- 委派给Azure NetApp Files 的子网。请参见 "Microsoft：将子网委派给Azure NetApp Files"。
- `ssubscriptionID` 来自启用了 Azure NetApp Files 的 Azure 订阅。
- `tenantID`， `clientID`，和 `clientSecret` 从 "应用程序注册" 在 Azure Active Directory 中，具有足够的 Azure NetApp Files 服务权限。应用程序注册应使用以下任一项：
  - 所有者或贡献者角色 "由Azure预定义"。
  - "自定义贡献者角色"订阅级别(`assignableScopes`的)具有以下权限，这些权限仅限于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/re
ad",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/wr
ite",

"Microsoft.NetApp/netAppAccounts/capacityPools/volumes/snapshots/de
lete",

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

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/read",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/write",

"Microsoft.Features/featureProviders/subscriptionFeatureRegistratio
ns/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 "委派子网"的 Azure `location`。后端配置文件的顶层需要 `location`参数。在虚拟池中指定的位置值将被忽略。
- 以使用 `Cloud Identity`请获取 `client ID` 从A "用户分配的托管身份" 并在中指定此ID

azure.workload.identity/client-id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx。

SMB卷的其他要求

要创建SMB卷、您必须具有：

- 已配置Active Directory并连接到Azure NetApp Files。请参见 "Microsoft：创建和管理Azure NetApp Files的Active Directory连接"。
- 一个Kubernetes集群、其中包含一个Linux控制器节点以及至少一个运行Windows Server 2022的Windows工作节点。Trident仅支持挂载到Windows节点上运行的Pod的SMB卷。
- 至少一个包含Active Directory凭据的Trident密钥、以便Azure NetApp Files可以向Active Directory进行身份验证。生成密钥 `smbcreds`：

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

- 配置为Windows服务的CSI代理。配置 `csi-proxy`、请参见 "GitHub：CSI代理" 或 "GitHub：适用于Windows的CSI代理" 适用于在Windows上运行的Kubernetes节点。

Azure NetApp Files 后端配置选项和示例

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

后端配置选项

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后端提供了这些配置选项。

参数	Description	Default
ve版本	后端配置版本。	始终为 1
`storageDriverName`	存储驱动程序的名称	"Azure-netapp-files"
`backendName`	存储后端的自定义名称	驱动程序名称 + "_" + 随机字符
`ssubscriptionID`。	Azure 订阅中的订阅 ID 如果在AKS集群上启用了受管标识、则为可选。	
租户 `ID`。	应用程序注册中的租户 ID 如果在AKS集群上使用托管身份或云身份、则可选。	
客户端 `ID`。	应用程序注册中的客户端 ID 如果在AKS集群上使用托管身份或云身份、则可选。	

参数	Description	Default
clientSecret 。	应用程序注册中的客户端密钥 如果在AKS集群上使用托管身份或云身份、则可选。	
s服务级别	S标准 ， 高级 或 超级 之一	"" （随机）
位置	要创建新卷的 Azure 位置的名称 如果在AKS集群上启用了受管标识、则为可选。	
cloudConfiguration.cloudName	标识后端的 Azure 云环境。使用 Azure Government 时设置为 AzureGovernment。	
resourceGroups	用于筛选已发现资源的资源组列表	"[]" （无筛选器）
netappAccounts	用于筛选已发现资源的 NetApp 帐户列表	"[]" （无筛选器）
容量池	用于筛选已发现资源的容量池列表	"[]" （无筛选器，随机）
virtualNetwork	具有委派子网的虚拟网络的名称	""
ssubnet	委派给 Microsoft.Netapp/volumes 的子网的名称	""
网络功能	卷的 VNet 功能集，可以是 Basic` 或 `Standard。并非所有地区都提供 Network Features，可能必须在订阅中启用。在未启用此功能时指定 `networkFeatures`会导致卷配置失败。	""
nfsMountOptions	NFS 挂载选项的精细控制。SMB 卷将被忽略。要使用 NFS 版本 4.1 挂载卷，请在逗号分隔的挂载选项列表中包含 `nfsvers=4`以选择 NFS v4.1。存储类定义中设置的挂载选项会覆盖后端配置中设置的挂载选项。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小超过此值，则配置失败	"" （默认情况下不强制实施）
debugTraceFlags	故障排除时要使用的调试标志。示例， `\ {"api"： false ， "method"： true ， "discovery"： true} `。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	空
nasType	配置NFS或SMB卷创建。选项包括 nfs， smb 或为空。默认情况下、将设置为空会将NFS卷设置为空。	nfs

参数	Description	Default
`supportedTopologies`	表示此后端支持的区域和区域的列表。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，单位为 MiB/秒。仅支持手动 QoS 容量池。	`4 MiB/sec`

有关网络功能的详细信息、请参见 "配置Azure NetApp Files 卷的网络功能"。

考虑 **Azure** 云环境（**26.06**）

从 26.06 版本开始，Trident 支持在多个 Azure 云环境中创建和管理 Azure NetApp Files 后端。

支持的 Azure 云包括：

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

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

对于 Azure Government 后端，请在后端配置中设置 `cloudConfiguration.cloudName`：

```
cloudConfiguration:
  cloudName: AzureGovernment
```

所需权限和资源

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

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

如果虚拟网络与Azure NetApp Files(ANF) 存储帐户位于不同的资源组中，则在配置后端资源组列表时，请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：

Type	格式。
Resource group	< 资源组 >
NetApp 帐户	< 资源组 >/< NetApp 帐户 >
容量池	< 资源组 >/< NetApp 帐户 >/< 容量池 >
虚拟网络	< 资源组 >/< 虚拟网络 >
Subnet	< 资源组 >/< 虚拟网络 >/< 子网 >

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	Description	Default
`exportRule`	新卷的导出规则。`exportRule` 必须是以CIDR表示法表示的任意IPv4地址或IPv4子网组合的逗号分隔列表。SMB卷已忽略。	"0.0.0.0/0
`snapshotDir`	访问 ` .snapshot` 目录	`true`, `false` (显式设置)。
s大小	新卷的默认大小	"100 克 "
`unixPermissions`	新卷的UNIX权限(4个八进制数字)。SMB卷已忽略。	"" （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。

▼ 最低配置

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并尝试某些操作时、此配置是理想的选择、但实际上、您需要为所配置的卷提供额外的范围界定。

```
---
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**

此后端配置会出现异常 `subscriptionID`， `tenantID`， `clientID`，和 `clientSecret`，在使用受管身份时是可选的。

```
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的云身份**

此后端配置会出现异常 `tenantID`， `clientID`，和 `clientSecret`，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
```

▼ 使用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

此后端配置将卷放置在 Azure 中 `eastus`具有手动 QoS 容量池的位置。

```
---
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为一个子网，并修改了某些卷配置默认值。

```
---
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。有关更多信息，请参阅"使用 CSI 拓扑"。

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

```
capacityPools:
  - application-group-1/account-1/ultra-1
  - application-group-1/account-1/ultra-2
supportedTopologies:
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-1
  - topology.kubernetes.io/region: eastus
    topology.kubernetes.io/zone: eastus-2
```

存储类定义

以下内容 `StorageClass` 定义是指上述存储池。

使用的示例定义 **`parameter.selector`** 字段

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卷的示例定义

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。

▼ 默认命名空间上的基本配置

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

▼ 每个命名空间使用不同的密钥

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

▼ 每个卷使用不同的密钥

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

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

创建后端

创建后端配置文件后，运行以下命令：

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

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

```
tridentctl logs
```

确定并更正配置文件中的问题后，您可以再次运行 create 命令。

Google Cloud NetApp卷

配置 **Google Cloud NetApp Volumes**

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

概述

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

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

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

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

这适用于以下环境：

- Trident 26.02 及更高版本
- 在 GCP 上运行的 Google Kubernetes Engine （GKE） 或 Red Hat OpenShift 集群
- OpenShift Workload Identity Federation (WIF) 需要 Trident 26.10 或更高版本
- Google Cloud NetApp Volumes 存储池

要配置块 (iSCSI) 存储，请参见 "配置块存储 (iSCSI)"。

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

在配置工作负载标识之前，请使用有权创建所需标识资源的帐户。对于 GKE，这需要 Google Cloud 权限才能创建和配置 Google Cloud 服务帐户。对于 OpenShift，这需要创建或选择集群的 WIF 配置以及配置 Google Cloud Workload Identity 池和提供商的权限。

使用 **GKE** 工作负载身份

要将 GKE 工作负载标识用于 Google Cloud NetApp Volumes，必须具有：

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

Trident 运算符

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

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

掌舵

使用 Helm 安装 Trident 时设置云提供商和云标识：

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

Tridentctl

通过指定云提供商和云身份来安装 Trident：

```
tridentctl install \
```

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

在 **OpenShift** 上使用 **Workload Identity Federation**

从 Trident 26.10 开始，您可以使用 Google Cloud Workload Identity Federation （WIF） 对在 GCP 上运行的 OpenShift 集群上的 Google Cloud NetApp Volumes 驱动程序进行身份验证。这适用于自管理的 OpenShift Container Platform （OCP，包括 IPI 部署） 和 OpenShift Dedicated （OSD），并同时支持 NAS (`google-cloud-netapp-volumes`） 和 SAN (`google-cloud-netapp-volumes-san`） 驱动程序。

WIF 将投影的 Kubernetes 服务帐户令牌交换为短期的 Google Cloud 凭据。Trident 使用这些凭据模拟 Google Cloud 服务帐户；该服务帐户的 IAM 角色授权其 Google Cloud NetApp Volumes API 调用。这样可以避免在后端配置中存储长期的服务帐户密钥。

> 在 OpenShift Dedicated 上，OIDC 提供程序可以与其他 OpenShift 组件共享。如果提供程序的 `allowedAudiences`列表为空，则 Google 接受提供程序的规范资源名称（带或不带 `https://`前缀），因此 Trident 受众无需更改提供程序。请勿仅为 Trident 设置明确的受众列表。如果提供程序已经有明确的受众，请在保留每个现有值的同时追加 Trident 受众。替换列表可能会导致其他 OpenShift 组件无法交换令牌。

准备 **OpenShift** 集群

对于 OSD，在创建集群时创建或选择 GCP WIF 配置。例如，OpenShift Cluster Manager CLI 可以创建一个：

```
ocm gcp create wif-config --name <wif-name> --project <gcp-project-id>
```

选择 Workload Identity Federation 作为集群身份验证类型，并将 WIF 配置与集群相关联。对于自我管理的 OCP，请配置信任集群服务帐户令牌颁发者的 Google Cloud OIDC 提供程序。集群配置和令牌投射步骤在 OSD 和自我管理的 OCP 之间有所不同；请遵循适用的 Red Hat 指导，并将投射的令牌提供给 Trident 控制器，位置为 `/var/run/secrets/wif-token/token`。

创建集群时，会配置 OSD 的 WIF 身份验证。Trident 的 WIF 受众是一个单独的安装设置；您无需重新创建集群即可为 Trident 启用 WIF。

在安装或配置 Trident 之前，请验证用于安装的 OpenShift 身份是否具有所选安装方法所需的集群权限。安装后，请确认控制器和节点 Pod 是否正常：

```
oc whoami
oc auth can-i create namespaces
oc auth can-i create csidrivers.storage.k8s.io
oc auth can-i create customresourcedefinitions.apiextensions.k8s.io
oc auth can-i create securitycontextconstraints.security.openshift.io

oc -n trident get pods
```

```
oc -n trident rollout status deploy/trident-controller --timeout=300s
oc -n trident rollout status ds/trident-node-linux --timeout=300s
```

使用 **WIF** 受众安装 **Trident**

仅当您在安装过程中配置受众时，Trident 才会为 WIF 投射令牌。在 Google Cloud 提供商和后端的 `wipCredentialConfig.audience`中使用相同的受众：

```
//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>
```

将 `cloudProvider`设置为 `GCP`，并使用您选择的安装方法配置 `gcpWIFAudience`：

Tridentctl

```
tridentctl install \
  --cloud-provider GCP \
  --gcp-wif-audience "<audience>" \
  --namespace trident
```

Trident 运算符

在 `TridentOrchestrator`规范中设置这些字段：

```
spec:
  cloudProvider: GCP
  gcpWIFAudience: "<audience>"
```

掌舵

安装 chart 时设置这些值：

```
helm install trident netapp-trident/trident-operator \
  --set cloudProvider=GCP \
  --set-string gcpWIFAudience="<audience>"
```

请勿同时为 OpenShift WIF 设置 GKE `cloudIdentity` / `--cloud-identity` 选项。安装后请勿修补 Trident 控制器 Deployment；Operator 会还原手动更改。投影令牌可在后端通过 `/var/run/secrets/wif-token/token`获取。 `trident-main` 容器不包含 shell，因此请检查控制器 Deployment 的卷配置，而不是使用 `oc exec`来检查文件：

```
oc -n trident get deploy trident-controller -o yaml
```

配置 Google Cloud 信任和权限

为集群配置 Google Cloud Workload Identity Pool 和 OIDC 提供程序。提供程序颁发者必须与令牌的 `iss`声明相匹配，并且其属性映射必须将 `assertion.sub`映射到 `google.subject`。提供程序允许的受众必须与投影令牌的 `aud`声明相匹配。

仅在使用 OIDC 发现时才需要公共 `.well-known/openid-configuration` 和 JWKS 端点。Google 还支持使用 `jwksJson` 或 `--jwk-json-path` 直接上传 JWKS。

在 Trident 将模拟的 Google Cloud 服务帐户上，向 WIF 主体授予 `roles/iam.workloadIdentityUser`角色。对于`system:serviceaccount:trident:trident-controller`的令牌主题，主体具有以下形式：

```
principal://iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/subject/system:serviceaccount:trident:trident-controller
```

在包含 Google Cloud NetApp Volumes 资源的项目上，向模拟服务帐户授予 `roles/netapp.admin`，或授予具有所需权限的同等自定义角色。这些是单独的授权：第一个允许模拟，第二个授权存储 API 操作。

如果令牌受众和提供商配置不匹配，Google Security Token Service 可能会因 `invalid_grant`受众错误而拒绝交换。使用以下命令检查提供商配置：

```
gcloud iam workload-identity-pools providers describe <provider-id> \
  --project=<wif-pool-project-id> \
  --location=global \
  --workload-identity-pool=<pool-id> \
  --format="yaml(oidc.allowedAudiences,oidc.issuerUri,attributeMapping)"
```

在诊断令牌配置时，将令牌的 `iss`、 `aud`和 `sub`声明与提供程序和 IAM 配置进行比较。请勿共享或记录完整的投影令牌。

准备 Google Cloud NetApp Volumes 存储

在后端使用的区域中创建或标识存储池。NAS 后端需要 FILE 池；SAN 后端使用 "配置块存储 (iSCSI)" 中所述的 Flex Unified 池。确保池的 VPC 和专用服务访问（PSA）配置适用于集群网络。

例如，在为池选择 PSA 范围之前，列出网络的现有全局 VPC 对等范围：

```
gcloud compute addresses list \
  --project <gcnv-project-id> \
  --global \
  --filter="purpose=VPC_PEERING AND network:<network-name>" \
```

```
  --format="table(name,address,prefixLength,status,network,purpose)"
```

在 `location`中使用池的区域，并在需要时使用 `storagePools`中的完整资源路径显式选择池。即使身份验证成功，区域和池位置不匹配、网络不正确或池不可用也可能会阻止发现或配置。

配置 Trident 后端

在提供程序配置和 `wipCredentialConfig.audience`中均使用投影令牌受众。 `projectNumber` 字段标识 Google Cloud NetApp Volumes 项目；受众使用包含 Workload Identity Pool 的项目编号。将此示例中的占位符替换为您环境的值：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-nas-wif
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
  projectNumber: "<project-number>"
  location: "<region>"
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
    network: "<vpc-network>"
    serviceLevel: Premium
  wipCredentialConfig:
    type: external_account
    audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
    subject_token_type: urn:ietf:params:oauth:token-type:jwt
    token_url: https://sts.googleapis.com/v1/token
    credential_source:
      file: /var/run/secrets/wif-token/token
    service_account_impersonation_url:
https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken
```

对于 SAN 后端，请将 `google-cloud-netapp-volumes-san`用作 `storageDriverName`和适用的 Flex Unified 池设置。除此之外，两个驱动程序的 WIF 凭据配置相同。

应用后端并检查其观察状态：

```
oc apply -f <backend-file>.yaml
oc -n trident get tridentbackendconfig <backend-name> -o yaml
```

在测试配置之前，确认后端是 `Bound`并且其最后一次操作成功。检查实时 `spec`和 `status`；以前应用的配置注释可能不会反映以后的更改。使用 WIF 时，请勿包含静态 `apiKey`凭据。

验证端到端配置

创建一个以该后端为目标的 StorageClass，然后创建一个小型测试 PVC。例如，NAS StorageClass 可以选择一个后端标签：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-wif-smoke
provisioner: csi.trident.netapp.io
parameters:
  backendType: "google-cloud-netapp-volumes"
  selector: "auth=wif"
allowVolumeExpansion: true
```

将匹配的 `auth: wif`标签添加到 NAS 后端的存储条目中，使用 `gcnv-wif-smoke`创建 PVC，并验证其是否到达 `Bound。对于 SAN 冒烟测试，请使用 `google-cloud-netapp-volumes-san`、SAN 驱动程序设置和适当的 Flex Unified 池。完成后删除测试 PVC 和 StorageClass。

对常见故障进行故障排除

症状	需要检查的内容
`invalid_grant` 或受众不匹配	将投影令牌的 `aud`与提供程序允许的受众和 `wipCredentialConfig.audience`进行比较。
`iam.serviceAccounts.getAccessToken` 权限被拒绝	检查 Trident 控制器的 `sub`的确切 `WIF` 主体在模拟的服务帐户上是否具有 `roles/iam.workloadIdentityUser`。
`netapp.storagePools.list` 权限被拒绝	检查模拟服务帐户在正确的项目上是否具有 Google Cloud NetApp Volumes 权限，例如 `roles/netapp.admin`。
未找到任何池或未找到该池	请检查后端 `location`、池区域和类型、网络、PSA 配置以及任何显式配置的存储池资源路径。
后端保持未绑定	检查 Trident 控制器日志和 TridentBackendConfig 状态，以查找底层身份验证、权限或池发现错误。

要检查控制器日志以了解后端协调错误，请执行以下操作：

```
oc -n trident logs deploy/trident-controller -c trident-main --tail=400
```

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

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

驱动程序详细信息

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

驱动程序支持以下访问模式：

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

驱动程序	协议	卷模式	支持的访问模式	支持的文件系统
`google-cloud-netapp-volumes`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	`nfs， smb`

配置 **Trident NAS** 后端

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

配置 **NAS** 卷

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

StorageClass 适用于 NFS 卷

要配置 NFS 卷，请将 `nasType`设置为 `nfs`。

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

StorageClass 适用于 SMB 卷

要配置 SMB 卷，请将 `nasType`设置为 `smb`并提供凭据。

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

PersistentVolumeClaim 示例 (RWX)

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

PersistentVolumeClaim 示例 (RWO)

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

NAS 卷使用 `volumeMode: Filesystem`。

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

使用 `google-cloud-netapp-volumes-san`驱动程序从 Google Cloud NetApp Volumes Flex Unified 存储池配置 iSCSI 块卷。

SAN 驱动程序仅支持块工作负载。要配置 NAS 卷，请使用 `google-cloud-netapp-volumes`驱动程序。

概述

驱动程序从 Flex Unified 存储池配置卷，并将其作为 iSCSI 块设备呈现给 Kubernetes 节点。

驱动程序支持以下环境：

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

从 Trident 26.10 开始，SAN 驱动程序支持在 GCP 上运行的自我管理 OpenShift Container Platform（OCP，包括 IPI 部署）和 OpenShift Dedicated（OSD）集群上的 Google Cloud Workload Identity Federation（WIF）。有关 WIF 先决条件和后端配置，请参见 "在 OpenShift 上使用 Workload Identity Federation"。使用此页面上的 SAN 驱动程序名称和 Flex Unified 存储设置。

为 **SAN** 配置 **WIF**

WIF 将投影的 Kubernetes 服务帐户令牌交换为短期的 Google Cloud 凭据。Trident 在调用 Google Cloud NetApp Volumes API 时使用这些凭据来模拟 Google Cloud 服务帐户。

为 OpenShift 集群配置 Google Cloud Workload Identity Pool 和 OIDC 提供程序。提供程序颁发者必须与令牌的

iss`声明相匹配，并且其属性映射必须将 `assertion.sub`映射到 `google.subject。

使用提供程序资源名称作为 WIF 受众安装 Trident。这会使 Trident 将控制器服务帐户令牌投影到 /var/run/secrets/wif-token/token。

Trident 运算符

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  cloudProvider: GCP
  gcpWIFAudience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
```

掌舵

```
helm install trident <trident-operator-package> \
  --namespace trident \
  --create-namespace \
  --set cloudProvider=GCP \
  --set-string gcpWIFAudience="//iam.googleapis.com/projects/<wif-pool-
project-number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
```

Tridentctl

```
tridentctl install \
  --namespace trident \
  --cloud-provider GCP \
  --gcp-wif-audience "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
```

为 Trident 安装和 wipCredentialConfig.audience`使用相同的受众。请勿同时配置 GKE `cloudIdentity`或 `--cloud-identity。请勿修补控制器 Deployment 以添加投影令牌；Trident operator 可以覆盖该更改。

> 在 OpenShift Dedicated 上，OIDC 提供程序可以与其他 OpenShift 组件共享。如果提供程序的 `allowedAudiences`列表为空，则 Google 接受提供程序的规范资源名称（带或不带 `https://`前缀），因此上述受众无需更改提供程序。请勿仅为 Trident 设置明确的受众列表。如果提供程序已有明确的受众，请在保留每个现有值的同时追加 Trident 受众。替换该列表可能会阻止其他 OpenShift 组件交换令牌。

向 Trident 控制器服务帐户的 WIF 主体授予 Trident 将模拟的 Google Cloud 服务帐户上的 `roles/iam.workloadIdentityUser`角色。对于默认的 `Trident` 控制器服务帐户，令牌主体为 `system:serviceaccount:trident:trident-controller`；请使用为您的安装配置的主体。在包含 Google Cloud NetApp Volumes 资源的项目上，向模拟的服务帐户授予 `roles/netapp.admin`或同等权限。

将以下节添加到 SAN 后端的 `spec`：

```
wipCredentialConfig:
  type: external_account
  audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<pool-
id>/providers/<provider-id>"
  subject_token_type: urn:ietf:params:oauth:token-type:jwt
  token_url: https://sts.googleapis.com/v1/token
  credential_source:
    file: /var/run/secrets/wif-token/token
  service_account_impersonation_url:
https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken
```

两个 GCNV 驱动程序的 WIF 配置相同。SAN 后端仍必须使用 `google-cloud-netapp-volumes-san`和受支持的 Flex Unified 存储池。

Flex Unified 存储池

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

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

后端 `location`必须与 Flex Unified 池的区域相匹配。如果 Trident 找不到该池，请验证区域、网络和池资源路径。

配置 Trident SAN 后端

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-san
  namespace: trident
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: "<project-number>"
  location: "<region>"
```

```
  sdkTimeout: "600"
  storage:
  - labels:
      cloud: gcp
      performance: flex
    network: "<vpc-network>"
    serviceLevel: Flex
```

创建 StorageClass

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

在 StorageClass 中指定文件系统类型，而不是在后端。

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

驱动程序支持以下文件系统类型：

- `ext4` (默认)
- `ext3`
- `xfs`

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

验证 WIF 配置

应用后端并验证 TridentBackendConfig 是否已绑定且其上次操作已成功：

```
oc apply -f <backend-file>.yaml
oc -n trident get tridentbackendconfig <backend-name> -o yaml
```

使用此页面上的 `gcnv-san` StorageClass 创建 PVC，并验证其是否到达 `Bound`。这将同时验证 WIF 交换、服务帐户模拟、Google Cloud NetApp Volumes 权限和 SAN 池选择。完成后删除测试 PVC。

如果后端协调失败，请检查 Trident 控制器日志：

```
oc -n trident logs deploy/trident-controller -c trident-main --tail=400
```

有关常见错误，请检查以下内容：

- 对于 `invalid_grant`受众错误，请将令牌的 `aud`声明与 OIDC 提供程序允许的受众和 `wipCredentialConfig.audience`进行比较。
- 对于 `iam.serviceAccounts.getAccessToken`权限错误，请验证 Trident 控制器的 WIF 主体对模拟服务帐户是否具有 `roles/iam.workloadIdentityUser`。
- 对于 `netapp.storagePools.list`权限错误，请验证模拟服务帐户对正确的项目具有所需的 Google Cloud NetApp Volumes 权限。
- 如果未找到池，请验证其位置和类型是否受支持，后端区域是否与池区域匹配，以及后端是否引用正确的网络和池。

配置块卷

ReadWriteOnce（RWO）

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

ReadWriteOncePod（RWOP）

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

ReadOnlyMany（ROX）

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

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

ReadWriteMany (RWX) — 仅原始块

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

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

块卷行为

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

块卷：

- 使用 iSCSI 协议
- 支持文件系统和原始块呈现
- 由 Trident 连接和管理

- 支持多种 Kubernetes 访问模式

访问模式

Trident 配置的块卷支持以下访问模式：

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

volumeMode 行为

`volumeMode` 字段控制块卷的暴露方式：

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

支持的操作

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

- 创建
- 删除
- 克隆
- Snapshot
- 调整大小
- 导入

卷大小开销

Google Cloud NetApp Volumes 块卷包括内部元数据开销。因此，内核可见的设备大小小于配置的容量。

观察到的开销：

- 创建卷时约为 300 KiB
- 调整大小后最高约 107 MiB

Google Cloud NetApp Volumes 接受以整数 GiB 为单位的容量分配。为确保可用设备大小满足或超过 PVC 请求，Trident：

- 将请求的大小向上舍入至下一个整 GiB。
- 添加 1 GiB 缓冲区。

例如：

- PVC 请求：100 GiB。
- 配置容量：101 GiB。
- 应用程序可见的可用容量：至少 100 GiB。

Pod 示例

文件系统挂载的块卷 (RWO)

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

原始块设备 (RWX)

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

连接和挂载行为

对于从 Google Cloud NetApp Volumes 配置的 SAN 卷：

- Trident 在 Flex Unified 存储池中创建逻辑单元号 (LUN)。
- 在发布期间， Trident 将 LUN 映射到每个节点的主机组。
- 在节点暂存期间，Trident：
  - 登录到 iSCSI 目标
  - 发现 LUN
  - 配置多路径
- 如果 `volumeMode: Filesystem`，Trident 会根据需要格式化设备并挂载它。
- 如果 `volumeMode: Block`，Trident 会连接设备并将其直接暴露给 pod，无需格式化或挂载。

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

准备配置**Google Cloud NetApp**卷后端

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

NFS 或 **SMB** 卷的先决条件

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

在配置Google Cloud NetApp卷后端之前、请确保您满足以下条件：

- 配置有Google Cloud NetApp卷服务的Google Cloud帐户。请参阅 "Google Cloud NetApp卷"。
- 您的Google Cloud帐户的项目编号。请参阅 "确定项目"。
- 具有NetApp卷管理员角色的Google Cloud服务帐户 (`roles/netapp.admin`。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"。

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

Google Cloud NetApp Volumes后端配置选项和示例

了解Google Cloud NetApp卷的后端配置选项并查看配置示例。

后端配置选项

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

参数	Description	Default
ve版本		始终为 1
storageDriverName	存储驱动程序的名称	的值 storageDriverName 必须指定为"gosle-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	用于指定用于创建卷的存储池的可选参数。	
projectNumber	Google Cloud 帐户项目编号。此值可在Google Cloud门户主页上找到。	
位置	Trident创建GCNV卷的Google Cloud位置。创建跨区域Kubnetes集群时、在中创建的卷 location 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成本。	
apiKey	具有角色的Google Cloud服务帐户的API密钥 netapp.admin 。它包括 Google Cloud 服务帐户专用密钥文件的 JSON 格式的内容（逐字复制到后端配置文件）。 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	精细控制 NFS 挂载选项。	"nfsvers=3"
limitVolumeSize	如果请求的卷大小超过此值、则配置失败。	"" （默认情况下不强制实施）
s服务级别	存储池及其卷的服务级别。值为 flex、、 standard premium`或 `extreme。	
标签	要应用于卷的一组任意 JSON 格式的标签	""
网络	用于GCNV卷的Google Cloud网络。	
debugTraceFlags	故障排除时要使用的调试标志。例如，{"api":false, "method":true}。除非您正在进行故障排除并需要详细的日志转储，否则请勿使用此功能。	空
nasType	配置NFS或SMB卷创建。选项包括 nfs， smb 或为空。默认情况下、将设置为空会将NFS卷设置为空。	nfs
supportedTopologies	表示此后端支持的区域和区域的列表。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

您可以在中控制默认卷配置 `defaults` 部分。

参数	Description	Default
`exportRule`	新卷的导出规则。必须是IPv4地址任意组合的逗号分隔列表。	"0.0.0.0/0
`snapshotDir`	访问 ` .snapshot` 目录	`true`, `false` (默认行为可能会有所不同。显式设置) NFSv3 的 "false"
`sSnapshot 预留`	为快照预留的卷百分比	""(接受默认值0)
`unixPermissions`	新卷的UNIX权限(4个八进制数字)。	""

示例配置

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

▼ 最低配置

这是绝对的最低后端配置。使用此配置时、Trident会发现在配置位置委派给Google Cloud NetApp卷的所有存储池、并随机将新卷放置在其中一个池上。由于 `nasType` 省略了、因此会 `nfs` 应用默认设置、后端将为NFS卷配置。

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

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

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-gcnv-secret
type: Opaque
stringData:
  private_key_id: "<id_value>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <key_value>
    -----END PRIVATE KEY-----
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcnv
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes
```

```
  projectNumber: "123455380079"
  location: europe-west6
  serviceLevel: premium
  apiKey:
    type: service_account
    project_id: my-gcnv-project
    client_email: myproject-prod@my-gcnv-project.iam.gserviceaccount.com
    client_id: "103346282737811234567"
    auth_uri: https://accounts.google.com/o/oauth2/auth
    token_uri: https://oauth2.googleapis.com/token
    auth_provider_x509_cert_url:
https://www.googleapis.com/oauth2/v1/certs
    client_x509_cert_url:
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
```

▼ **SMB卷的配置**

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

```
    name: backend-tbc-gcnv-secret
```

▼ 使用StoragePools筛选器的配置

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

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

```
    name: backend-tbc-gcnv-secret
```

▼ 虚拟池配置

此后端配置可在一个文件中定义多个虚拟池。虚拟池在一节中进行了定义 `storage` 。如果您有多个存储池支持不同的服务级别、而您希望在Kubbernetes中创建表示这些服务级别的存储类、则这些选项板非常有用。虚拟池标签用于区分池。例如、在以下示例中 `performance` 、标签和 `serviceLevel` 类型用于区分虚拟池。

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

有关详细信息，请参阅 "虚拟池"。

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

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

```
https://www.googleapis.com/robot/v1/metadata/x509/myproject-prod%40my-
gcnv-project.iam.gserviceaccount.com
  credentials:
    name: backend-tbc-gcnv-secret
  defaults:
    snapshotReserve: "10"
    exportRule: 10.0.0.0/24
  storage:
    - labels:
        performance: extreme
      serviceLevel: extreme
      defaults:
        snapshotReserve: "5"
        exportRule: 0.0.0.0/0
    - labels:
        performance: premium
      serviceLevel: premium
    - labels:
        performance: standard
      serviceLevel: standard
```

▼ 适用于**GKE**的云身份

当您使用 GKE 云身份安装 Trident 时，后端将作为工作负载身份进行身份验证，并省略 `apiKey`块。云身份的配置方式对于两种 `Google Cloud NetApp Volumes` 驱动程序相同。唯一的区别是 `storageDriverName`值：对于 NAS（NFS 或 SMB）卷使用 `google-cloud-netapp-volumes`，对于 SAN（iSCSI 块）卷使用 `google-cloud-netapp-volumes-san`。

> 对 `google-cloud-netapp-volumes-san` 驱动程序的云身份支持需要 Trident 26.06 或更高版本。SAN 卷是从 Flex Unified 存储池调配的，因此请将 `serviceLevel` 设置为 `flex`。

NAS 驱动程序（<code>google-cloud-netapp-volumes</code>）

```
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
```

SAN 驱动程序（<code>google-cloud-netapp-volumes-san</code>）

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-gcp-gcnv-san
spec:
  version: 1
  storageDriverName: google-cloud-netapp-volumes-san
  projectNumber: '012345678901'
  network: gcnv-network
  location: us-west2
  serviceLevel: flex
  storagePool: pool-flex1
```

▼ 支持的拓扑配置

Trident可以根据区域和可用性区域为工作负载配置卷。 `supportedTopologies`此后端配置中的块用于提供每个后端的区域和分区列表。此处指定的区域和分区值必须与每个Kubnetes集群节点上标签中的区域和分区值匹配。这些区域和分区表示可在存储类中提供的允许值列表。对于包含后端提供的部分区域和区域的存储类、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，则后端配置出现问题。您可以使用命令说明后端 `kubectl get tridentbackendconfig <backend-name>` 、或者运行以下命令查看日志以确定原因：

```
tridentctl logs
```

确定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

下面是有关上述后端的基本 `StorageClass` 定义。

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

使用字段的示例定义 **`parameter.selector`**：

使用、 `parameter.selector` 您可以为每个指定 `StorageClass` "虚拟池" 用于托管卷的。卷将在选定池中定义各个方面。

```yaml
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
```

有关存储类的详细信息，请参见 "创建存储类。"。

SMB卷的示例定义

使用 `nasType`、 `node-stage-secret-name`和 `node-stage-secret-namespace`，您可以指定SMB卷并提供所需的Active Directory凭据。任何具有任何/无权限的Active Directory用户/密码均可用作节点阶段密钥。

▼ 默认命名空间上的基本配置

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

▼ 每个命名空间使用不同的密钥

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcnv-sc-smb
provisioner: csi.trident.netapp.io
parameters:
```

```
  backendType: "google-cloud-netapp-volumes"
  trident.netapp.io/nasType: "smb"
  csi.storage.k8s.io/node-stage-secret-name: "smbcreds"
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

▼ 每个卷使用不同的密钥

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

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

PVC定义示例

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

要验证PVC是否已绑定、请运行以下命令：

```
kubectl get pvc gcnv-nfs-pvc

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

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

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

概述

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

Trident 仅在卷创建时应用自动分层设置。Trident 26.10 不支持预配后更改。

概念

自动分层

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

分层策略

分层策略确定是否为卷启用自动分层。

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

冷却天数

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

配置模型

配置范围

可以在多个范围内配置自动分层：

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

Trident 根据每个设置的定义位置确定有效配置。

配置优先级

当在多个作用域中定义了相同的设置时，Trident 应用以下优先级顺序：

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

在较高优先级定义的设置将覆盖较低级别的值。

Trident 26.10 中支持的功能

Trident 26.10 支持 Google Cloud NetApp Volumes 的以下自动分层功能：

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

Trident 26.10 中不支持的功能

不支持以下操作：

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

后端配置参数

以下参数在 Trident 后端配置中定义时控制自动分层行为：

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

使用 PersistentVolumeClaim 注释的卷级覆盖

支持的注释

PersistentVolumeClaim 批注允许按卷覆盖自动分层设置。

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

示例：PersistentVolumeClaim 使用自动分层覆盖

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-tiering-pvc
  annotations:
    trident.netapp.io/tieringPolicy: auto
```

```
    trident.netapp.io/tieringMinimumCoolingDays: "45"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: google-cloud-netapp-volumes-auto-tiering
  resources:
    requests:
      storage: 500Gi
```

行为和限制

配置行为

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

平台限制

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

支持的值

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

配置 **Google Cloud NetApp Volumes ONTAP** 模式存储池

从 Trident 26.06 开始，您可以配置现有 `ontap-san`和 `ontap-nas`后端以在 Google Cloud NetApp Volumes (GCNV) ONTAP 模式存储池上预置存储。此功能是技术预览版。

备注	此功能是 Trident 26.10 中的技术预览功能。技术预览功能不支持用于生产环境。在正式发布 (GA) 之前，相关功能、配置字段和支持矩阵可能会发生变化。

工作原理

Trident 重用现有的 `ontap-san` 和 `ontap-nas` 驱动程序逻辑，并通过 GCNV 代理端点路由 ONTAP REST 调用。这种方法使您能够在 GCNV 托管的 ONTAP 集群上置备和管理存储，同时保持与直接 ONTAP 后端相同的操作工作流。ONTAP 模式不会引入新的存储驱动程序。您可以通过向 `ontap-san` 或 `ontap-nas` 后端添加 `gcnv` 配置块来针对每个后端启用它。

配置路径如下所示：

`PersistentVolumeClaim` → Trident (`ontap-san` 或 `ontap-nas`) → ONTAP REST 客户端 → GCNV 代理

→ GCNV 池中的 ONTAP 集群

支持和不支持的范围

此技术预览版支持以下功能：

- 驱动程序： `ontap-san` (iSCSI) 和 `ontap-nas` (NFS 或 SMB)。
- 通过代理的生命周期操作：后端初始化、卷创建和删除以及相关的 ONTAP REST 流。
- 身份验证模型：Workload Identity Pool (WIP)、服务帐户密钥和 Application Default Credentials (ADC) 回退。

以下内容超出了此技术预览的范围：

- `ontap-san-economy`、 `ontap-nas-economy`和 `ontap-nas-flexgroup`驱动程序。
- ASA r2 personality 路径。
- ZAPI 回退。ONTAP 模式仅使用 ONTAP REST。

前提条件

在配置 ONTAP 模式后端之前，请确保您已具备以下条件：

- Trident 26.06 或更高版本。
- 目标项目和位置中的 GCNV ONTAP 模式存储池。
- 用于代理操作的 GCP IAM 权限，适用于您的环境。
- 包含 `proxyURL`、 `projectNumber`、 `location`和 `poolID`的后端配置。
- 对于 iSCSI SAN 工作流，您的平台的节点端 iSCSI 和多路径先决条件。请参阅 "准备工作节点"。

配置后端

`TridentBackendConfig` 需要 `credentials` 引用 Kubernetes Secret。对于 ONTAP 模式，`secret` 为 GCP 代理访问提供凭据，而不是 ONTAP `managementLIF` / `username` / `password`。根据身份验证方法，它可以包含服务帐户密钥字段或为空。 `svm` 是可选的：每个 GCNV ONTAP 模式池当前有一个 SVM，省略时 Trident 会从池中获取它。

对于服务帐户密钥身份验证，请使用与原生 GCNV 相同的模式：非敏感字段位于 `gcnv.apiKey`中，`private_key`和 `private_key_id`位于 secret 中。请参见 "Google Cloud NetApp Volumes 配置示例"。对于 Workload Identity Pool 或 GKE 云身份验证，请参见 身份验证 中的示例。

ontap-san 后端

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-san
  namespace: trident
```

```
spec:
  version: 1
  storageDriverName: ontap-san
  backendName: gcnv-ontap-san
  credentials:
    name: gcnv-sa-secret
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
  svm: "<svm-name>"
```

ontap-nas 后端

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas
  credentials:
    name: gcnv-sa-secret
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
    apiKey:
      type: service_account
      project_id: "<project-id>"
      client_email: "<service-account-email>"
      client_id: "<client-id>"
      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: "<client-x509-cert-url>"
  svm: "<svm-name>"
```

机密

```
apiVersion: v1
kind: Secret
metadata:
  name: gcnv-sa-secret
  namespace: trident
type: Opaque
stringData:
  private_key_id: "<private-key-id>"
  private_key: |
    -----BEGIN PRIVATE KEY-----
    <private-key>
    -----END PRIVATE KEY-----
```

备注	请勿将 `private_key`或 `private_key_id`放入 `TridentBackendConfig` spec 中——CRD 会拒绝它们。使用 `ontap-san`，并采用相同的 `gcnv`和凭据模式进行块存储。

身份验证

Trident 按以下顺序解析代理访问凭据：

1. 工作负载身份池（WIP）
2. 服务帐户密钥
3. 应用程序默认凭据（ADC）

对于此技术预览，请使用以下做法：

- 在可用时，首选 Workload Identity Pool。
- 请勿在版本控制后端文件中嵌入原始私钥。将凭据存储在 Kubernetes 密钥中。
- 将最小特权 IAM 作用域应用于服务帐户。

> `gcnv` 块中的 `location` 必须与池的位置匹配：区域池使用可用区，区域性池使用区域。

工作负载身份池

将 `gcnv.wipCredential`用于在 `TridentBackendConfig`上配置的显式外部帐户联合。在 `OpenShift` 上，`Trident` 会将令牌投射到 `/var/run/secrets/wif-token/token`；这是 Trident 为 OpenShift WIF 投射的唯一令牌路径，因此请将其用于 `credential_source.file`。按照"使用 Workload Identity Federation 配置 Google Cloud NetApp Volumes"中的说明安装具有 WIF 受众的 Trident。`credentials`字段仍然引用一个 `Secret`，但该 `Secret` 不得包含 `private_key`。

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: gcnv-ontap-empty
  namespace: trident
type: Opaque
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas-wip
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas-wip
  useREST: true
  nasType: nfs
  nfsMountOptions: nfsvers=4.1
  credentials:
    name: gcnv-ontap-empty
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
    wipCredential:
      type: external_account
      audience: "//iam.googleapis.com/projects/<wif-pool-project-
number>/locations/global/workloadIdentityPools/<wif-pool-
id>/providers/<provider-id>"
      subject_token_type: "urn:ietf:params:oauth:token-type:jwt"
      token_url: "https://sts.googleapis.com/v1/token"
      service_account_impersonation_url:
"https://iamcredentials.googleapis.com/v1/projects/-
/serviceAccounts/<service-account-email>:generateAccessToken"
      credential_source:
        file: "/var/run/secrets/wif-token/token"
```

GKE 云身份

对于 GKE，请在安装时配置 Trident 的云身份，如"使用 GKE 工作负载身份"中所述。后端使用应用程序默认凭据（ADC）；请省略 `gcnv.apiKey`和 `gcnv.wipCredential。 credentials`字段仍然引用 `Secret`，但不要在其中放置 `private_key。

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: gcnv-ontap-empty
  namespace: trident
type: Opaque
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: gcnv-ontap-nas-wi
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: gcnv-ontap-nas-wi
  useREST: true
  nasType: nfs
  nfsMountOptions: nfsvers=4.1
  credentials:
    name: gcnv-ontap-empty
    type: secret
  gcnv:
    proxyURL: "https://netapp.googleapis.com"
    projectNumber: "<project-number>"
    location: "<pool-location>"
    poolID: "<pool-id>"
```

验证和初始化行为

Trident 在初始化期间验证 ONTAP 模式后端，并快速失败而不是部分初始化。

如果缺少必填 `gcnv`字段、驱动程序不支持 ONTAP 模式或凭据解析失败，则后端初始化失败。

已知限制

- 此功能是技术预览。行为和支持的矩阵可能会在 GA 之前发生变化。
- ZAPI 回退未在 ONTAP 模式中使用。
- 协议和节点就绪性要求仍然适用。例如，iSCSI 工作流需要 "准备工作节点" 中描述的节点端先决条件。
- 现有的 ONTAP 和 GCNV 环境约束仍然适用。

升级和兼容性

ONTAP 模式可通过 `gcnv` 配置块在每个后端中选择启用：

- 现有的非 GCNV ONTAP 后端不受影响。
- 通过后端级配置支持将直接 ONTAP 后端和 ONTAP 模式代理后端相结合的混合部署。每个后端都是独立配

置的。

- 若要停止使用 ONTAP 模式，请删除或更换受影响的后端。无需全局切换。

下一步是什么?

在 TBC 之前应用 secret；使用 `kubectl apply`和 `-n trident`来匹配示例。

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提供了 `solidfire-san`用于与集群通信的存储驱动程序。支持的访问模式包括：*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`卷模式、Trident会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	卷模式	支持的访问模式	支持的文件系统
solidfire-san	iSCSI	块	Rwo、ROX、rwx、RWOP	无文件系统。原始块设备。
solidfire-san	iSCSI	文件系统	Rwo、RWO1.	xfs， ext3，ext4

开始之前

在创建Element后端之前、您需要满足以下要求。

- 运行 Element 软件的受支持存储系统。
- NetApp HCI/SolidFire 集群管理员或租户用户的凭据，可用于管理卷。
- 所有 Kubernetes 工作节点都应安装适当的 iSCSI 工具。请参见 "工作节点准备信息"。

后端配置选项

有关后端配置选项，请参见下表：

参数	Description	Default
ve版本		始终为 1
storageDriverName	存储驱动程序的名称	始终为SolidFire SAN
backendName	自定义名称或存储后端	"iSCSI_"+存储(SolidFire) IP地址
端点	使用租户凭据的 SolidFire 集群的 MVIP	

参数	Description	Default
sVIP	存储（ iSCSI ） IP 地址和端口	
标签	要应用于卷的一组任意 JSON 格式的标签。	""
租户名称	要使用的租户名称（如果未找到，则创建）	
InitiatorIFace	将 iSCSI 流量限制为特定主机接口	default
UseCHAP	使用CHAP对iSCSI进行身份验证。Trident使用CHAP。	true
访问组	要使用的访问组 ID 列表	查找名为Trident的访问组的ID
类型	QoS 规范	
limitVolumeSize	如果请求的卷大小超过此值，则配置失败	"" （默认情况下不强制实施）
debugTraceFlags	故障排除时要使用的调试标志。例如、 {"api"：false、"METHO"：true}	空

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

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

此示例显示了一个后端文件，该文件使用 CHAP 身份验证并使用特定 QoS 保证对三种卷类型进行建模。然后，您很可能会使用 IOPS storage class 参数定义存储类以使用其中的每种类型。

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

此示例显示了使用虚拟池配置的后端定义文件以及引用这些池的StorageClasses。

配置时、Trident会将存储池上的标签复制到后端存储LUN。为了方便起见、存储管理员可以按标签为每个虚拟池和组卷定义标签。

在下面所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、这些存储池设置了 `type` 在Silver。虚拟池在中进行定义 `storage` 部分。在此示例中、某些存储池会设置自己的类型、而某些存储池会覆盖上面设置的默认值。

```
---
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定义引用了上述虚拟池。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。卷将在选定虚拟池中定义各个方面。

第一个StorageClass(`solidfire-gold-four`)将映射到第一个虚拟池。这是唯一一个提供金牌性能的池 `Volume Type QoS`。最后一个StorageClass(`solidfire-silver`)会调用任何提供银牌性能的存储池。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
```

了解更多信息

- "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提供了以下SAN存储驱动程序来与ONTAP集群进行通信。支持的访问模式包括：*ReadWriteOnce*(RWO)、*ReadOnlyMany*(ROX)、*ReadWriteMany*(rwx)、*ReadWriteOncePod*(RWOP)。

驱动程序	协议	卷模式	支持的访问模式	支持的文件系统
`ontap-san`	基于FC的iSCSI SCSI	块	Rwo、ROX、rwx、RWOP	无文件系统；原始块设备
`ontap-san`	基于FC的iSCSI SCSI	文件系统	Rwo、RWO1. Rox和rwx在文件系统卷模式下不可用。	`xfs ， ext3 ， ext4`
`ontap-san`	NVMe/TCP 请参见NVMe/TCP的其他注意事项。	块	Rwo、ROX、rwx、RWOP	无文件系统；原始块设备
`ontap-san`	NVMe/TCP 请参见NVMe/TCP的其他注意事项。	文件系统	Rwo、RWO1. 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
`ontap-san-economy.`	iSCSI	文件系统	Rwo、RWO1. Rox和rwx在文件系统卷模式下不可用。	`xfs ， ext3 ， ext4`

警告

- 使用 ... `ontap-san-economy` 只有当永久性卷使用量计数预期高于时、才会显示此值 "支持的ONTAP卷限制"。
- 使用 ... `ontap-nas-economy` 只有当永久性卷使用量计数预期高于时、才会显示此值 "支持的ONTAP卷限制" 和 `ontap-san-economy` 无法使用驱动程序。
- 请勿使用 `ontap-nas-economy` 预测数据保护、灾难恢复或移动性的需求。
- `ontap-san` 和 `ontap-san-economy` 支持 ONTAP 卷自动调整大小。ONTAP NAS 驱动程序不支持此功能。有关详细信息和限制，请参见 "配置 ONTAP 卷自动调整大小"。

用户权限

Trident应以ONTAP或SVM管理员身份运行、通常使用集群用户 `vsadmin`或SVM用户、或者使用 `admin`具有相同角色的其他名称的用户。对于Amazon FSx for NetApp ONTAP部署、Trident应使用集群用户 `vsadmin`或SVM用户以ONTAP或SVM管理员身份运行、或者使用具有相同角色的其他名称的用户运行 `fsxadmin`。此 `fsxadmin`用户只能有限地替代集群管理员用户。

备注

如果使用 `limitAggregateUsage`参数、则需要集群管理员权限。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中创建一个可以由三端驱动程序使用的限制性更强的角色、但我们不建议这样做。大多数新

版本的 Trident 都会调用需要考虑的其他 API ，从而使升级变得困难且容易出错。

NVMe/TCP的其他注意事项

Trident使用以下驱动程序支持非易失性内存快速(NVMe)协议 `ontap-san`：

- IPv6
- NVMe卷的快照和克隆
- 调整NVMe卷大小
- 导入在Trident外部创建的NVMe卷、以便Trident可以管理其生命周期
- NVMe本机多路径
- 正常或非正常关闭K8s节点(24.06)

Trident不支持：

- NVMe 原生支持的 DH-HMAC-CHAP
- 设备映射程序(Device maper、DM)多路径
- Linux 统一密钥设置（LUKS）加密

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

准备使用ONTAP SAN驱动程序配置后端

了解使用ONTAP SAN驱动程序配置ONTAP后端的要求和身份验证选项。

要求

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

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

请记住，您还可以运行多个驱动程序，并创建指向其中一个驱动程序的存储类。例如，您可以配置使用 `ontap-san` 驱动程序的 `san-dev` 类和使用 `ontap-san-economy-one` 的 `san-default` 类。

所有Kubernetes工作节点都必须安装适当的iSCSI工具。请参见 "准备工作节点" 了解详细信息。

对ONTAP后端进行身份验证

Trident提供了两种对ONTAP后端进行身份验证的模式。

- Credential Based ： 具有所需权限的 ONTAP 用户的用户名和密码。建议使用 `admin` 或 `vsadmin` 等预定义的安全登录角色，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）（建议）。

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

警告	如果您尝试同时提供*凭据和证书*、则后端创建将失败、并显示一条错误、指出配置文件中提供了多种身份验证方法。

启用基于凭据的身份验证

Trident需要SVM范围/集群范围的管理员的凭据才能与ONTAP后端进行通信。建议使用标准的预定义角色，如 `admin`或 `vsadmin`。这样可以确保与未来ONTAP版本的正向兼容性、这些版本可能会公开未来Trident版本要使用的功能API。可以创建自定义安全登录角色并将其用于Trident、但不建议这样做。

后端定义示例如下所示：

YAML

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

JSON

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

请注意，后端定义是凭据以纯文本格式存储的唯一位置。创建后端后，用户名 / 密码将使用 Base64 进行编码并存储为 Kubernetes 密钥。创建或更新后端是唯一需要了解凭据的步骤。因此，这是一项仅由管理员执行的操作，由 Kubernetes 或存储管理员执行。

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 后端进行通信。后端定义需要三个参数。

- clientCertificate ： 客户端证书的 Base64 编码值。

- clientPrivateKey： 关联私钥的 Base64 编码值。
- trustedCACertifate： 受信任 CA 证书的 Base64 编码值。如果使用可信 CA ，则必须提供此参数。如果不使用可信 CA ，则可以忽略此设置。

典型的工作流包括以下步骤。

步骤

1. 生成客户端证书和密钥。生成时，将公用名（ Common Name ， CN ）设置为要作为身份验证的 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. 将可信 CA 证书添加到 ONTAP 集群。此问题可能已由存储管理员处理。如果未使用可信 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（从步骤 1 开始）。

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

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

4. 确认 ONTAP 安全登录角色支持 `cert` 身份验证方法。

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

5. 使用生成的证书测试身份验证。将 <SVM 管理 LIF> 和 <SVM 名称 > 替换为管理 LIF IP 和 ONTAP 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其他身份验证方法或轮换其凭据。这两种方式都适用：使用用户名 / 密码的后端可以更新为使用证书；使用证书的后端可以更新为基于用户名 / 密码的后端。为此、您必须删除现有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端.json文件执行`tridentctl backend update`。

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

```
"version": 1,
"storageDriverName": "ontap-san",
"backendName": "SanBackend",
"managementLIF": "1.2.3.4",
"svm": "vserver_test",
"username": "vsadmin",
"password": "password",
"storagePrefix": "myPrefix_"
}

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

备注	轮换密码时，存储管理员必须先在 ONTAP 上更新用户的密码。然后进行后端更新。轮换证书时，可以向用户添加多个证书。之后，后端将更新以使用新证书，然后可以从 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，也不会影响在之后建立的卷连接。后端更新成功表示Trident可以与ONTAP后端通信并处理未来的卷操作。

为**Trident**创建自定义**ONTAP**角色

您可以创建Privileges最低的ONTAP集群角色、这样就不必使用ONTAP管理员角色在Trident中执行操作。如果在Trident后端配置中包含用户名、则Trident将使用您创建的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**命令行界面

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系统管理器中执行以下步骤：

1. 创建自定义角色：
   a. 要在集群级别创建自定义角色，请选择*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*用户和角色*旁边的箭头图标(→)。
   c. 在*角色*下选择*+添加*。
   d. 定义角色的规则，然后单击*Save*。
2. 将角色映射到**Trident user**：+在*Users and Roles*页面上执行以下步骤：
   a. 在*用户*下选择添加图标*+*。
   b. 选择所需的用户名，然后在下拉菜单中为*rouser*选择一个角色。
   c. 单击 * 保存 * 。

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可以使用和 `ontap-san-economy`驱动程序的双向CHAP对iSCSI会话进行身份验证 `ontap-san`。这需要在后端定义中启用此 `useCHAP`选项。设置为时 `true`，Trident会将SVM的默认启动程序安全性配置为双向CHAP，并设置后端文件中的用户名和密钥。NetApp 建议使用双向 CHAP 对连接进行身份验证。请参见以下配置示例：

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap_san_chap
managementLIF: 192.168.0.135
svm: ontap_iscsi_svm
useCHAP: true
```

```
username: vsadmin
password: password
chapInitiatorSecret: cl9qxIm36DKyawxy
chapTargetInitiatorSecret: rqxigXgkesIpwxyz
chapTargetUsername: iJF4heBRT0TCwxyz
chapUsername: uh2aNCLSd6cNwxyz
```

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

除了 useCHAP=true 之外，后端定义还必须包括 chapInitiatorSecret ， chapTargetInitiatorSecret ， chapTargetUsername 和 chapUsername 字段。通过运行 tridentctl update 创建后端，可以更改这些密钥。

工作原理

如果将设置 `useCHAP`为true、则存储管理员将指示Trident在存储后端配置CHAP。其中包括：

- 在 SVM 上设置 CHAP ：
  - 如果SVM的默认启动程序安全类型为none (默认设置)*和*卷中已没有已有的LUN、则Trident会将默认安全类型设置为 CHAP、并继续配置CHAP启动程序以及目标用户名和密码。
  - 如果SVM包含LUN、则Trident不会在此SVM上启用CHAP。这样可确保对SVM上已存在的LUN的访问不受限制。
- 配置 CHAP 启动程序以及目标用户名和密码；必须在后端配置中指定这些选项（如上所示）。

创建后端后、Trident会创建相应的 `tridentbackend`CRD并将CHAP密码和用户名存储为Kubbernetes密码。Trident在此后端创建的所有PV,都将通过CHAP进行挂载和连接。

轮换凭证并更新后端

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

警告	更新后端的CHAP密码时、必须使用 `tridentctl`更新后端。请勿使用ONTAP命令行界面或ONTAP系统管理器更新存储集群上的凭据、因为Trident将无法接受这些更改。

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

现有连接不会受到影响；如果Trident在SVM上更新凭据、这些连接将继续保持活动状态。新连接将使用更新后的凭据、现有连接将继续保持活动状态。断开并重新连接旧的 PV 将导致它们使用更新后的凭据。

ONTAP SAN配置选项和示例

创建 ONTAP SAN 驱动程序并在 Trident 安装中使用。本节提供后端配置示例以及将后端映射到 StorageClasses 的详细信息。"ASA r2系统" 在其存储层的实现上与其他 ONTAP 系统（ASA、AFF 和 FAS）有所不同。这些差异会影响某些参数的使用，具体如注释所示。"详细了解 ASA r2 系统与其他 ONTAP 系统之间的差异"。在 Trident 后端配置中，无需指定您的系统为 ASA r2。当您选择 `ontap-san`作为 `storageDriverName`时，Trident 会自动检测 ASA r2 或其他 ONTAP 系统。某些后端配置参数不适用于 ASA r2 系统，如下表所示。

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

后端配置选项

从 26.06 版本开始，您可以启用 ONTAP 均衡放置，以便 ONTAP 为新卷选择聚合。有关要求和示例，请参阅 "使用 ONTAP 后端的均衡放置"。

有关后端配置选项，请参见下表：

参数	Description	Default
ve版本		始终为 1
storageDriverName	存储驱动程序的名称。创建后无法修改。要更新此参数，必须创建一个新的后端。	ontap-san`或 `ontap-san-economy
backendName	自定义名称或存储后端	驱动程序名称+"_"+ dataLIF

参数	Description	Default
m年 月 日	集群或SVM管理LIF的IP地址。 可以指定完全限定域名(FQDN)。 如果Trident是使用IPv6标志安装的、则可以设置为使用IPv6地址。IPv6地址必须用方括号定义，例如 `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。 有关无缝MetroCluster切换的信息，请参见MetroCluster示例。 备注：如果您使用的是"vsadmin"凭据、`managementLIF`则必须是SVM的凭据；如果使用的是"admin"凭据、则必须是集群的凭据 `managementLIF`。	"10.0.0.1"、"[2001：1234：abc：：：fefe]"
dataLIF	协议 LIF 的 IP 地址。如果Trident是使用IPv6标志安装的、则可以设置为使用IPv6地址。IPv6地址必须用方括号定义，例如 `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。*不指定iSCSI。*Trident使用"ONTAP 选择性LUN映射"发现建立多路径会话所需的iSCSI LUN。如果明确定义、则会生成警告 `dataLIF`。*省略MetroCluster。*请参见MetroCluster示例。	由SVM派生
sVM	要使用的 Storage Virtual Machine *对于MetroCluster省略。*请参见 MetroCluster示例。	如果指定了 SVM `managementLIF`，则派生
使用 `CHAP`	使用CHAP对iSCSI的ONTAP SAN驱动程序进行身份验证[布尔值]。将设置为 `true`、以便Trident配置双向CHAP并将其用作后端中给定SVM的默认身份验证。有关详细信息、请参见 "准备使用ONTAP SAN驱动程序配置后端" 。不支持 **FCP** 或 **NVMe/TCP**。	`false`
`chapInitiatorSecret`	CHAP 启动程序密钥。如果为 `useCHAP=true` ，则为必需项	""
标签	要应用于卷的一组任意 JSON 格式的标签	""
`chapTargetInitiatorSecret`	CHAP 目标启动程序密钥。如果为 `useCHAP=true` ，则为必需项	""
`chapUsername`	入站用户名。如果为 `useCHAP=true` ，则为必需项	""
`chapTargetUsername`	目标用户名。如果为 `useCHAP=true` ，则为必需项	""
客户端证书	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
`clientPrivateKey`	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""

参数	Description	Default
`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。	""
用户名	与ONTAP集群通信所需的用户名。用于基于凭证的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	""
密码	与ONTAP集群通信所需的密码。用于基于凭证的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	""
`sVM`	要使用的 Storage Virtual Machine	如果指定了 SVM `managementLIF`，则派生
s存储前缀	在 SVM 中预置新卷时使用的前缀。以后无法修改。要更新此参数，需要创建新的后端。	`trident`
聚合	用于配置的聚合（可选；如果设置，则必须分配给 SVM）。对于 `ontap-nas-flexgroup`驱动程序，此选项将被忽略。如果未分配，则可以使用任何可用的聚合来配置 FlexGroup 卷。 备注：当聚合在 SVM 中更新时，它会通过轮询 SVM 自动在 Trident 中更新，而无需重新启动 Trident Controller。当您在 Trident 中配置了特定聚合以配置卷时，如果该聚合被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将在 Trident 中移至失败状态。您必须将聚合更改为 SVM 上存在的聚合，或者将其完全删除，以使后端重新联机。 不要指定 **ASA r2** 系统。 当 **`useBalancedPlacement`**`设置为 **`true`**`时，请勿指定。	""
`useBalancedPlacement`	将新卷的聚合选择委托给 ONTAP [Boolean]。需要 ONTAP 9.19.1 或更高版本，并将 `useREST`设置为 `true`。不能与 `aggregate`结合使用。 有关要求和示例，请参见 "使用 ONTAP 后端的均衡放置"。	`false`
`limitAggregateUsage`	如果使用量超过此百分比，则配置失败。如果您使用的是Amazon FSx for NetApp ONTAP后端，请勿指定 `limitAggregateUsage`。提供的和 `vsadmin`不包含使用Trident检索聚合使用情况并对其进行限制所需的 `fsxadmin`权限。不要指定 **ASA r2** 系统。	"" （默认情况下不强制实施）
`limitVolumeSize`	如果请求的卷大小超过此值、则配置失败。此外、还会限制它为LUN管理的卷的大小上限。	""(默认情况下不强制实施)

参数	Description	Default
lunsPerFlexvol	每个 FlexVol 的最大 LUN 数，必须在 50 ， 200 范围内	100
debugTraceFlags	故障排除时要使用的调试标志。例如、 {"api"：false、"METHO"：true} 除非正在进行故障排除并需要详细的日志转储、否则请勿使用。	null
useREST	使用ONTAP REST API 的布尔参数。 `useREST`当设置为 `true`，Trident使用ONTAP REST API 与后端通信；当设置为 `false`，Trident使用 ONTAPI （ZAPI）调用与后端进行通信。此功能需要ONTAP 9.11.1 及更高版本。此外，使用的ONTAP登录角色必须具有访问 `ontapi`应用。这是通过预定义的 `vsadmin`和 `cluster-admin` 角色。从Trident 24.06 版本和ONTAP 9.15.1 或更高版本开始，`useREST`设置为 `true`默认；改变 `useREST`到 `false`使用 ONTAPI (ZAPI) 调用。 `useREST`完全符合 NVMe/TCP 的要求。 备注 NVMe 仅支持ONTAP REST API，不支持 ONTAPI (ZAPI)。 如果指定，则始终设置为 **`true`**适用于**ASA r2** 系统。	true 对于ONTAP 9.151或更高版本，否则 false。
sanType	用于为iSCSI、 nvme`NVMe/TCP或基于光纤通道的 `fcp`SCSI (FC)选择 `iscsi。	iscsi 如果为空
formatOptions	使用 `formatOptions`为 `mkfs`命令指定命令行参数，该命令在卷格式化时应用。这允许您根据自己的偏好格式化卷。请务必指定与 mkfs 命令选项类似的 formatOptions，不包括设备路径。示例："-E nodiscard" 支持 **`ontap-san`**和 **`ontap-san-economy`**带有 **iSCSI** 协议的驱动程序。*此外，在使用 iSCSI 和 NVMe/TCP 协议时，支持 ASA r2 系统。*	
limitVolumePoolSize	在LUS-SAN-Economy后端使用ONTAP时可要求的最大FlexVol大小。	"" （默认情况下不强制实施）

参数	Description	Default
`denyNewVolumePools`	限制 `ontap-san-economy`后端创建新的FlexVol卷以包含其LUN。仅会使用已有的FlexVol配置新的PV.	

有关使用**formatOptions**的建议

Trident建议采用以下选项来加快格式化过程：

- **-E nodiscard (ext3, ext4):** 不要尝试在 mkfs 时丢弃块（最初丢弃块对固态设备和稀疏/精简配置存储很有用）。这将取代已弃用的选项“-K”，并且适用于 ext3、ext4 文件系统。
- **-K (xfs):** 不要在执行 mkfs 时尝试丢弃块。此选项适用于 xfs 文件系统。

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

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

步骤

1. 为后端 SVM 配置域名系统 (DNS) 设置：

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

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

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

3. 使用此命令创建 AD 用户或组来管理集群或 SVM

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

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

用于配置卷的后端配置选项

您可以在中使用这些选项控制默认配置 `defaults` 配置部分。有关示例，请参见以下配置示例。

参数	Description	Default
`spaceAllocation`	LUN 的空间分配	"true" 如果指定，则设置为 **`true`**适用于 **ASA r2** 系统。
`s`页面预留	空间预留模式；"无"(精简)或"卷"(厚)。设置为 **`none`** 适用于 **ASA r2** 系统。	"无"
`sSnapshot` 策略	要使用的Snapshot策略。设置为 **`none`**适用于 **ASA r2** 系统。	"无"

参数	Description	Default
qosPolicy	要为创建的卷分配的 QoS 策略组。选择每个存储池 / 后端的 qosPolicy 或 adaptiveQosPolicy 之一。将QoS策略组与Trident结合使用需要使用ONTAP 9™8或更高版本。您应使用非共享QoS策略组、并确保此策略组分别应用于每个成分卷。共享QoS策略组会对所有工作负载的总吞吐量实施上限。	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。选择每个存储池 / 后端的 qosPolicy 或 adaptiveQosPolicy 之一	""
sSnapshot 预留	为快照预留的卷百分比。不要为 **ASA r2** 系统指定。	如果为"0"、则为"0" snapshotPolicy 为"none"、否则为""
splitOnClone	创建克隆时，从其父级拆分该克隆	false
加密	在新卷上启用 NetApp 卷加密 (NVE)；默认为 false。必须在集群上获得许可并启用 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。有关详细信息，请参阅：“Trident如何与NVE和NAE配合使用”。	“false” 如果指定，则设置为 `**true**`适用于 **ASA r2** 系统。
luksEncryption	启用LUKS加密。请参阅 “使用Linux统一密钥设置(LUKS)”。	"" 设置为 `**false**`适用于 **ASA r2** 系统。
分层策略	分层策略使用“无” 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

下面是一个定义了默认值的示例：

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

```
  snapshotPolicy: default
  snapshotReserve: '10'
```

备注	对于使用 `ontap-san`驱动程序创建的所有卷，Trident 会向 FlexVol 额外增加 10% 的容量以容纳 LUN 元数据。LUN 将按照用户在 PVC 中请求的确切大小进行配置。Trident 向 FlexVol 增加 10%（在 ONTAP 中显示为可用大小）。用户现在可以获得其所请求的可用容量。此更改还可防止 LUN 变为只读，除非可用空间已被完全利用。这不适用于 ontap-san-economy。

对于定义的后端 `snapshotReserve`，Trident将按如下所示计算卷的大小：

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

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



目前，调整大小是对现有卷使用新计算的唯一方法。

最低配置示例

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

备注	如果您在NetApp ONTAP上使用Amazon FSx和、NetApp建议您为Trident指定DNS名称、而不是IP地址。

▼ ONTAP SAN示例

这是使用的基本配置 `ontap-san` 驱动程序。

```
---
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

您可以对后端进行配置、以避免在切换和切回后手动更新后端定义 "SVM复制和恢复"。

要进行无缝切换和切回、请使用指定SVM `managementLIF`、并省略这些 `svm`参数。例如：

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

▼ **ONTAP SAN**经济性示例

```
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

在本基本配置示例中 `clientCertificate`， `clientPrivateKey`，和 `trustedCACertificate` (如果使用可信CA、则可选)将填充 `backend.json` 和分别采用客户端证书、专用密钥和可信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

这些示例使用创建后端 `useCHAP` 设置为 `true`。

ONTAP SAN CHAP示例

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

ONTAP SAN经济性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>
```

▼ 均衡放置示例

此示例启用 ONTAP 平衡放置，以便 ONTAP 为新卷选择聚合。均衡放置需要 ONTAP 9.19.1 或更高版本。有关详细信息，请参阅 "使用 ONTAP 后端的均衡放置"。

```
---
version: 1
storageDriverName: ontap-san
backendName: san-balanced-placement
managementLIF: 10.0.0.1
svm: svm_iscsi
```

```
username: vsadmin
password: <password>
useBalancedPlacement: true
```

▼ **NVMe/TCP**示例

您必须在ONTAP后端为SVM配置NVMe。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后端为SVM配置FC。这是FC的基本后端配置。

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

▼ 使用**nameTemplate**的后端配置示例

```
---
version: 1
storageDriverName: ontap-san
backendName: ontap-san-backend
managementLIF: <ip address>
svm: svm0
username: <admin>
password: <password>
defaults:
```

```
  nameTemplate:
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.vo\
    lume.RequestName}}"
labels:
  cluster: ClusterA
  PVC: "{{.volume.Namespace}}_{{.volume.RequestName}}"
```

▼ **formatOptions的ONTAP SAN经济驱动程序示例**

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

虚拟池后端示例

在这些示例后端定义文件中、为所有存储池设置了特定默认值、例如 `spaceReserve` 无、 `spaceAllocation` 为false、和 `encryption` 为false。虚拟池在存储部分中进行定义。

Trident会在"Comments"字段中设置配置标签。在配置时、FlexVol volume Trident会将虚拟池上的所有标签复制到存储卷上、从而设置注释。为了方便起见、存储管理员可以按标签为每个虚拟池和组卷定义标签。

在这些示例中、某些存储池会自行设置 `spaceReserve`， `spaceAllocation`，和 `encryption` 值、而某些池会覆盖默认值。

▼ **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经济性示例**

```
---
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[虚拟池后端示例]。使用 `parameters.selector`字段，每个 StorageClass 指定哪些虚拟池可用于托管卷。卷现在具有所选虚拟池中定义的属性。

- The `protection-gold` StorageClass 映射到 `ontap-san`后端的第一个虚拟池。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"
```

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

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

- 该 `app-mysqldb` StorageClass 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"
```

- The `protection-silver-creditpoints-20k` StorageClass 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 积分的池。

```
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"
```

- The `creditpoints-5k` StorageClass 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"
```

- The `my-test-app-sc` StorageClass 映射到 `ontap-san` 驱动程序中具有 `sanType: nvme` 的 `testAPP` 虚拟池。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 决定选择哪个虚拟池，并确保满足存储要求。

刷新 NVMe 数据 LIF

启用 NVMe 数据 LIF 刷新，以允许已发布的 NVMe 卷使用添加、删除或分配新地址的 LIF，而无需重新创建后端或重新启动 Trident。

概述

当您添加 ONTAP SAN 后端时，Trident 会发现 SVM 的 NVMe 数据 LIF。默认情况下，Trident 会保留该组 LIF，直到您再次添加后端或重新启动 Trident。

将 `enableDataLIFRefresh`设置为 `true`，以便在 Trident 运行时刷新 NVMe 数据 LIF。然后，已发布的 NVMe 卷将使用添加、删除或分配了新地址的 LIF。

此选项是安装参数，而不是后端配置参数。默认情况下，它处于禁用状态。

从 Trident 26.10 开始，NVMe 数据 LIF 刷新可用。

此功能适用于：

- `ontap-san` 采用 `sanType: nvme`统一 ONTAP。
- `ontap-san` 在 ASA r2 上使用 NVMe/TCP。

注意	同时删除或替换所有数据 LIF 会造成中断。如果您计划删除或替换所有数据 LIF，请确保先准备好替换 LIF，以免数据访问中断。

启用 NVMe 数据 LIF 刷新

Trident 运算符

对于新安装，请在 `TridentOrchestrator` spec 中设置 `enableDataLIFRefresh`：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableDataLIFRefresh: true
```

应用清单：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

对于现有安装，请修补 `TridentOrchestrator`：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableDataLIFRefresh":true}}'
```

验证应用值：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableDataLIFRefresh}'
```

掌舵

对于新安装，请设置 Helm 值：

```
helm install trident netapp-trident/trident-operator \
  --namespace trident --create-namespace \
  --set enableDataLIFRefresh=true
```

对于现有的 Helm 安装，请升级版本并启用该值：

```
helm upgrade trident netapp-trident/trident-operator \
  --namespace trident \
  --reuse-values \
  --set enableDataLIFRefresh=true
```

您也可以在 Helm values 文件中设置 `enableDataLIFRefresh: true`，并通过 `-f <values-file>`将其传递给 `helm upgrade`。

Tridentctl

对于新安装，请使用 `--enable-datalif-refresh` 标志：

```
tridentctl install --enable-datalif-refresh -n trident
```

要在现有 `tridentctl`安装上启用此选项，请卸载 Trident 并使用该标志重新安装。卸载前，请遵循"卸载 Trident"中的注意事项。

```
tridentctl uninstall -n trident
```

```
tridentctl install --enable-datalif-refresh -n trident
```

备注 `--enable-datalif-refresh` 标记是安装选项，未显示在 `tridentctl install --help`中。

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提供了以下NAS存储驱动程序来与ONTAP集群进行通信。支持的访问模式包括：*ReadWriteOnce*(RWO)、*ReadOnlyMany*(ROX)、*ReadWriteMany*(rwx)、*ReadWriteOncePod*(RWOP)。

驱动程序	协议	卷模式	支持的访问模式	支持的文件系统
`ontap-NAS`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	""、 `nfs`， `smb`
`ontap-nas-economy.`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	""、 `nfs`， `smb`
`ontap-nas-flexgroup`	NFS SMB	文件系统	Rwo、ROX、rwx、RWOP	""、 `nfs`， `smb`

警告

- 使用 ... `ontap-san-economy` 只有当永久性卷使用量计数预期高于时、才会显示此值 "支持的ONTAP卷限制"。
- 使用 ... `ontap-nas-economy` 只有当永久性卷使用量计数预期高于时、才会显示此值 "支持的ONTAP卷限制" 和 `ontap-san-economy` 无法使用驱动程序。
- 请勿使用 `ontap-nas-economy` 预测数据保护、灾难恢复或移动性的需求。
- ONTAP NAS 驱动程序不支持 ONTAP 卷自动调整大小。有关支持的 SAN 驱动程序和配置详细信息，请参见 "配置 ONTAP 卷自动调整大小"。

用户权限

Trident应以ONTAP或SVM管理员身份运行、通常使用集群用户 `vsadmin`或SVM用户、或者使用 `admin`具有相同角色的其他名称的用户。

对于Amazon FSx for NetApp ONTAP部署、Trident应使用集群用户 `vsadmin`或SVM用户以ONTAP或SVM管理员身份运行、或者使用具有相同角色的其他名称的用户运行 `fsxadmin`。此 `fsxadmin`用户只能有限地替代集群管理员用户。

备注 如果使用 `limitAggregateUsage`参数、则需要集群管理员权限。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中创建一个可以由三端驱动程序使用的限制性更强的角色、但我们不建议这样做。大多数新版本的 Trident 都会调用需要考虑的其他 API ，从而使升级变得困难且容易出错。

准备使用**ONTAP NAS**驱动程序配置后端

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

备注	只有 `ontap-nas`AFX 系统支持 NFS 协议驱动程序；不支持 SMB 协议。

要求

- 对于所有 ONTAP 后端，Trident 要求至少将一个聚合分配给 SVM。
- 您可以运行多个驱动程序，并创建指向其中一个驱动程序的存储类。例如、您可以配置一个使用的黄金类 `ontap-nas` 驱动程序和使用的铜牌类 `ontap-nas-economy` 一个。
- 所有Kubernetes工作节点都必须安装适当的NFS工具。请参见 "此处" 有关详细信息：
- Trident仅支持挂载到Windows节点上运行的Pod的SMB卷。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提供了两种对ONTAP后端进行身份验证的模式。

- 基于凭据：此模式需要对ONTAP后端具有足够的权限。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）（建议）。

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

警告	如果您尝试同时提供*凭据和证书*、则后端创建将失败、并显示一条错误、指出配置文件中提供了多种身份验证方法。

启用基于凭据的身份验证

Trident需要SVM范围/集群范围的管理员的凭据才能与ONTAP后端进行通信。建议使用标准的预定义角色，如 `admin`或 `vsadmin`。这样可以确保与未来ONTAP版本的正向兼容性、这些版本可能会公开未来Trident版本要使用的功能API。可以创建自定义安全登录角色并将其用于Trident、但不建议这样做。

后端定义示例如下所示：

YAML

```yaml
---
version: 1
backendName: ExampleBackend
storageDriverName: ontap-nas
```

```
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
credentials:
  name: secret-backend-creds
```

JSON

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

请注意，后端定义是凭据以纯文本格式存储的唯一位置。创建后端后，用户名 / 密码将使用 Base64 进行编码并存储为 Kubernetes 密钥。创建 / 更新后端是唯一需要了解凭据的步骤。因此，这是一项仅由管理员执行的操作，由 Kubernetes 或存储管理员执行。

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 后端进行通信。后端定义需要三个参数。

- clientCertificate ： 客户端证书的 Base64 编码值。
- clientPrivateKey ： 关联私钥的 Base64 编码值。
- trustedCACertifate ： 受信任 CA 证书的 Base64 编码值。如果使用可信 CA ，则必须提供此参数。如果不使用可信 CA ，则可以忽略此设置。

典型的工作流包括以下步骤。

步骤

1. 生成客户端证书和密钥。生成时，将公用名（ Common Name ， CN ）设置为要作为身份验证的 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. 将可信 CA 证书添加到 ONTAP 集群。此问题可能已由存储管理员处理。如果未使用可信 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（从步骤 1 开始）。

```
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 <SVM 管理 LIF> 和 <SVM 名称 > 替换为管理 LIF IP 和 ONTAP 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其他身份验证方法或轮换其凭据。这两种方式都适用：使用用户名 / 密码的后端可以更新为使用证书；使用证书的后端可以更新为基于用户名 / 密码的后端。为此、您必须删除现有身份验证方法并添加新的身份验证方法。然后、使用更新后的backend.json文件、该文件包含要执行的所需参数 `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 |
+------------+----------------+--------------------------------------
+--------+---------+
```

备注	轮换密码时，存储管理员必须先在 ONTAP 上更新用户的密码。然后进行后端更新。轮换证书时，可以向用户添加多个证书。之后，后端将更新以使用新证书，然后可以从 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，也不会影响在之后建立的卷连接。后端更新成功表示Trident可以与ONTAP后端通信并处理未来的卷操作。

为**Trident**创建自定义**ONTAP**角色

您可以创建Privileges最低的ONTAP集群角色、这样就不必使用ONTAP管理员角色在Trident中执行操作。如果在Trident后端配置中包含用户名、则Trident将使用您创建的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**命令行界面

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系统管理器中执行以下步骤：

1. 创建自定义角色：
   a. 要在集群级别创建自定义角色，请选择*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*用户和角色*旁边的箭头图标(→)。
   c. 在*角色*下选择*+添加*。
   d. 定义角色的规则，然后单击*Save*。
2. 将角色映射到**Trident user**：+在*Users and Roles*页面上执行以下步骤：
   a. 在*用户*下选择添加图标*+*。
   b. 选择所需的用户名，然后在下拉菜单中为*rouser*选择一个角色。
   c. 单击 * 保存 * 。

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导出策略控制对其配置的卷的访问。

使用导出策略时、Trident提供了两个选项：

- Trident可以动态管理导出策略本身；在此操作模式下、存储管理员可以指定一个表示可接受IP地址的CIDR块列表。Trident会在发布时自动将这些范围内的适用节点IP添加到导出策略中。或者、如果未指定CIDR、则在要发布卷的节点上找到的所有全局范围单播IP都将添加到导出策略中。
- 存储管理员可以手动创建导出策略和添加规则。除非在配置中指定了其他导出策略名称、否则Trident将使用默认导出策略。

动态管理导出策略

通过Trident、可以动态管理ONTAP后端的导出策略。这样，存储管理员就可以为工作节点 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；修改导出策略不再需要手动干预存储集群。此外、这还有助于将对存储集群的访问限制为仅限正在挂载卷且IP位于指定范围内的工作节点访问、从而支持精细的自动化管理。

备注	使用动态导出策略时、请勿使用网络地址转换(Network Address Translation、NAT)。使用NAT时、存储控制器会看到前端NAT地址、而不是实际IP主机地址、因此、如果在导出规则中找不到匹配项、则会拒绝访问。

示例

必须使用两个配置选项。下面是一个后端定义示例：

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

备注	使用此功能时、您必须确保SVM中的根接合具有先前创建的导出策略、并具有允许节点CIDR块的导出规则(例如默认导出策略)。始终遵循NetApp建议的最佳实践、将SVM专用于Trident。

以下是使用上述示例对此功能的工作原理进行的说明：

- `autoExportPolicy`设置为 `true`。这表示Trident会为SVM的使用此后端配置的每个卷创建一个导出策略 `svm1`、并使用地址块处理规则的添加和删除 `autoexportCIDRs`。在将卷连接到节点之前、此卷会使用一个空导出策略、此策略不带任何规则来防止对该卷进行不必要的访问。将卷发布到节点后、Trident会创建一个与指定CIDR块中包含节点IP的底层qtree同名的导出策略。这些IP也会添加到父FlexVol volume使用的导出策略中
  - 例如：
    - 后端UUID 403b5326/8482-40db-96d0-d83fb3f4daec
    - `autoExportPolicy`将设置为 `true`
    - 存储前缀 `trident`
    - pvc UUID a79bcf5f-7b6d-4a40-9876- e2551f159c1c
    - 名为svm_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c的qtree会为名为的FlexVol创建一个导出策略、为名为的qtree创建一个导出策略、`trident_pvc_a79bcf5f_7b6d_4a40_9876_e2551f159c1c`并在`Trident`上创建 `trident-403b5326-8482-40db96d0-d83fb3f4daec`一个名为的空导出策略 `trident_empty`。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。空导出策略将由所有未附加的卷重复使用。
- `autoExportCIDRs`包含地址块列表。此字段为可选字段，默认为 "0.0.0.0/0 ， " ： ： /0" 。如果未定义、则Trident会添加在具有出版物的工作节点上找到的所有全局范围单播地址。

在此示例中、 `192.168.0.0/24`提供了地址空间。这表示属于此地址范围且发布内容的`Kub`联网 节点`IP`将添加到`Trident`创建的导出策略中。当`Trident`注册运行该功能的节点时，它将检索该节点的`IP`地址，并根据中提供的地址块对其进行检查 `autoExportCIDRs`。发布时，在筛选IP之后，Trident将为要发布到的节点的客户端IP创建导出策略规则。

创建后，您可以为后端更新 `autosExportPolicy` 和 `autosExportCIDR` 。您可以为自动管理的后端附加新的 CIDR ，也可以删除现有的 CIDR 。删除 CIDR 时请务必小心，以确保现有连接不会断开。您也可以选择对后端禁用 `autosExportPolicy` ，并回退到手动创建的导出策略。这需要在后端配置中设置 `exportPolicy` 参数。

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

```
./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
```

删除节点后、Trident会检查所有导出策略以删除与该节点对应的访问规则。通过从受管后端的导出策略中删除此节点IP、Trident可防止恶意挂载、除非集群中的新节点重复使用此IP。

对于以前存在的后端、使用更新后端 `tridentctl update backend`可确保Trident自动管理导出策略。这样会根据需要创建两个新的导出策略、并以后端的UUID和qtree名称命名。后端上的卷在卸载并重新挂载后将使用新创建的导出策略。

备注	删除具有自动管理导出策略的后端将删除动态创建的导出策略。如果重新创建后端，则会将其视为新的后端，并会创建新的导出策略。

如果更新了活动节点的IP地址、则必须在此节点上重新启动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

只需稍作准备、您就可以使用配置SMB卷 `ontap-nas` 驱动程序。

警告	您必须在SVM上同时配置NFS和SMB/CCIFS协议、才能为ONTAP内部集群创建 `ontap-nas-economy`SMB卷。如果未能配置其中任一协议、则发生原因 SMB卷创建将失败。

备注 `autoExportPolicy`SMB卷不支持。

开始之前

在配置SMB卷之前、您必须满足以下条件。

- 一个Kubernetes集群、其中包含一个Linux控制器节点以及至少一个运行Windows Server 2022的Windows工作节点。Trident仅支持挂载到Windows节点上运行的Pod的SMB卷。
- 至少一个包含Active Directory凭据的Trident密钥。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。配置 `csi-proxy`、请参见 "GitHub：CSI代理" 或 "GitHub：适用于Windows的CSI代理" 适用于在Windows上运行的Kubernetes节点。

步骤

1. 对于内部ONTAP、您可以选择创建SMB共享、也可以选择Trident为您创建一个共享。

   备注 Amazon FSx for ONTAP需要SMB共享。

   您可以使用以下两种方式之一创建SMB管理共享 "Microsoft管理控制台" 共享文件夹管理单元或使用ONTAP命令行界面。要使用ONTAP 命令行界面创建SMB共享、请执行以下操作：

   a. 如有必要，为共享创建目录路径结构。

      。 `vserver cifs share create` 命令会在创建共享期间检查-path选项中指定的路径。如果指定路径不存在，则命令将失败。

   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. 验证是否已创建共享：

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

   备注 请参见 "创建 SMB 共享" 了解完整详细信息。

2. 创建后端时、必须配置以下内容以指定SMB卷。有关适用于ONTAP 后端的所有FSX配置选项、请参见 "适用于ONTAP 的FSX配置选项和示例"。

参数	Description	示例
smbShare	您可以指定以下选项之一：使用Microsoft管理控制台或ONTAP命令行界面创建的SMB共享的名称；允许Trident创建SMB共享的名称；或者、您可以将参数留空以防止对卷进行通用共享访问。对于内部ONTAP、此参数是可选的。此参数对于Amazon FSx for ONTAP后端为必填项、不能为空。	smb-share
nasType	*必须设置为 smb`如果为空、则默认为 `nfs。	smb
securityStyle	新卷的安全模式。必须设置为 **ntfs** 或 **mixed** 用于**SMB**卷。	ntfs 或 mixed 对于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 支持使用以下方式创建的 SMB 卷的安全配置 `ontap-nas`和 `ontap-nas-economy`后端。启用安全 SMB 后，您可以使用访问控制列表 (ACL) 为 Active Directory (AD) 用户和用户组提供对 SMB 共享的受控访问。

需要记住的要点

- 输入 `ontap-nas-economy`不支持卷。
- 仅支持只读克隆 `ontap-nas-economy`卷。
- 如果启用了安全 SMB，Trident 将忽略后端提到的 SMB 共享。
- 更新 PVC 注释、存储类注释和后端字段不会更新 SMB 共享 ACL。
- 克隆 PVC 注释中指定的 SMB 共享 ACL 将优先于源 PVC 中的 ACL。
- 启用安全 SMB 时，请确保提供有效的 AD 用户。无效用户将不会被添加到 ACL。
- 如果您在后端、存储类和 PVC 中为同一个 AD 用户提供不同的权限，则权限优先级为：PVC、存储类、后端。
- 安全 SMB 支持 `ontap-nas`托管卷导入，不适用于非托管卷导入。

步骤

1. 在 TridentBackendConfig 中指定 adAdminUser，如以下示例所示：

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

```
  defaults:
    adAdminUser: tridentADtest
  credentials:
    name: backend-tbc-ontap-invest-secret
```

2. 在存储类中添加注释。

   添加 `trident.netapp.io/smbShareAdUser`注释到存储类，以启用安全 SMB 而不会失败。为注释指定的用户值 `trident.netapp.io/smbShareAdUser`应该与 `smbcreds`秘密。您可以选择以下之一 `smbShareAdUserPermission： full_control ， change ， 或者 read 。默认权限是 full_control 。

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

1. 创建 PVC。

   以下示例创建 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc4
  namespace: trident
  annotations:
    trident.netapp.io/snapshotDirectory: "true"
    trident.netapp.io/smbShareAccessControl: |
      read:
        - tridentADtest
spec:
  accessModes:
    - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-smb-sc
```

ONTAP NAS配置选项和示例

创建并使用 ONTAP NAS 驱动程序以配合您的 Trident 安装。本节提供后端配置示例以及将后端映射到 StorageClasses 的详细信息。从 25.10 版本开始，NetApp Trident 支持"NetApp AFX 存储系统"。NetApp AFX 存储系统在存储层的实现方面与其他基于 ONTAP 的系统（ASA、AFF 和 FAS）有所不同。

备注	只有 `ontap-nas`NetApp AFX 系统支持 NFS 协议驱动程序；不支持 SMB 协议。

后端配置选项

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

从 26.06 版本开始，您可以启用 ONTAP 平衡放置，以便 ONTAP 为新卷选择聚合。在 ONTAP 9.19.1 或更高版本上的 AFX 存储系统需要平衡放置。有关要求和示例，请参阅 "使用 ONTAP 后端的均衡放置"。

下表显示了后端配置选项：

参数	Description	Default
ve版本		始终为 1
storageDriverName	存储驱动程序的名称。 创建后无法修改。要更新此参数，您必须创建一个新的后端。 备注 仅适用于NetApp AFX 系统 `ontap-nas` 已支持。	ontap-nas ontap-nas-economy`或 `ontap-nas-flexgroup
backendName	自定义名称或存储后端	驱动程序名称+"_"+ dataLIF
m年 月 日	集群或SVM管理LIF的IP地址可以指定完全限定域名(FQDN)。如果Trident是使用IPv6标志安装的、则可以设置为使用IPv6地址。IPv6地址必须用方括号定义，例如 [28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]。有关无缝MetroCluster切换的信息，请参见MetroCluster示例。	"10.0.0.1"、"[2001：1234：abc：：：fefe]"

参数	Description	Default
`dataLIF`	协议 LIF 的 IP 地址。NetApp建议指定 `dataLIF`。如果未提供此参数、则Trident将从SVM提取数据LIF。您可以指定要用于NFS挂载操作的完全限定域名(FQDN)、以便创建轮叫DNS来在多个dataLIF之间进行负载平衡。可以在初始设置后更改。请参阅 。如果Trident是使用IPv6标志安装的、则可以设置为使用IPv6地址。IPv6地址必须用方括号定义，例如 `[28e8:d9fb:a825:b7bf:69a8:d02f:9e7b:3555]`。*省略MetroCluster。*请参见MetroCluster示例。	指定的地址或派生自SVM (如果未指定)(不建议)
`sVM`	要使用的 Storage Virtual Machine *对于MetroCluster省略。*请参见 MetroCluster示例。	如果指定了 SVM `managementLIF`，则派生
`autosExportPolicy`	启用自动创建和更新导出策略[布尔值]。使用 `autoExportPolicy`和 `autoExportCIDRs`选项、Trident可以自动管理导出策略。	false
`autosExportCIDR`	用于筛选KubeNet节点IP的CIDR列表(启用时)。`autoExportPolicy`使用 `autoExportPolicy`和 `autoExportCIDRs`选项、Trident可以自动管理导出策略。	["0.0.0.0/0、"：：：/0"]`
标签	要应用于卷的一组任意 JSON 格式的标签	""
客户端证书	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
`clientPrivateKey`	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""
`trustedCACertifate`	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证	""
用户名	用于连接到集群/SVM 的用户名。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	
密码	连接到集群/SVM 的密码。用于基于凭据的身份验证。有关 Active Directory 身份验证，请参阅 "使用 Active Directory 凭据向后端 SVM 验证Trident 的身份"。	
s存储前缀	在 SVM 中配置新卷时使用的前缀。设置后无法更新 备注 当使用 ontap-nas-economy 和包含 24 个或更多字符的 storagePrefix 时，qtree 没有嵌入存储前缀，尽管它位于卷名称中。	"三级联"

参数	Description	Default
聚合	用于配置的聚合（可选；如果设置，则必须分配给SVM）。对于 `ontap-nas-flexgroup`驱动程序，此选项将被忽略。如果未分配，则可以使用任何可用的聚合来配置 FlexGroup 卷。 备注：当聚合在 SVM 中更新时，它会通过轮询 SVM 自动在 Trident 中更新，而无需重启 Trident Controller。当您在 Trident 中配置了特定聚合以配置卷时，如果该聚合被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将在 Trident 中移至失败状态。您必须将聚合更改为 SVM 上存在的聚合，或将其完全删除，以使后端恢复联机。 请勿指定用于 **AFX** 存储系统。 当 **`useBalancedPlacement`**`设置为 **`true`**`时，请勿指定。	""
`useBalancedPlacement`	将新卷的聚合选择委托给 ONTAP [Boolean]。需要 ONTAP 9.19.1 或更高版本，并将 `useREST`设置为 `true`。不能与 `aggregate`结合使用。 有关要求和示例，请参见 "使用 ONTAP 后端的均衡放置"。 如果指定，对于 **ONTAP 9.19.1** 或更高版本上的 **AFX** 存储系统，始终设置为 **`true`**。	`false`； `true` 适用于 ONTAP 9.19.1 或更高版本上的 AFF 存储系统
`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**。请勿指定用于 **AFX** 存储系统。	"" （默认情况下不强制实施）
FlexgroupGroupGroupRegateList	要配置的聚合列表(可选；如果已设置、则必须将其分配给SVM)。分配给SVM的所有聚合均用于配置FlexGroup卷。支持* ONTAP－NAS－FlexGroup－Storage驱动程序。 备注：在 SVM 中更新聚合列表时，Trident 会通过轮询 SVM 自动更新列表，而无需重新启动 Trident Controller。当您在 Trident 中配置了特定聚合列表以配置卷时，如果聚合列表被重命名或移出 SVM，则在轮询 SVM 聚合时，后端将在 Trident 中移至失败状态。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`	如果请求的卷大小大于此值，则配置失败。	""(默认情况下不强制实施)

参数	Description	Default
`debugTraceFlags`	故障排除时要使用的调试标志。例如、 {"api"：false、"METHO"：true} 请勿使用 `debugTraceFlags` 除非您正在进行故障排除并需要详细的日志转储。	空
`nasType`	配置 NFS 或 SMB 卷的创建。选项有 `nfs`， `smb`或空值。设置为 null 则默认使用 NFS 卷。如果指定，则始终设置为 **`nfs`**适用于 **AFX** 存储系统。	`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，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 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 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管理控制台或ONTAP命令行界面创建的SMB共享的名称；允许Trident创建SMB共享的名称；或者、您可以将参数留空以防止对卷进行通用共享访问。对于内部ONTAP、此参数是可选的。此参数对于Amazon FSx for ONTAP后端为必填项、不能为空。	`smb-share`
`useREST`	使用ONTAP REST API 的布尔参数。 `useREST`设置为 `true`Trident使用ONTAP REST API 与后端通信；当设置为 `false`Trident使用 ONTAPI (ZAPI) 调用与后端通信。此功能需要ONTAP 9.11.1 及更高版本。此外，所使用的ONTAP登录角色必须具有访问权限。 `ontapi`应用。预定义项满足了这一点。 `vsadmin`和 `cluster-admin`角色。从Trident 24.06 版本和ONTAP 9.15.1 或更高版本开始， `useREST`设置为 `true`默认；更改 `useREST`到 `false`使用 ONTAPI (ZAPI) 调用。如果指定，则始终设置为 **`true`**适用于 **AFX** 存储系统。	`true` 对于ONTAP 9.151或更高版本，否则 `false`。
`limitVolumePoolSize`	在qtree-NAS ONTAP经济型后端使用qtrees时可请求的最大FlexVol大小。	"" （默认情况下不强制实施）
`denyNewVolumePools`	限制 `ontap-nas-economy`后端创建新的FlexVol卷以包含其qtrees。仅会使用已有的FlexVol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。使用此参数可为 SMB 共享提供具有完全控制权的管理员权限。	

用于配置卷的后端配置选项

您可以在中使用这些选项控制默认配置 `defaults` 配置部分。有关示例，请参见以下配置示例。

参数	Description	Default
spaceAllocation	qtrees的空间分配	"正确"
s页面预留	空间预留模式；"无"(精简)或"卷"(厚)	"无"
sSnapshot 策略	要使用的 Snapshot 策略	"无"
qosPolicy	要为创建的卷分配的 QoS 策略组。选择每个存储池 / 后端的 qosPolicy 或 adaptiveQosPolicy 之一	""
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。选择每个存储池 / 后端的 qosPolicy 或 adaptiveQosPolicy 之一。不受 ontap-nas-economy.	""
sSnapshot 预留	为快照预留的卷百分比	如果为"0"、则为"0" snapshotPolicy 为"none"、否则为""
splitOnClone	创建克隆时，从其父级拆分该克隆	false
加密	在新卷上启用 NetApp 卷加密 (NVE)；默认为 false。必须在集群上获得许可并启用 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。有关详细信息，请参阅："Trident如何与NVE和NAE配合使用"。	false
分层策略	使用"无"的层策略	
unixPermissions	新卷的模式	"777"表示NFS卷；空(不适用)表示SMB卷
snapshotDir	控制对的访问 .snapshot 目录	true, false (显式设置)。
exportPolicy	要使用的导出策略	default
securityStyle	新卷的安全模式。NFS支持 mixed 和 unix 安全模式。SMB支持 mixed 和 ntfs 安全模式。	NFS默认值为 unix。SMB默认值为 ntfs。
nameTemplate	用于创建自定义卷名称的模板。	""

备注	将QoS策略组与Trident结合使用需要使用ONTAP 9™8或更高版本。您应使用非共享QoS策略组、并确保此策略组分别应用于每个成分卷。共享QoS策略组会对所有工作负载的总吞吐量实施上限。

卷配置示例

下面是一个定义了默认值的示例：

```
---
version: 1
storageDriverName: ontap-nas
backendName: customBackendName
managementLIF: 10.0.0.1
```

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

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

计算方法如下：

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

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



升级 Trident 时，先前安装的现有后端将按上述方式配置卷。对于升级前创建的卷，应调整其卷的大小，以便观察更改。例如，使用 `snapshotReserve=50`早期版本的 2 GiB PVC 所创建的卷仅提供 1 GiB 的可写空间。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。

最低配置示例

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

备注	如果在采用 Trident 的 NetApp ONTAP 上使用 Amazon FSx ，建议为 LIF 指定 DNS 名称，而不是 IP 地址。

▼ **ONTAP NAS**经济性示例

```
---
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

您可以对后端进行配置、以避免在切换和切回后手动更新后端定义 "SVM复制和恢复"。

要进行无缝切换和切回、请使用指定SVM `managementLIF` 并省略 `dataLIF` 和 `svm` parameters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

这是一个最低后端配置示例。 `clientCertificate`， `clientPrivateKey`，和 `trustedCACertificate` (如果使用可信CA、则可选)将填充 `backend.json` 和分别采用客户端证书、专用密钥和可信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

此示例介绍如何指示Trident使用动态导出策略自动创建和管理导出策略。这对于和 `ontap-nas-flexgroup`驱动程序是相同的 `ontap-nas-economy`。

```
---
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
```

▼ 均衡放置示例

此示例启用 ONTAP 平衡放置，以便 ONTAP 为新卷选择聚合。均衡放置需要 ONTAP 9.19.1 或更高版本。有关详细信息，请参阅 "使用 ONTAP 后端的均衡放置"。

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-balanced-placement
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: svm_nfs
username: vsadmin
password: password
useBalancedPlacement: true
```

▼ **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
```

▼ **Amazon FSx for ONTAP**使用**SMB**卷示例

- `smbShare` 使用SMB卷的FSx for ONTAP需要参数。

```
---
version: 1
backendName: SMBBackend
storageDriverName: ontap-nas
managementLIF: example.mgmt.fqdn.aws.com
nasType: smb
```

```
dataLIF: 10.0.0.15
svm: nfs_svm
smbShare: smb-share
clientCertificate: ZXR0ZXJwYXB...ICMgJ3BhcGVyc2
clientPrivateKey: vciwKIyAgZG...0cnksIGRlc2NyaX
trustedCACertificate: zcyBbaG...b3Igb3duIGNsYXNz
storagePrefix: myPrefix_
```

▼ 使用**nameTemplate**的后端配置示例

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

虚拟池后端示例

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默认值、例如 `spaceReserve` 无、`spaceAllocation` 为false、和 `encryption` 为false。虚拟池在存储部分中进行定义。

Trident会在"Comments"字段中设置配置标签。注释在FlexVol for或FlexGroup `ontap-nas-flexgroup` for上设置 `ontap-nas`。配置时、Trident会将虚拟池上的所有标签复制到存储卷。为了方便起见、存储管理员可以按标签为每个虚拟池和组卷定义标签。

在这些示例中、某些存储池会自行设置 `spaceReserve`， `spaceAllocation`，和 `encryption` 值、而某些池会覆盖默认值。

▼ **ONTAP NAS**示例

```
---
version: 1
storageDriverName: ontap-nas
managementLIF: 10.0.0.1
svm: svm_nfs
username: admin
password: <password>
```

```
nfsMountOptions: nfsvers=4
defaults:
  spaceReserve: none
  encryption: "false"
  qosPolicy: standard
labels:
  store: nas_store
  k8scluster: prod-cluster-1
region: us_east_1
storage:
  - labels:
      app: msoffice
      cost: "100"
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: "true"
      unixPermissions: "0755"
      adaptiveQosPolicy: adaptive-premium
  - labels:
      app: slack
      cost: "75"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      department: legal
      creditpoints: "5000"
    zone: us_east_1b
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0755"
  - labels:
      app: wordpress
      cost: "50"
    zone: us_east_1c
    defaults:
      spaceReserve: none
      encryption: "true"
      unixPermissions: "0775"
  - labels:
      app: mysqldb
      cost: "25"
```

```
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

▼ **ONTAP NAS FlexGroup示例**

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

```
      unixPermissions: "0775"
  - labels:
      protection: bronze
      creditpoints: "10000"
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: "false"
      unixPermissions: "0775"
```

▼ **ONTAP NAS经济性示例**

```
---
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 定义请参阅 [虚拟池后端示例]。使用 `parameters.selector` 字段，每个 StorageClass 指定可用于托管卷的虚拟池。卷具有所选虚拟池中定义的属性。

- The `protection-gold` StorageClass 映射到 `ontap-nas-flexgroup`后端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"
```

- The `protection-not-gold` StorageClass 映射到 `ontap-nas-flexgroup`后端的第三和第四个虚拟池。这些是唯一提供黄金级别以外保护级别的池。

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

- 该 `app-mysqldb` StorageClass 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"
```

- The `protection-silver-creditpoints-20k` StorageClass 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 积分的池。

```
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"
```

- The `creditpoints-5k` StorageClass 映射到 `ontap-nas`后端的第三个虚拟池和 `ontap-nas-economy`后端的第二个虚拟池。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 决定选择哪个虚拟池，并确保满足存储要求。

更新 `dataLIF` 初始配置后

您可以在初始配置后更改dataLIF、方法是运行以下命令、为新的后端JSON文件提供更新后的dataLIF。

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

备注	如果一个或多个Pod连接了PVC、则必须先关闭所有相应的Pod、然后再将其重新启动、以使新的dataLIF生效。

安全 **SMB** 示例

使用 **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
```

具有存储池的后端配置

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

采用 **ontap-nas** 驱动程序的存储类示例

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

备注	确保添加 `annotations`启用安全 SMB。如果没有注释，安全 SMB 就无法工作，无论后端或 PVC 中设置了什么配置。

采用 **ontap-nas-economy** 驱动程序的存储类示例

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

具有单个 **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
```

使用 **ONTAP** 后端的均衡放置

您可以在 ONTAP 后端上启用 ONTAP 平衡放置。通过平衡放置，ONTAP 为每个新卷选择最佳聚合，因此您无需在 Trident 中管理聚合分配。

均衡放置的工作原理

默认情况下，Trident 会发现分配给 SVM 的聚合，将每个聚合视为存储池，并在配置卷时选择聚合。

启用均衡放置时：

- Trident 将整个 SVM 报告为单个存储池，而不是报告单个聚合。
- ONTAP 为每个新卷选择聚合，并在可用聚合之间平衡卷。
- Trident 使用 ONTAP `/application/containers` REST API 创建卷。

均衡放置可用于统一 ONTAP 系统（AFF、FAS 和 ASA）和 NetApp AFX 存储系统。

要求

- 后端必须使用 ONTAP 9.19.1 或更高版本。
- Trident 必须使用 REST 与 ONTAP 通信。ZAPI 不支持均衡放置。

- 后端配置不得指定聚合。不能将平衡放置与显式聚合选择相结合。

默认值

`useBalancedPlacement` 的默认值取决于存储系统：

存储系统	Default	说明
Unified ONTAP（AFF、FAS、ASA）	`false`	除非您启用均衡放置，否则 Trident 会自动选择聚合。
NetApp AFX（ONTAP 9.19.1 或更高版本）	`true`	均衡放置为必填项。将 `useBalancedPlacement`设置为 `false`会导致后端验证失败。
NetApp AFX（早于 ONTAP 9.19.1）	`false`	平衡放置不可用。

配置选项

参数	Description	Default
`useBalancedPlacement`	将新卷的聚合选择委托给 ONTAP。在后端配置的根目录或单个虚拟池上设置此选项。	`false`; `true` 适用于 ONTAP 9.19.1 或更高版本的 AFX

使用虚拟池的均衡放置

无论是否使用虚拟池，均衡放置均可正常工作：

- 如果没有虚拟池，Trident 会将 SVM 报告为单个存储池，并使用平衡放置创建所有新卷。
- 使用虚拟池，Trident 会照常报告虚拟池并将卷与其匹配。如果匹配的虚拟池启用了平衡放置，则 ONTAP 会为卷选择聚合。

在后端配置的根目录设置 `useBalancedPlacement`可将其应用于所有虚拟池，或在单个虚拟池上进行设置以选择性地启用它。

示例

以下示例在统一的 ONTAP 后端上实现平衡放置：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-balanced-placement
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: cluster-admin
password: <password>
```

```
useBalancedPlacement: true
```

以下示例使用 `TridentBackendConfig`实现平衡放置：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
spec:
  version: 1
  storageDriverName: ontap-nas
  backendName: nas-balanced-placement
  managementLIF: 10.0.0.1
  svm: trident_svm
  useBalancedPlacement: true
  credentials:
    name: backend-tbc-ontap-nas-secret
```

以下示例仅在 `gold`虚拟池上启用平衡放置。Trident 继续为与 `bronze`虚拟池匹配的卷选择聚合：

```
---
version: 1
storageDriverName: ontap-nas
backendName: nas-virtual-pools
managementLIF: 10.0.0.1
svm: trident_svm
username: cluster-admin
password: <password>
storage:
- labels:
    performance: gold
  useBalancedPlacement: true
- labels:
    performance: bronze
```

后端验证错误

在以下情况下，后端创建或更新失败：

配置	结果
`useBalancedPlacement: true` 早于 9.19.1 的 ONTAP	验证失败。升级到 ONTAP 9.19.1 或更高版本。
`useBalancedPlacement: true` 在后端使用 ZAPI	验证失败。配置后端以使用 REST。

配置	结果
`useBalancedPlacement: true` 结合特定聚合	验证失败。删除聚合配置或禁用平衡放置。
`useBalancedPlacement: false` 在使用 ONTAP 9.19.1 或更高版本的 AFX 上	验证失败。ONTAP 9.19.1 或更高版本上的 AFX 需要平衡放置。

适用于 NetApp ONTAP 的 Amazon FSX

将Trident与Amazon FSx for NetApp ONTAP结合使用

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

要求

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

- 安装了 `kubectl`的现有自托管 Kubernetes 集群或 Amazon EKS 集群，或者安装了 `oc`的自托管 OpenShift 集群。
- 可从集群的工作节点访问的现有Amazon FSx for NetApp ONTAP文件系统和Storage Virtual Machine (SVM)。
- 为准备工作的工作节点 "NFS或iSCSI"。

备注	确保按照Amazon Linux和Ubuntu所需的节点准备步骤进行操作 "Amazon Machine 映像" （ AMIS ），具体取决于您的 EKS AMI 类型。

注意事项

- SMB卷：
  - SMB卷支持使用 `ontap-nas` 仅限驱动程序。
  - 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"。
- 自我管理的 OpenShift 集群：
  - 对于推荐的 Trident 集成（使用 AWS Secrets Manager secret 作为 FSx for ONTAP 凭据），OpenShift 上的 Trident 控制器 Pod 必须在将 `hostNetwork`设置为 `true`的情况下运行。
- 在Trident 24.02之前的版本中、Trident无法删除在已启用自动备份的Amazon FSx文件系统上创建的卷。要在Trident 24.02或更高版本中防止此问题，请在AWS FSx for ONTAP的后端配置文件中指定 `fsxFilesystemID`、AWS、 `apikey`AWS `apiRegion`和AWS `secretKey`。

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

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

如果您计划将 ontap-san 驱动程序（例如 iSCSI）与 AWS（EKS、ROSA、EC2 或任何其他实例）一起使用，则节点上所需的多路径配置可能会与 Amazon Elastic Block Store (EBS) CSI 驱动程序冲突。为了确保多路径功能不会干扰同一节点上的 EBS 磁盘，您需要在多路径设置中排除 EBS。这个例子展示了 `multipath.conf`包含所需Trident设置的文件，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提供两种身份验证模式。

- 基于凭据(建议)：将凭据安全地存储在AWS机密管理器中。您可以使用文件系统的用户、也可以使用 `fsxadmin` `vsadmin` 为SVM配置的用户。

警告	Trident应以SVM用户身份运行、或者以具有相同角色的其他名称的用户身份运行 `vsadmin` 。Amazon FSx for NetApp ONTAP的某个 `fsxadmin`用户只能有限地替代ONTAP `admin`集群用户。强烈建议将与Trident结合使用 `vsadmin`。

- 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、请参阅适用于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"

测试过的**Amazon**计算机映像**(AMI)**

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计算机映像(AMI)。以下 AMI 已通过 NetApp Trident 25.02 测试。

AMI	NAS	NAS经济型	iSCSI	iSCSI经济型
AL2023_x86_64_STANDARD	是的。	是的。	是的。	是的。

AL2_x86_64	是的。	是的。	是 *	是 *
BOTTLEROCKET_x86_64	是**	是的。	不适用	不适用
AL2023_ARM_64_STANDARD	是的。	是的。	是的。	是的。
AL2_ARM_64	是的。	是的。	是 *	是 *
BOTTLEROCKET_ARM_64	是**	是的。	不适用	不适用

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

备注	如果此处未列出您所需的AMI、并不表示它不受支持、而只是表示它尚未经过测试。此列表可作为已知可运行的 AMI 的指南。

使用以下项执行的测试：

- EKS版本：1.32
- 安装方法：Helm 25.06 和 AWS 附加组件 25.06
- 对于NAS、我们同时测试了NFSv3和NFSv4.1。
- 对于SAN、测试的是仅iSCSI、而不是NVMe-oF。

执行的测试：

- 创建：存储类、PVC、POD
- 删除：POD、PVC (常规、qtree/LUN—经济型、NAS与AWS备份)

了解更多信息

- "Amazon FSX for NetApp ONTAP 文档"
- "有关适用于 NetApp ONTAP 的 Amazon FSX 的博客文章"

创建 **IAM** 角色和 **AWS Secrets Manager** 机密

您可以通过作为AWS IAM角色进行身份验证(而不是提供显式AWS凭据)来配置Kubbernetes Pod以访问AWS资源。

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

创建**AWS**机密管理器密钥

Trident 需要凭据来管理 FSx for ONTAP SVM 上的存储。将 `vsadmin`凭据存储在 AWS Secrets Manager 密钥中。

以下示例将创建一个AWS机密管理器密钥来存储Trident CSI凭据：

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

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来为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 Trident CSI access to FSx for ONTAP
and AWS Secrets Manager" \
  --query Policy.Arn --output text
```

该命令返回策略 ARN。在以下命令中使用它。

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 **Pod** 身份创建 **IAM** 角色或为服务帐户（**IRSA**）创建 **IAM** 角色

您可以配置 Kubernetes 服务帐户，以通过 EKS Pod Identity 或服务帐户的 IAM 角色（IRSA）代入 AWS Identity and Access Management（IAM）角色。然后，配置为使用该服务帐户的 Pod 即可访问该角色获准访问的 AWS 服务。

Pod 身份（**EKS**）

Amazon EKS Pod Identity 关联提供了管理应用程序凭证的能力，类似于 Amazon EC2 实例配置文件向 Amazon EC2 实例提供凭证的方式。

在您的 **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 身份：

```
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 文件，使 EKS 服务主体能够承担 Pod Identity 的此角色。然后创建一个具有以下信任策略的角色：

```
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创建的 IAM 角色：

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

创建 **pod** 身份关联：

在 IAM 角色和 Trident 服务帐户之间创建 Pod 身份关联 (`trident-controller`）。

```
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
```

服务帐户的 **IAM** 角色（**IRSA**）（仅限 **EKS**）
使用 **AWS CLI**：

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

信任关系**.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"
        }
      }
    }
  ]
}
```

更新文件中的以下值 `trust-relationship.json`：

- **<account_id>** - 您的 AWS 账户 ID
- **<oidc_provider>** - EKS 集群的 OIDC 提供程序 URL。您可以通过运行以下命令来获取：

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

将 **IAM** 策略附加到 **IAM** 角色：

创建角色后、使用以下命令将在上述步骤中创建的策略附加到此角色：

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

验证 **OIDC** 提供者关联：

运行以下命令，验证 OIDC 提供程序是否与集群关联：

```
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与集群关联：

```
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
```

自我管理的 **OpenShift** 的 **IAM** 角色

自我管理的 OpenShift 部署会创建 IAM 角色并将其附加到集群节点。将 Trident IAM 策略附加到分配给每个节点的角色。

要查找集群中节点的实例 ID，请运行以下命令：

```
oc get nodes -o jsonpath='{range
.items[*]}{.spec.providerID}{"\n"}{end}'
```

要查找与每个实例关联的 IAM 角色，请为每个实例运行以下命令：

```
aws ec2 describe-instances --instance-ids <instance-id> --query
'Reservations[*].Instances[*].IamInstanceProfile.Arn' --output text
```

通过运行以下命令将 IAM 策略附加到每个角色：

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

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：

- 掌舵
- 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" 。

通过舵安装**Trident**

Pod 身份（**EKS**）

1. 添加Trident Helm存储库：

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

2. 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、应用程序版本和修订版号。

```
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
```

服务帐户的 **IAM** 角色 **(IRSA) (EKS)**

1. 添加Trident Helm存储库：

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

2. 设置“云提供商”和“云身份”的值：

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

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

```
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
```

自我管理的 **OpenShift** 的 **IAM** 角色

1. 添加Trident Helm存储库：

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

2. 使用以下示例安装 Trident：

```
helm install trident-operator netapp-trident/trident-operator
--version 100.2606.1 --namespace trident --set hostNetwork=true
--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、应用程序版本和修订版号。

```
helm list -n trident
```

```
NAME                NAMESPACE   REVISION   UPDATED
STATUS     CHART                          APP VERSION
trident-operator   trident     1           2026-11-10
16:58:16.108649912 -0600 CST  deployed   trident-operator-100.2606.1
26.06.1
```

备注	当启用 `hostNetwork` 时，自我管理的 OpenShift 安装程序不会创建 Trident 在 TCP 端口 8443 上所需的安全组规则。将 TCP 端口 8443 的入站规则从 VPC CIDR 添加到由 OpenShift 安装程序创建的 `<cluster_name>-<random_string>-node` 和 `<cluster_name>-<random_string>-controlplane` 安全组。

备注	如果您计划使用 iSCSI，请确保在客户端计算机上启用了 iSCSI。如果您使用的是 AL2023 工作

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插件安装Trident

Trident EKS加载项包括最新的安全修补程序和错误修复、并已通过AWS验证、可与Amazon EKS配合使用。通过EKS加载项、您可以始终确保Amazon EKS集群安全稳定、并减少安装、配置和更新加载项所需的工作量。

前提条件

在配置适用于AWS EKS的Trident加载项之前、请确保满足以下条件：

- 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

启用适用于**AWS**的**Trident**加载项

管理控制台

1. 打开Amazon EKS控制台，网址为 https://console.aws.amazon.com/eks/home#/clusters。
2. 在左侧导航窗格中，选择*群集*。
3. 选择要为其配置NetApp Trident CSI加载项的集群的名称。
4. 选择*Add-ones*，然后选择*Get more add-ones*。
5. 请按照以下步骤选择附加组件：
   a. 向下滚动到 **AWS Marketplace** 附加组件 部分，然后在搜索框中输入 **“Trident”**。
   b. 选中 NetApp 的 Trident 框右上角的复选框。
   c. 选择 * 下一步 * 。
6. 在“*配置选定的附加项*设置”页面上，执行以下操作：

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

   a. 选择要使用的*版本*。
   b. 如果您使用 IRSA 身份验证，请确保设置可选配置设置中可用的配置值：

- 选择要使用的*版本*。
- 按照 附加组件配置模式，将 配置值 部分中的 **configurationValues** 参数设置为您在上一步中创建的角色 ARN（值应采用以下格式）：

```
{

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

}
```

+
如果您为冲突解决方法选择覆盖、则可以使用Amazon EKS附加设置覆盖现有附加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解决冲突。在选择此选项之前、请确保Amazon EKS附加组件未管理您需要自行管理的设置。

7. 选择“下一步”。
8. 在*Review and add*页上，选择*Cree*。

   加载项安装完成后、您将看到已安装的加载项。

AWS命令行界面

1.创建 `**add-on.json**`文件：

对于 **Pod Identity**，请使用以下格式：

备注	使用

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

对于 **IRSA** 身份验证，请使用以下格式：

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

```
}
```

备注 | 替换 `<role ARN>`为上一步中创建的角色的ARN。

2.安装 **Trident EKS** 插件。

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

eksc

以下示例命令将安装Trident EKS加载项：

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

更新**Trident EKS**加载项

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*群集*。
3. 选择要更新NetApp Trident CSI加载项的集群的名称。
4. 选择*Add-ones*选项卡。
5. 选择* Trident按NetApp显示*，然后选择*编辑*。
6. 在“*按NetApp配置Trident”页上，执行以下操作：
   a. 选择要使用的*版本*。
   b. 展开*可选配置设置*并根据需要进行修改。
   c. 选择 * 保存更改 * 。

AWS命令行界面

以下示例将更新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>'\"}"
```

eksc

- 检查FSxN Trident CSI加载项的当前版本。请替换 `my-cluster`为您的集群名称。

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

示例输出：

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

- 将此加载项更新到上一步输出中的update下返回的版本。

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

如果您删除了该 `--force` 选项、并且任何Amazon EKS附加设置与您的现有设置冲突、则更新Amazon EKS附加设置将失败；您将收到一条错误消息、以帮助您解决冲突。在指定此选项之前、请确保Amazon EKS附加组件不会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 Kubenetes字段管理的详细信息，请参阅 "Kubbernetes现场管理"。

卸载/删除**Trident EKS**加载项

您可以通过两种方式删除Amazon EKS附加项：

- 保留集群上的附加软件–此选项将删除Amazon EKS对任何设置的管理。此外、它还会使Amazon EKS无法通知您更新、并在您启动更新后自动更新Amazon EKS附加项。但是、它会保留集群上的附加软件。此选项可使附加组件成为自管理安装、而不是Amazon EKS附加组件。通过此选项、此附加组件不会出现停机。保留命令中的 `--preserve` 选项以保留此附加项。
- 从集群中完全删除附加软件–NetApp建议您仅在集群中没有依赖于此附加软件的资源时、才从集群中删除此附加软件。从命令中删除 `--preserve` 此选项 `delete` 以删除此加载项。

备注 如果此附加项具有关联的IAM帐户、则不会删除此IAM帐户。

管理控制台

1. 打开Amazon EKS控制台，网址为 https://console.aws.amazon.com/eks/home#/clusters。
2. 在左侧导航窗格中，选择*群集*。
3. 选择要删除的NetApp Trident CSI加载项的集群名称。
4. 选择*Add-ons*选项卡，然后选择Trident by NetApp。*
5. 选择 * 删除 * 。
6. 在*Remove NetApp_trdent-operator con確 認*对话框中，执行以下操作：

a. 如果您希望Amazon EKS停止管理此附加组件的设置、请选择*保留集群*。如果要在集群上保留附加软件、以便您可以自行管理附加软件的所有设置、请执行此操作。

b. 输入*NetApp_trdent-operator*。

c. 选择 * 删除 * 。

AWS命令行界面

请使用集群的名称进行替换 `my-cluster` 、然后运行以下命令。

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

eksc

以下命令将卸载Trident EKS加载项：

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

配置存储类

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

创建 **StorageClass** 对象

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

要手动配置存储后端，请参阅 创建没有自动后端配置的 Kubernetes StorageClass，了解有关单独创建 Trident 后端和 StorageClass 的说明。

备注	在 OpenShift 上，您必须单独创建 Trident 后端和 StorageClass。

了解自动后端配置的工作原理

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

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

指定所需 **StorageClass** 参数

创建 StorageClass 时需要定义以下三个参数：

参数	Required	Type	Description
`fsxFilesystemID`	是的。	string	FSx for NetApp ONTAP 文件系统 ID
`storageDriverName`	是的。	string	Trident 存储驱动程序（例如， `ontap-nas` 或 `ontap-san`）
`credentialsName`	是的。	string	包含 FSx for ONTAP 凭据的 Kubernetes Secret 或 AWS Secrets Manager secret ARN 的名称

指定可选参数

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

示例 **StorageClass** 配置文件。

以下示例显示了触发自动后端配置的 StorageClass。

YAML

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

JSON

```json
{
  "apiVersion": "storage.k8s.io/v1",
  "kind": "StorageClass",
  "metadata": {
    "name": "ontap-fsx-demo",
    "annotations": {
      "description": "Demo StorageClass for FSx for NetApp ONTAP"
    }
  },
```

```
  "provisioner": "csi.trident.netapp.io",
  "parameters": {
    "fsxFilesystemID": "fs-0abc123",
    "storageDriverName": "ontap-nas",
    "credentialsName": "trident-fsx-credentials"
  },
  "allowVolumeExpansion": true,
  "reclaimPolicy": "Delete",
  "volumeBindingMode": "Immediate"
}
```

创建 **StorageClass**

创建配置文件后，请运行以下命令来创建存储类。

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

现在，Kubernetes和Trident中都应显示一个*BASIC－Csi*存储类，并且Trident应已发现后端的池。

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

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

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

验证后端配置状态

Trident 在 StorageClass 注释中记录后端创建的结果。

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

要验证状态，请运行：

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

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

添加其他 FSxN 文件系统

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

编辑 StorageClass 并添加以下注释：

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

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

操作注意事项和限制

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

创建没有自动后端配置的 Kubernetes StorageClass

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

创建 Trident 后端

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

YAML

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-nas
  namespace: trident
spec:
  version: 1
  defaults:
    skipRecoveryQueue: "true"
  storageDriverName: ontap-nas
```

```
  backendName: tbc-ontap-nas
  svm: svm-name
  aws:
    fsxFilesystemID: fs-xxxxxxxxxx
    apiRegion: us-west-2
  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,
    "defaults": {
      "skipRecoveryQueue": "true"
    },
    "storageDriverName": "ontap-nas",
    "backendName": "tbc-ontap-nas",
    "svm": "svm-name",
    "aws": {
      "fsxFilesystemID": "fs-xxxxxxxxxx",
      "apiRegion": "us-west-2"
    },
    "credentials": {
      "name": "arn:aws:secretsmanager:us-west-2:xxxxxxxx:secret:secret-
name",
      "type": "awsarn"
    }
  }
}
```

FSx for ONTAP驱动程序详细信息

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

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

有关驱动程序详细信息、请参见 "NAS驱动程序" 和 "SAN驱动程序"。

创建后端

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

- 从YAML文件创建Trident后端配置(TBC)并运行以下命令：

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

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

- 验证是否已成功创建Trident后端配置(TBC)：

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

```
NAME                            BACKEND NAME          BACKEND UUID
PHASE   STATUS
backend-tbc-ontap-nas           tbc-ontap-nas         933e0071-66ce-4324-
b9ff-f96d916ac5e9   Bound   Success
```

有关其他配置选项的详细信息，请参见以下 后端高级配置和示例 部分。

配置存储类*无*自动后端配置

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

NFS 存储类

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

）：

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

iSCSI 的存储类

使用此示例为使用 iSCSI 的卷设置 StorageClass：

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

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

要在AWS Bottler套 件上配置NFS3卷、请将所需添加 `mountOptions`到存储类：

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

```
  - nfsvers=3
  - nolock
```

Trident StorageClass 属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	Type	值	优惠	请求	支持
介质[1]	string	HDD ，混合，SSD	Pool 包含此类型的介质；混合表示两者	指定的介质类型	ontap-nas ，ontap-nas-economy. ontap-nas-flexgroup ，ontap-san ，solidfire-san
配置类型	string	精简，厚	Pool 支持此配置方法	指定的配置方法	Thick：All ONTAP ；Thin ：All ONTAP & solidfire-san
后端类型	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	池	true false	Pool 支持具有快照的卷	启用了快照的卷	ontap-nas 、ontap-san 、solidfire-san
克隆	池	true false	Pool 支持克隆卷	启用了克隆的卷	ontap-nas 、ontap-san 、solidfire-san
加密	池	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	内部	正整数	Pool 能够保证此范围内的 IOPS	卷保证这些 IOPS	solidfire-san

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

有关存储类如何与和参数交互以控制Trident如何配置卷的详细信息 `PersistentVolumeClaim`、请参见"Kubernetes 和 Trident 对象"。

创建存储类

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

步骤

1. 这是一个Kubbernetes对象、因此请使用 `kubectl` 以在Kubernetes中创建。

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

2. 现在，Kubernetes和Trident中都应显示一个*BASIC－Csi*存储类，并且Trident应已发现后端的池。

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

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

提供 **SMB** 卷

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

后端高级配置和示例

有关后端配置选项，请参见下表：

参数	Description	示例
ve版本		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

参数	Description	示例
m年 月 日	集群或SVM管理LIF的IP地址可以指定完全限定域名(FQDN)。如果Trident是使用IPv6标志安装的、则可以设置为使用IPv6地址。IPv6地址必须用方括号定义、例如：[28e8：d9fb：a825：b7bf：69a8：d02f：9e7b：3555]。如果在字段下 aws`提供 `fsxFilesystemID、则无需提供、 managementLIF`因为Trident会从AWS检索SVM `managementLIF`信息。因此、您必须提供SVM下某个用户的凭据(例如vsadmin)、并且该用户必须具有此 `vsadmin 角色。	"10.0.0.1"、"[2001：1234：abc：：：fefe]"
dataLIF	协议 LIF 的 IP 地址。**ONTAP NAS** 驱动程序：NetApp 建议指定 dataLIF。如果未提供，Trident 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。**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 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]。	
autosExportPolicy	启用自动创建和更新导出策略[布尔值]。使用 `autoExportPolicy`和 `autoExportCIDRs`选项、Trident可以自动管理导出策略。	false
autosExportCIDR	用于筛选KubeNet节点IP的CIDR列表(启用时)。 `autoExportPolicy`使用 `autoExportPolicy`和 `autoExportCIDRs`选项、Trident可以自动管理导出策略。	"["0.0.0.0/0、"：：：/0"]"
标签	要应用于卷的一组任意 JSON 格式的标签	""
客户端证书	客户端证书的 Base64 编码值。用于基于证书的身份验证	""
clientPrivateKey	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证	""

参数	Description	示例
trustedCACertifate	受信任 CA 证书的 Base64 编码值。可选。用于基于证书的身份验证。	""
用户名	用于连接到集群或SVM的用户名。用于基于凭据的身份验证。例如、vsadmin。	
密码	用于连接到集群或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SVM管理LIF则派生。
s存储前缀	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	*请勿指定Amazon FSx for NetApp ONTAP。*提供的和 `vsadmin`不包含使用Trident检索聚合使用情况并对其进行限制所需的 `fsxadmin`权限。	请勿使用。
limitVolumeSize	如果请求的卷大小超过此值、则配置失败。此外、还会限制它为qtrees和FlexVol volume管理的卷的大小上限、并且此选项允许自定义每个LUN `qtreesPerFlexvol`的qtrees的最大数量	"" （默认情况下不强制实施）
lunsPerFlexvol	每个FlexVol volume的最大LUN数必须在[50、200]范围内。仅SAN。	“100”
debugTraceFlags	故障排除时要使用的调试标志。例如、 {"api"：false、"METHO"：true} 请勿使用 debugTraceFlags 除非您正在进行故障排除并需要详细的日志转储。	空
nfsMountOptions	NFS挂载选项的逗号分隔列表。通常会在存储类中为Kubnetes-永久性卷指定挂载选项、但如果在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。如果在存储类或配置文件中未指定挂载选项、则Trident不会在关联的永久性卷上设置任何挂载选项。	""
nasType	配置NFS或SMB卷创建。选项包括 nfs， smb`或为空。*必须设置为 `smb 对于SMB卷。*如果设置为空、则默认为NFS卷。	nfs
qtreesPerFlexvol	每个FlexVol volume的最大qtrees数、必须在[50、300]范围内	"200"

参数	Description	示例
smbShare	您可以指定以下选项之一：使用Microsoft管理控制台或ONTAP命令行界面创建的SMB共享的名称、或者允许Trident创建SMB共享的名称。对于Amazon FSx for ONTAP后端、此参数是必需的。	smb-share
useREST	用于使用 ONTAP REST API 的布尔参数。如果设置为 true，则Trident将使用ONTAP REST API与后端进行通信。此功能需要使用ONTAP 9.11.1及更高版本。此外、使用的ONTAP登录角色必须有权访问 ontap 应用程序。预定义的和角色可以满足这一 vsadmin 要求 cluster-admin 。	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机密密钥。	 "" "" ""
credentials	指定要存储在AWS密码管理器中的FSx SVM凭据。- name：密钥的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机密管理器密钥" 。	

用于配置卷的后端配置选项

您可以在中使用这些选项控制默认配置 defaults 配置部分。有关示例，请参见以下配置示例。

参数	Description	Default
spaceAllocation	LUN 的空间分配	true
s页面预留	空间预留模式；"无"(精简)或"卷"(厚)	无
sSnapshot 策略	要使用的 Snapshot 策略	无
qosPolicy	要为创建的卷分配的 QoS 策略组。选择每个存储池或后端的qosPolicy或adaptiveQosPolicy之一。将QoS策略组与Trident结合使用需要使用ONTAP 9™8或更高版本。您应使用非共享QoS策略组、并确保此策略组分别应用于每个成分卷。共享QoS策略组会对所有工作负载的总吞吐量实施上限。	""

参数	Description	Default
adaptiveQosPolicy	要为创建的卷分配的自适应 QoS 策略组。选择每个存储池或后端的qosPolicy或adaptiveQosPolicy之一。不受 ontap-nas-economy.	""
sSnapshot 预留	为快照"0"预留的卷百分比	如果 snapshotPolicy`为 `none，else“”
splitOnClone	创建克隆时，从其父级拆分该克隆	false
加密	在新卷上启用NetApp卷加密(NVE)；默认为 false。要使用此选项，必须在集群上获得 NVE 的许可并启用 NVE 。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，请参阅：“Trident如何与NVE和NAE配合使用”。	false
luksEncryption	启用 LUKS 加密。请参见 “使用Linux统一密钥设置(LUKS)”。仅限 SAN。	""
分层策略	要使用的层策略 none	
unixPermissions	新卷的模式。对于**SMB**卷保留为空。	""
securityStyle	新卷的安全模式。NFS支持 mixed 和 unix 安全模式。SMB支持 mixed 和 ntfs 安全模式。	NFS默认值为 unix。SMB默认值为 ntfs。

配置 **PVC**

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

创建**PVC**

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

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

示例清单

以下示例显示了基本的 PVC 配置选项。

PVC、可接入rwx

此示例显示了一个具有rwx访问权限的基本PVC，该PVC与名为的StorageClass关联 `basic-csi`。

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

使用 **iSCSI** 示例的 **PVC**

此示例展示了具有 RWO 访问权限的 iSCSI 基本 PVC，它与名为 `protection-gold`。

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

创建 **PVC**

步骤

1. 创建 PVC 。

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

2. 验证PVC状态。

```
kubectl get pvc
```

```
NAME        STATUS    VOLUME        CAPACITY    ACCESS MODES    STORAGECLASS
VOLUMEATTRIBUTESCLASS    AGE
fsx-demo    Bound     pvc-volume    1Gi         RWX             fsx-nfs
<unset>                 13s
```

有关存储类如何与和参数交互以控制Trident如何配置卷的详细信息 `PersistentVolumeClaim`、请参见"Kubernetes 和 Trident 对象"。

部署应用程序

创建存储类和PVC后、您可以将PV挂载到Pod。本节列出了将PV连接到POD的示例命令和配置。

部署示例应用程序

步骤

1. 将卷挂载到Pod中。

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

以下示例显示了将PVC连接到POD的基本配置：基本配置：

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

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

2. 验证卷是否已挂载到上 `/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。Pod应用程序将不再存在、但卷将保留。

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

在EKS集群上配置Trident EKS加载项

NetApp Trident简化了Kubelnetes中适用于NetApp ONTAP的Amazon FSx存储管理、使开发人员和管理员能够专注于应用程序部署。NetApp Trident EKS加载项包括最新的安全修补程序和错误修复、并已通过AWS验证、可与Amazon EKS配合使用。通过EKS加载项、您可以始终确保Amazon EKS集群安全稳定、并减少安装、配置和更新加载项所需的工作量。

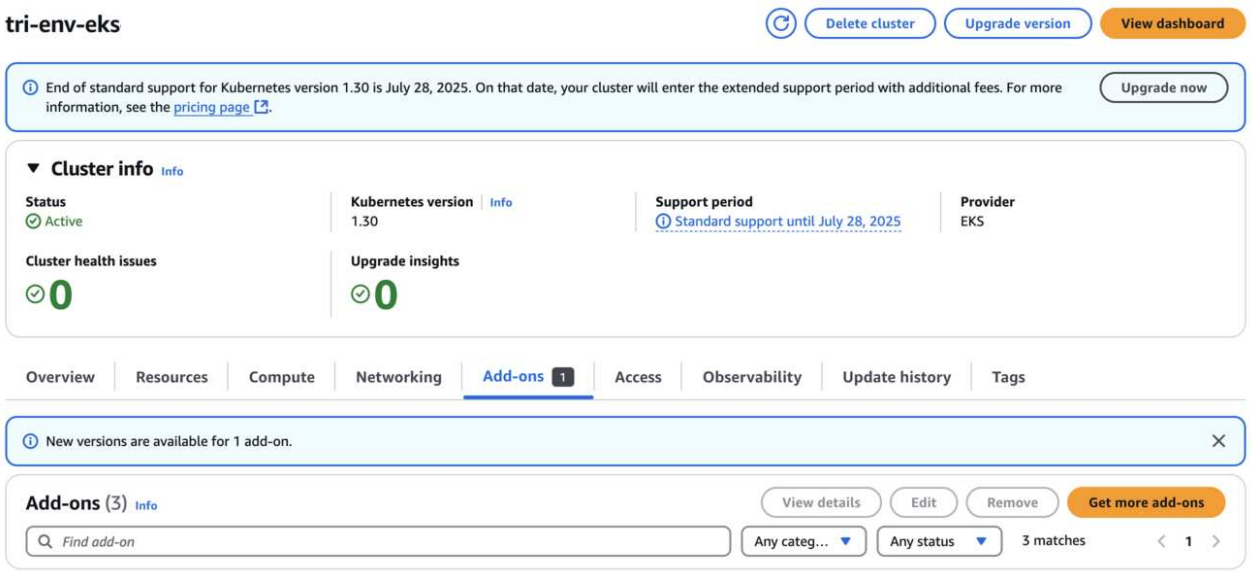
前提条件

在配置适用于AWS EKS的Trident加载项之前、请确保满足以下条件：

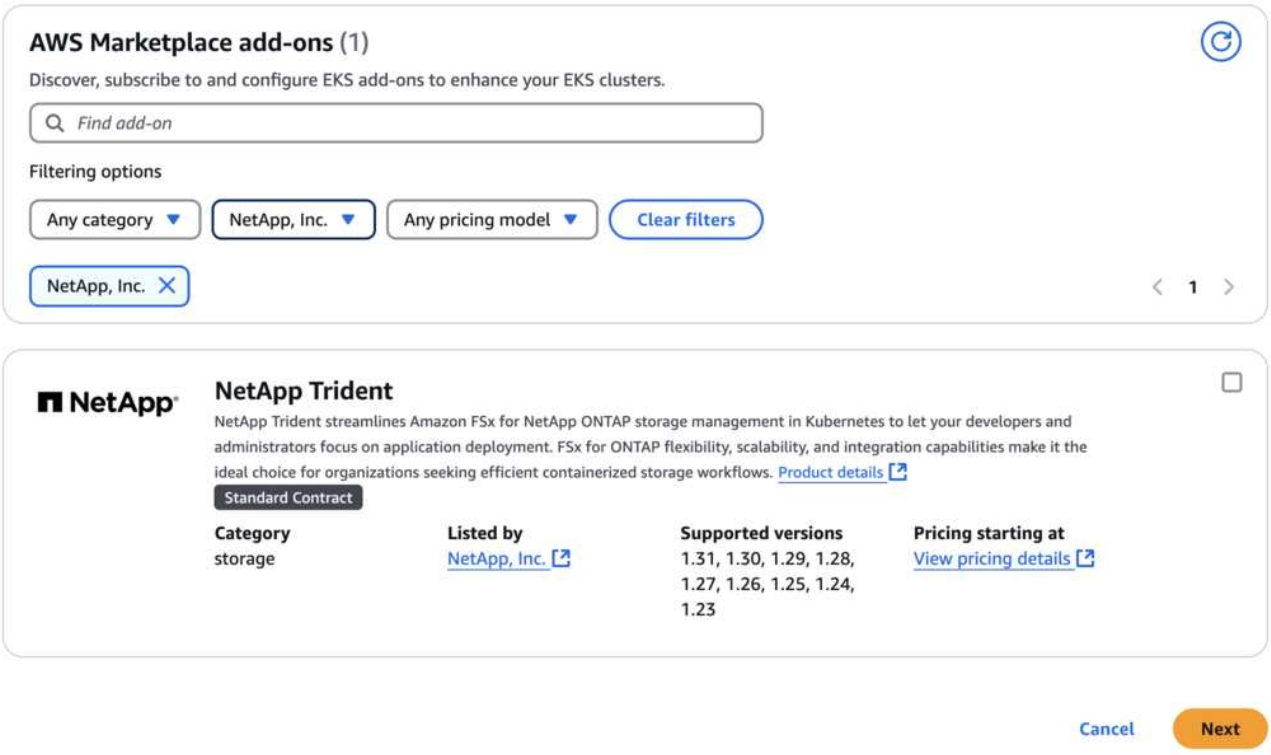
- 具有使用加载项的权限的Amazon EKS集群帐户。请参阅 "Amazon EKS附加项"。
- AWS对AWS Marketplace的权限：
  `"aws-marketplace:ViewSubscriptions",`
  `"aws-marketplace:Subscribe",`
  `"aws-marketplace:Unsubscribe`
- AMI类型：Amazon Linux 2 (AL2_x86_64)或Amazon Linux 2 ARM (AL2_ARM_64)
- 节点类型：AMD或ARM
- 现有Amazon FSx for NetApp ONTAP文件系统

步骤

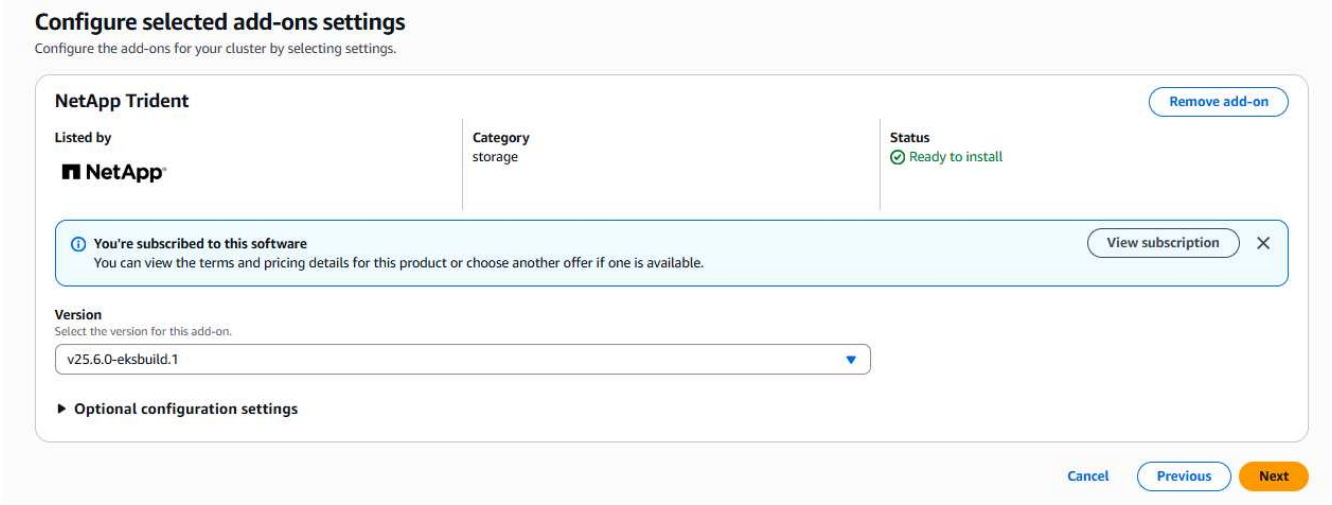
1. 请务必创建IAM角色和AWS密钥、以使EKS Pod能够访问AWS资源。有关说明，请参阅"创建IAM角色和AWS机密"。
2. 在EKS Kubernetes集群上、导航到*加载项*选项卡。



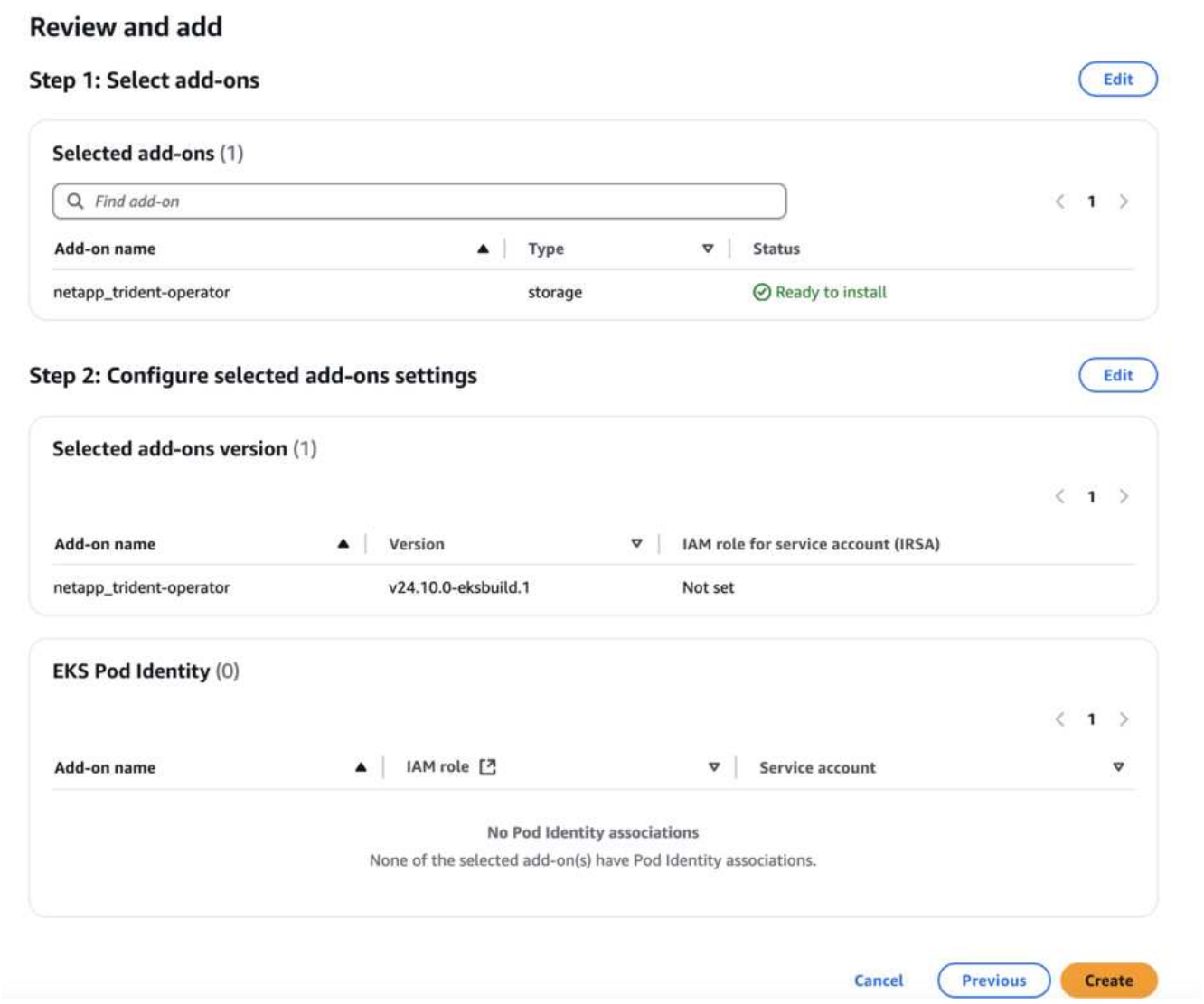
3. 转到*AWS Marketplace附加项*并选择_storage_类别。



4. 找到*Next* NetApp Trident并选中Trident插件的复选框，然后单击*Next*。
5. 选择所需的附加软件版本。

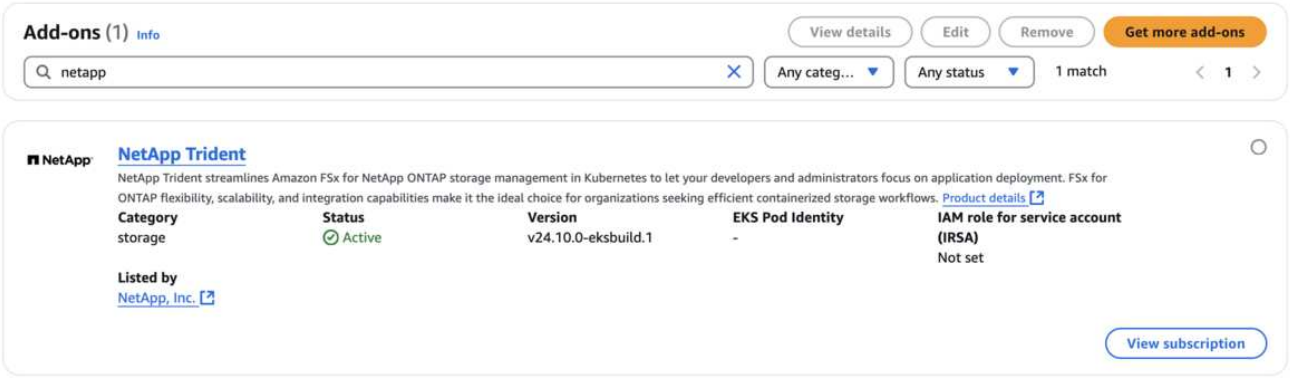


6. 配置所需的附加组件设置。



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

9. 验证此加载项的状态是否为_Active_.



10. 运行以下命令以验证Trident是否已正确安装在集群上：

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

11. 继续设置并配置存储后端。有关信息，请参见"配置存储后端"。

使用命令行界面安装/卸载**Trident EKS**加载项

使用命令行界面安装**NetApp Trident EKS**加载项：

以下示例命令安装Trident EKS 附加组件：

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

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）
```

使用命令行界面卸载**NetApp Trident EKS**加载项：

以下命令将卸载Trident EKS加载项：

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

使用 **kubectl** 创建后端

后端用于定义Trident与存储系统之间的关系。它会告诉Trident如何与该存储系统通信、以及Trident如何从该存储系统配置卷。安装Trident后、下一步是创建后端。`TridentBackendConfig`通过自定义资源定义(CRD)、您可以直接通过Kubednetes界面创建和管理Trident后端。您可以使用或对Kubornetes分发等效的命令行界面工具来执行此操作 `kubectl。

TridentBackendConfig

TridentBackendConfig(tbc tbconfig、、 tbackendconfig)是一个前端，具有名称节奏的CRD，使您可以使用管理Trident后端 kubectl。现在，Kubbernetes和存储管理员可以直接通过Kubbernetes 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 对象时，将发生以下情况：

- Trident会根据您提供的配置自动创建后端。这在内部表示为 TridentBackend (tbe， tridentbackend) CR。
- 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。前者是为用户提供的用于设计和配置后端的接口；后者是 Trident 表示实际后端对象的方式。

警告	TridentBackend`CRS由Trident自动创建。您 * 不应 * 修改它们。如果要更新后端、请通过修改对象来执行此操作 `TridentBackendConfig。

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
```

您还可以查看中的示例 "Trident 安装程序" 所需存储平台 / 服务的示例配置目录。

。 spec 获取后端特定的配置参数。在此示例中、后端使用 ontap-san 存储驱动程序、并使用此处所示的配置参数。有关所需存储驱动程序的配置选项列表、请参阅 "存储驱动程序的后端配置信息"。

sPec 部分还包括 credentials 和 deletionPolicy 字段，这些字段在 TridentBackendConfig CR 中新增：

- credentials ： 此参数为必填字段，包含用于向存储系统 / 服务进行身份验证的凭据。此密码设置为用户创建的 Kubernetes Secret 。凭据不能以纯文本形式传递，因此会导致错误。
- deeltionPolicy ： 此字段定义删除 TridentBackendConfig 时应发生的情况。它可以采用以下两种可能值之一：
  - delete ： 这将同时删除 TridentBackendConfig CR 和关联后端。这是默认值。

- `retain`：删除 `TrdentBackendConfig` CR 后，后端定义仍存在，可使用 `tridentctl` 进行管理。将删除策略设置为 `retain` 允许用户降级到早期版本（21.04 之前）并保留创建的后端。创建 `TridentBackendConfig` 后，可以更新此字段的值。

备注	后端名称使用 `sPec.backendName` 设置。如果未指定，则后端的名称将设置为 `TridentBackendConfig` 对象的名称（metadata.name）。建议使用 `sPec.backendName` 显式设置后端名称。
提示	使用创建的后端 `tridentctl`没有关联 `TridentBackendConfig`对象。您可以通过创建CR来 `TridentBackendConfig`选择使用管理此类后端 `kubectl`。必须注意指定相同的配置参数(如 `spec.backendName`、、 `spec.storagePrefix` `spec.storageDriverName`等)。Trident将自动将新创建的与已有的后端绑定 `TridentBackendConfig`。

步骤概述

要使用 `kubectl` 创建新后端，应执行以下操作：

1. 创建 "Kubernetes 机密"。此密钥包含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，并引用了上一步中创建的密钥。

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

第 **1** 步：创建 **Kubernetes** 机密

创建一个机密，其中包含后端的访问凭据。这是每个存储服务 / 平台所特有的。以下是一个示例：

```
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
```

下表汇总了每个存储平台的机密中必须包含的字段：

存储平台机密字段问题描述	机密	字段问题描述
Azure NetApp Files	clientId	应用程序注册中的客户端 ID
Element （ NetApp HCI/SolidFire ）	端点	使用租户凭据的 SolidFire 集群的 MVIP

存储平台机密字段问题描述	机密	字段问题描述
ONTAP	username	用于连接到集群 /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	客户端权限密钥	客户端专用密钥的 Base64 编码值。用于基于证书的身份验证
ONTAP	用户名	入站用户名。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy` .
ONTAP	chapInitiatorSecret	CHAP 启动程序密钥。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy` .
ONTAP	chapTargetUsername	目标用户名。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy` .
ONTAP	chapTargetInitiatorSecret	CHAP 目标启动程序密钥。如果 useCHAP=true ，则为必需项。适用于 `ontap-san` 和 `ontap-san-economy` .

在下一步中创建的 `TrdentBackendConfig` 对象的 `sPec.credentials` 字段将引用此步骤中创建的机密。

第2步：创建 **`TridentBackendConfig` CR**

现在，您可以创建 `TridentBackendConfig` CR 了。在此示例中，使用 `TriventBackendConfig` 对象创建使用` ontap-san `驱动程序的后端，如下所示：

```
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：验证的状态 **TridentBackendConfig CR**

现在，您已创建 TridentBackendConfig CR ，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
```

已成功创建后端并将其绑定到 TridentBackendConfig CR 。

阶段可以采用以下值之一：

- Bound： TridentBackendConfig CR与后端关联、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 CRS 均处于此阶段。此阶段发生更改后，它将无法再次还原为 "Unbound （已取消绑定） " 。
- Deleting： TridentBackendConfig CR deletionPolicy 已设置为delete。当 TridentBackendConfig CR将被删除、它将过渡到Deleting状态。
  - 如果后端不存在永久性卷请求(PVC)、则删除 TridentBackendConfig `将导致Trident删除后端以及CR。 `TridentBackendConfig
  - 如果后端存在一个或多个 PVC ，则会进入删除状态。TridentBackendConfig CR 随后也会进入删除阶段。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， TridentBackendConfig CR 仍引用已删除的后端。无论 detionPolicy 值如何，仍可删除 TridentBackendConfig CR 。
- Unknown：Trident无法确定与CR关联的后端的状态或是否存在 TridentBackendConfig。例如、如果API服务器未响应或 `tridentbackends.trident.netapp.io`缺少CRD。这可能需要干预。

在此阶段，已成功创建后端！此外，还可以处理多个操作，例如 "后端更新和后端删除"。

（可选）第 **4** 步：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 YAML/JSON 转储 `TridentBackendConfig` 。

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

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  creationTimestamp: 2021-04-21T20:45:11Z
  finalizers:
    - trident.netapp.io
  generation: 1
  name: backend-tbc-ontap-san
  namespace: trident
  resourceVersion: "947143"
  uid: 35b9d777-109f-43d5-8077-c74a4559d09c
spec:
  backendName: ontap-san-backend
  credentials:
    name: backend-tbc-ontap-san-secret
  managementLIF: 10.0.0.1
  dataLIF: 10.0.0.2
  storageDriverName: ontap-san
  svm: trident_svm
  version: 1
status:
  backendInfo:
    backendName: ontap-san-backend
    backendUUID: 8d24fce7-6f60-4d4a-8ef6-bab2699e6ab8
  deletionPolicy: delete
  lastOperationStatus: Success
  message: Backend 'ontap-san-backend' created
  phase: Bound
```

`backendInfo`包含 `backendName`响应CR而创建的后端的 `TridentBackendConfig`和 `backendUUID。 lastOperationStatus`字段表示CR的上次操作状态，可以是用户触发的操作(例如，用户在中更改了某些内容)，也可以是Trident触发的操作 `TridentBackendConfig(例如， spec` 在Trident重新启动期间)。可以是成功、也可以是失败。 `phase`表示CR和后端之间关系的状态 `TridentBackendConfig。在上面的示例中、 `phase`具有绑定值、这意味着 `TridentBackendConfig`CR与后端关联。

您可以运行 `kubectl -n trident describe tbc <tbc-cr-name>` 命令来获取事件日志的详细信息。

警告	您不能使用 `tridentctl` 更新或删除包含关联的 `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** 管理 **Trident** 后端

了解如何使用 `kubectl` 执行后端管理操作。

删除后端

通过删除 `TridentBackendConfig`，您可以指示Trident删除/保留后端(基于 `deletionPolicy`)。要删除后端、请确保 `deletionPolicy`将设置为`delete`。要仅删除 `TridentBackendConfig`，请确保 `deletionPolicy`将设置为保留。这样可以确保后端仍然存在，并且可以使用进行管理 `tridentctl`。

运行以下命令：

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

Trident不会删除正在使用的Kubnetes加密 `TridentBackendConfig`。Kubernetes 用户负责清理密钥。删除机密时必须小心。只有在后端未使用机密时，才应将其删除。

查看现有后端

运行以下命令：

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

您也可以运行 `tridentctl get backend -n trident` 或 `tridentctl get backend -o YAML -n trident` 来获取存在的所有后端的列表。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、必须更新对象中使用的Kubbernetes机密 `TridentBackendConfig`。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（例如所使用的 ONTAP SVM 的名称）。
  - 您可以更新 `TridentBackendConfig` 使用以下命令直接通过KubeNet访问对象：

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

- 或者、您也可以对现有进行更改 `TridentBackendConfig` 使用以下命令执行CR：

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

备注

- 如果后端更新失败，则后端仍会保持在其上次已知配置中。您可以通过运行 `kubectl get tbc <tbc-name> -o yaml -n trident` 或 `kubectl describe tbc <tbc-name> -n trident` 来查看日志以确定发生原因。
- 确定并更正配置文件中的问题后，您可以重新运行 update 命令。

使用 **tridentctl** 管理 **Trident** 后端

了解如何使用 `tridentctl` 执行后端管理操作。

创建后端

创建后 "后端配置文件"下，运行以下命令：

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

如果后端创建失败，则后端配置出现问题。您可以运行以下命令来查看日志以确定发生原因：

```
tridentctl logs -n trident
```

确定并更正配置文件中的问题后，您只需再次运行 `create` 命令即可。

删除后端

要从Trident中删除后端、请执行以下操作：

1. 检索后端名称：

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

2. 删除后端：

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

备注 如果Trident从此后端配置了仍存在的卷和快照、则删除后端将阻止其配置新卷。后端将继续处于"

正在删除"状态。

查看现有后端

要查看 Trident 了解的后端，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，则后端配置出现问题或您尝试的更新无效。您可以运行以下命令来查看日志以确定发生原因：

```
tridentctl logs -n trident
```

确定并更正配置文件中的问题后，您只需再次运行 `update` 命令即可。

确定使用后端的存储类

以下是您可以使用问题解答与 JSON 回答的问题的示例，这些问题会 `tridentctl` 后端对象的输出。此操作将使用 `JQ` 实用程序，您需要安装该实用程序。

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

这也适用于使用 `TridentBackendConfig` 创建的后端。

选择 **Trident** 后端管理方法

了解在Trident中管理后端的不同方法。

随附 `TridentBackendConfig`现在、管理员可以通过两种独特的方式管理后端。这会提出以下问题：

- 使用 `tridentctl` 创建的后端是否可以使用 `TridentBackendConfig` 进行管理?
- 使用 `TridentBackendConfig` 创建的后端是否可以使用 `tridentctl` 进行管理?

管理 `tridentctl` 后端使用 `TridentBackendConfig`

本节介绍通过创建 `TridentBackendConfig` 对象直接通过 Kubernetes 界面管理使用 `tridentctl` 创建的后端所需的步骤。

这适用于以下情形：

- 已有后端、但没有 `TridentBackendConfig` 因为它们是使用创建的 `tridentctl`。
- 使用 `tridentctl` 创建的新后端，而存在其他 `TridentBackendConfig` 对象。

在这两种情况下、都将继续存在后端、Trident会为这些后端计划卷并在其上运行。管理员可以选择以下两种方式之一：

- 继续使用 `tridentctl` 管理使用它创建的后端。
- 使用 `tridentctl` 创建的后端绑定到新的 `TridentBackendConfig` 对象。这样做意味着将使用 `kubectl` 而不是 `tridentctl` 来管理后端。

要使用 `kubectl` 管理已有后端，您需要创建一个绑定到现有后端的 `TridentBackendConfig` 。下面简要介绍了它的工作原理：

1. 创建 Kubernetes 机密。此密钥包含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，并引用了上一步中创建的密钥。必须注意指定相同的配置参数（例如 `sPec.backendName` ， `sPec.storagePrefix` ， `sPec.storageDriverName` 等）。`sPec.backendName` 必须设置为现有后端的名称。

第 **0** 步：确定后端

以创建 `TridentBackendConfig` 如果绑定到现有后端、则需要获取后端配置。在此示例中，假设已使用以下 JSON 定义创建了后端：

```
tridentctl get backend ontap-nas-backend -n trident
+---------------------+----------------
+--------------------------------------+--------+---------+
|           NAME        | STORAGE DRIVER |                  UUID
| STATE   | VOLUMES |
+---------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas-backend   | ontap-nas      | 52f2eb10-e4c6-4160-99fc-
96b3be5ab5d7 | online |      25 |
+---------------------+----------------
```

```
+-------------------------------------+--------+---------+
```

```
cat ontap-nas-backend.json
```

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.10.10.1",
  "dataLIF": "10.10.10.2",
  "backendName": "ontap-nas-backend",
  "svm": "trident_svm",
  "username": "cluster-admin",
  "password": "admin-password",
  "defaults": {
    "spaceReserve": "none",
    "encryption": "false"
  },
  "labels": {
    "store": "nas_store"
  },
  "region": "us_east_1",
  "storage": [
    {
      "labels": {
        "app": "msoffice",
        "cost": "100"
      },
      "zone": "us_east_1a",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "true",
        "unixPermissions": "0755"
      }
    },
    {
      "labels": {
        "app": "mysqldb",
        "cost": "25"
      },
      "zone": "us_east_1d",
      "defaults": {
        "spaceReserve": "volume",
        "encryption": "false",
        "unixPermissions": "0775"
```

```
        }
      }
    ]
}
```

第 1 步：创建 **Kubernetes** 机密

创建一个包含后端凭据的机密，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
```

第2步：创建 **`TridentBackendConfig` CR**

下一步是创建一个 `TridentBackendConfig` CR ，该 CR 将自动绑定到已有的 `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)必须与原始后端的顺序相同。
- 凭据通过 Kubernetes Secret 提供，而不是以纯文本形式提供。

在这种情况下， `TridentBackendConfig` 将如下所示：

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

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
```

```
  name: tbc-ontap-nas-backend
spec:
  version: 1
  storageDriverName: ontap-nas
  managementLIF: 10.10.10.1
  dataLIF: 10.10.10.2
  backendName: ontap-nas-backend
  svm: trident_svm
  credentials:
    name: mysecret
  defaults:
    spaceReserve: none
    encryption: 'false'
  labels:
    store: nas_store
  region: us_east_1
  storage:
  - labels:
      app: msoffice
      cost: '100'
    zone: us_east_1a
    defaults:
      spaceReserve: volume
      encryption: 'true'
      unixPermissions: '0755'
  - labels:
      app: mysqldb
      cost: '25'
    zone: us_east_1d
    defaults:
      spaceReserve: volume
      encryption: 'false'
      unixPermissions: '0775'
```

```
kubectl create -f backend-tbc-ontap-nas.yaml -n trident
tridentbackendconfig.trident.netapp.io/tbc-ontap-nas-backend created
```

第3步：验证的状态 **TridentBackendConfig** CR

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，可以使用 `tbc-ontap-nas-backend` `TridentBackendConfig` 对象对后端进行全面管理。

管理 **TridentBackendConfig** 后端使用 **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` 来完全管理此类后端，方法是删除 `TridentBackendConfig` 并确保将 `spec.deletionPolicy` 设置为 `retain` 。

第 0 步：确定后端

例如，假设使用 `TridentBackendConfig` 创建了以下后端：

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

```
+--------------------------------------+--------+---------+
```

从输出中可以看出这一点 `TridentBackendConfig` 已成功创建并绑定到后端[观察后端的UUID]。

第1步：确认 `deletionPolicy` 设置为 `retain`

让我们来看看的价值 `deletionPolicy`。需要将其设置为 `retain`。这样可以确保在删除CR时 `TridentBackendConfig`，后端定义仍然存在，并且可以使用进行管理 `tridentctl`。

```
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        delete

# Patch value of deletionPolicy to retain
kubectl patch tbc backend-tbc-ontap-san --type=merge -p
'{"spec":{"deletionPolicy":"retain"}}' -n trident
tridentbackendconfig.trident.netapp.io/backend-tbc-ontap-san patched

#Confirm the value of deletionPolicy
kubectl get tbc backend-tbc-ontap-san -n trident -o wide
NAME                     BACKEND NAME        BACKEND UUID
PHASE   STATUS    STORAGE DRIVER   DELETION POLICY
backend-tbc-ontap-san   ontap-san-backend   81abcb27-ea63-49bb-b606-
0a5315ac5f82   Bound   Success   ontap-san        retain
```

备注	请勿继续执行下一步，除非将 `deletionPolicy` 设置为 `retain` 。

第2步：删除 `TridentBackendConfig` CR

最后一步是删除 `TridentBackendConfig` CR 。确认 `deeltionPolicy` 设置为 `retain` 后，您可以继续执行删除：

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

删除对象后 `TridentBackendConfig`、Trident会直接将其删除、而不会实际删除后端本身。

创建和管理存储类

创建 **Trident** 存储类

配置Kubnetes StorageClass对象并创建存储类、以指示Trident如何配置卷。

配置**Kubnetes StorageClass**对象

`https://kubernetes.io/docs/concepts/storage/storage-classes/["Kubnetes StorageClass对象"^]`将`Trident`标识为用于该类的配置程序、并指示`Trident`如何配置卷。例如：

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

有关存储类如何与和参数交互以控制Trident如何配置卷的详细信息 `PersistentVolumeClaim`、请参见"Kubernetes 和 Trident 对象"。

创建存储类。

创建StorageClass对象后、您可以创建存储类。 [存储类示例] 提供了一些可供您使用或修改的基本示例。

步骤

1. 这是一个Kubbernetes对象、因此请使用 `kubectl` 以在Kubernetes中创建。

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

2. 现在，Kubernetes和Trident中都应显示一个*BASIC－Csi*存储类，并且Trident应已发现后端的池。

```
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提供 "适用于特定后端的简单存储类定义"。

或者、您也可以进行编辑 `sample-input/storage-class-csi.yaml.templ` 安装程序随附的文件、然后进行替换 *`BACKEND_TYPE`* 和存储驱动程序名称。

```
./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 Trident 存储类

您可以查看现有存储类、设置默认存储类、确定存储类后端以及删除存储类。

查看现有存储类

- 要查看现有 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的已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。如果用户未在永久性卷声明（ PVC ）中指定永久性卷，则此存储类将用于配置永久性卷。

- 通过在存储类定义中将标注 `storageclass.Kubernetes 。 io/is-default-class` 设置为 true 来定义默认存储类。根据规范，任何其他值或标注不存在均视为 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"}}}'
```

- 同样，您也可以使用以下命令删除默认存储类标注：

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

Trident 安装程序包中也有包含此标注的示例。

备注	集群中一次只能有一个默认存储类。Kubernetes 在技术上不会阻止您拥有多个存储类，但其行为就像根本没有默认存储类一样。

确定存储类的后端

以下是您可以使用为Trident后端对象输出的JSON回答的问题示例 `tridentctl`。这将使用 `jq`实用程序、您可能需要先安装该实用程序。

```
tridentctl get storageclass -o json | jq  '[.items[] | {storageClass:
.Config.name, backends: [.storage]|unique}]'
```

删除存储类

要从 Kubernetes 中删除存储类，请运行以下命令：

```
kubectl delete storageclass <storage-class>
```

` < 存储类 >` 应替换为您的存储类。

通过此存储类创建的任何永久性卷将保持不变、Trident将继续对其进行管理。

备注	Trident会对其创建的卷强制使用空白 `fsType`。对于iSCSI后端、建议在StorageClass中强制实施 `parameters.fsType`。您应删除现有StorageClasses、然后使用指定的重新创建它们 `parameters.fsType`。

配置和管理卷

预配 **Trident** 卷

创建一个使用已配置的Kubernetes StorageClass来请求PV访问权限的永久性卷请求(PVC)。然后、您可以将PV挂载到POD。

概述

A "*PersigentVolumeClaim*" (PVC)是指请求访问集群上的永久卷。

可以将PVC配置为请求特定大小的存储或访问模式。通过使用关联的StorageClass，集群管理员可以控制不限于持续卷大小和访问模式(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. 将卷挂载到Pod中。

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

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

2. 验证卷是否已挂载到上 `/my/mount/path`。

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

3. 现在、您可以删除Pod。Pod应用程序将不再存在、但卷将保留。

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

示例清单

▼ **PersentVolumeClaim**示例清单

这些示例显示了基本的PVC配置选项。

PVC、带读取器

此示例显示了一个具有读取权限的基本PVC、该PVC与名为的StorageClass关联 `basic-csi`。

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

采用**NVMe/TCP**的**PVC**

此示例显示了一个与名为的StorageClass关联的具有读取权限的NVMe/TCP的基本PVC `protection-gold`。

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

▼ **POD**清单示例

这些示例显示了将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
```

有关存储类如何与和参数交互以控制Trident如何配置卷的详细信息 `PersistentVolumeClaim`、请参见"Kubernetes 和 Trident 对象"。

移动卷

从 Trident 26.06 开始，存储管理员可以无中断地将 iSCSI 卷移动到同一 ONTAP 集群中的

不同 ONTAP 节点和聚合，同时使用该卷的工作负载保持运行。您可以使用 `TridentVolumeMove`自定义资源或 `tridentctl`等效资源以声明方式请求移动，Trident 将协调 ONTAP 和主机，确保 I/O 不会中断。

要求

卷移动仅适用于 iSCSI 卷的 `ontap-san` 驱动程序。不支持 NVMe 和 NAS 协议。

在移动卷之前，请确保满足以下要求：

- 后端使用 `ontap-san` 驱动程序与 ONTAP REST API。
- ONTAP 集群至少有两个节点。
- 您拥有以下凭据集之一：
  - 如果使用专用自定义 ONTAP 角色，则为 ONTAP 9.19.1 或更高版本。
  - 早期 ONTAP 版本上的集群管理员凭据。

工作原理

Trident 在同一 ONTAP 集群内的两个位置之间移动卷 (LUN)。在数据移动之前，Trident 会在现有路径旁边向目标添加新的 iSCSI 路径，以便主机多路径设备永远不会失去访问权限。ONTAP 完成移动后，Trident 会删除陈旧的源路径。

Trident 通过一组有序状态驱动移动，这些状态在 `TridentVolumeMove`资源中可见，以便您可以跟踪移动进度。

移动卷

识别卷和位置

1. 查找要移动的 PV 的名称。PV 名称也是移动资源的名称。
2. 在 ONTAP 集群中，记录该卷的当前（源）节点和聚合，并选择目标节点和聚合。

开始移动

使用任一方法。两者都创建相同的 `TridentVolumeMove`资源，其名称为 PV 名称。

使用 **tridentctl**

```
tridentctl create volume-move \
  --volume      pvc-8a814d62-bd58-4253-b0d1-82f2885db671 \
  --source-node ontap-node-01 \
  --source-pool aggr1_node01 \
  --target-node ontap-node-02 \
  --target-pool aggr1_node02 \
  -n trident
```

`tvm` 是 `volume-move` 的别名。添加 `--delete-after-success 10m` 以在移动成功后自动删除资源。

使用 **TridentVolumeMove** 资源

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeMove
metadata:
  name: pvc-8a814d62-bd58-4253-b0d1-82f2885db671   # the PV to move
  namespace: trident
spec:
  sourceNode: ontap-node-01
  sourcePool: aggr1_node01
  targetNode: ontap-node-02
  targetPool: aggr1_node02
  # deleteAfterSuccess: 10m   # optional
```

应用资源：

```
kubectl apply -f tridentvolumemove.yaml
```

规格字段

sourceNode / targetNode

ONTAP 节点名称（非 UUID）。

sourcePool / targetPool

ONTAP 聚合名称。

deleteAfterSuccess

可选持续时间，例如 `30s`或 `10m`。设置为 `0`可立即删除资源，或省略此选项以保留资源。

监控移动

使用 `tvm`简称或 `tridentvolumemove`跟踪进度：

```
kubectl get tridentvolumemove -n trident
```

要查看当前状态和每个节点的进度，请描述资源：

```
kubectl describe tridentvolumemove <pv-name> -n trident
```

卷可以在 `Moving`状态期间具有并发附件，因为 ONTAP 可能需要一段时间才能完成大卷的移动。为了确保无缝移动，Trident 在 Trident 控制器和 Trident 节点之间进行协调。每个状态按顺序执行以下步骤，最终以 `Succeeded`或 `Failed`结束。

状态	发生了什么
`Pending`	Trident接受并验证移动请求，并在开始任何存储更改之前收集所需的信息。
`ControllerStaging`	Trident 准备目标位置。它将目标节点的存储接口映射到 LUN 并更新 Kubernetes，以便将来节点重启时重新连接到正确的位置。
`NodeStaging`	在每个已挂载卷的主机上，Trident 会在现有路径旁边添加来自目标节点的新存储路径，以便 Pod 的 I/O 在两个路径集上继续进行。
`Moving`	ONTAP 将卷数据复制并转移到目标聚合。这是最长的阶段，ONTAP 全程处理，而 Trident 持续轮询直至完成。在此状态下允许新发布。
`ControllerUnstaging`	数据移动完成。Trident 更新 Kubernetes 以从所需状态中删除源路径，然后从 LUN 中取消源节点的存储接口的映射。
`NodeUnstaging`	在每个主机上，Trident 会删除过时的源路径。主机只保留目标路径，多路径设备继续提供 I/O 服务而不会中断。
`Succeeded`	此卷在目标位置已完全移动并处于联机状态。Trident 会保留 `TridentVolumeMove`资源，如果您设置了 `deleteAfterSuccess`，则会自动将其删除。
`Failed`	无法恢复的错误导致移动停止或无法启动。Trident 重置卷状态，以便恢复正常操作，并保留带有错误消息的资源。请查看此消息，修复根本原因，删除此资源，然后重试。

该 `.status.attachments`列表报告了每个附加节点在其 `state`字段中的进度：

附加状态	含义
`Pending`	节点被标识为已连接；工作尚未启动。
`Bridged`	在现有路径旁边添加新的目标路径。
`Migrated`	现在通过新路径提供流量。
`Cleaned`	已从节点中删除过时的源路径。
`Detached`	节点在移动过程中分离，在 Trident 评估附加操作是否已准备好继续下一步时，该节点被排除在进一步考虑范围之外。
`Failed`	此节点上出现终端错误。查看消息。

清理移动资源

如果您设置了 `deleteAfterSuccess`，Trident 将自动删除该资源。否则，您可以手动将其删除。在请求对同一卷进行另一次移动之前，请确保该卷不存在处于 `Succeeded` 或 `Failed` 状态的 `TridentVolumeMove` 资源。

备注	Trident不会自动删除失败的移动。Trident保留 `Failed`资源作为审计记录。修复根本原因后，请手动删除资源，然后重试。

扩展 **Trident** 卷

Trident为 Kubernetes 用户提供了在创建卷后扩展卷的能力。查找有关扩展 iSCSI、NFS

、SMB、NVMe/TCP 和 FC 卷所需的配置的信息。

展开 iSCSI 卷

您可以使用 CSI 配置程序扩展 iSCSI 永久性卷（ PV ）。

备注	iSCSI 卷扩展受 `ontap-san` ， `ontap-san-economy` ， `solidfire-san` 驱动程序支持，需要 Kubernetes 1.16 及更高版本。

第 1 步：配置 StorageClass 以支持卷扩展

编辑StorageClass定义以设置 `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，使其包含 `allowVolumeExpansion` 参数。

第 2 步：使用您创建的 StorageClass 创建 PVC

编辑PVC定义并更新 `spec.resources.requests.storage` 以反映新需要的大小、该大小必须大于原始大小。

```
cat pvc-ontapsan.yaml
```

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

```
  storageClassName: ontap-san
```

Trident会创建一个永久性卷(PV)并将其与此永久性卷请求(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
```

第 3 步：定义连接 PVC 的 POD

将PV连接到POD以调整大小。调整 iSCSI PV 大小时，有两种情况：

- 如果PV连接到Pod、则Trident会扩展存储后端的卷、重新扫描设备并重新设置文件系统大小。
- 尝试调整未连接PV的大小时、Trident会扩展存储后端的卷。将 PVC 绑定到 Pod 后， Trident 会重新扫描设备并调整文件系统大小。然后， Kubernetes 会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
```

第 4 步：展开 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：

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

展开 FC 卷

您可以使用CSI配置程序扩展FC持久卷(PV)。

> 备注　驱动程序支持FC卷扩展 `ontap-san`、并且需要Kubernetes 1.16及更高版本。

第 1 步：配置 StorageClass 以支持卷扩展

编辑StorageClass定义以设置 `allowVolumeExpansion` 字段设置为 `true`。

```
cat storageclass-ontapsan.yaml
```

```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，使其包含 `allowVolumeExpansion` 参数。

第 2 步：使用您创建的 **StorageClass** 创建 **PVC**

编辑PVC定义并更新 `spec.resources.requests.storage` 以反映新需要的大小、该大小必须大于原始大小。

```
cat pvc-ontapsan.yaml
```

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

Trident会创建一个永久性卷(PV)并将其与此永久性卷请求(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
```

第 3 步：定义连接 **PVC** 的 **POD**

将PV连接到POD以调整大小。调整FC PV大小有两种情形：

- 如果PV连接到Pod、则Trident会扩展存储后端的卷、重新扫描设备并重新设置文件系统大小。
- 尝试调整未连接PV的大小时、Trident会扩展存储后端的卷。将 PVC 绑定到 Pod 后， Trident 会重新扫描设备并调整文件系统大小。然后， Kubernetes 会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
```

第 4 步：展开 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：

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

展开 NFS 卷

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

第 1 步：配置 **StorageClass** 以支持卷扩展

要调整 NFS PV 的大小，管理员首先需要将 `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
```

如果您已创建没有此选项的存储类，则只需使用 `kubectl edit storageclass` 编辑现有存储类即可进行卷扩展。

第 2 步：使用您创建的 **StorageClass** 创建 **PVC**

```
cat pvc-ontapnas.yaml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Trident应该为该 PVC 创建一个 20 MiB 的 NFS PV：

```
kubectl get pvc
NAME           STATUS    VOLUME
CAPACITY     ACCESS MODES    STORAGECLASS     AGE
ontapnas20mb   Bound     pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s
```

```
kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

第 3 步：展开 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`至 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
# ...
```

第 4 步：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常：

```
kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n trident
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

了解 **RWX NVMe** 子系统限制

使用 NVMe 协议的 ReadWriteMany (RWX) 卷的可扩展性限制为每卷 64 个节点。以下包括限制，解释了所涉及的 NVMe 子系统体系结构，并概述了所需的解决步骤。

了解 **64** 节点限制

如果您计划将 ReadWriteMany (RWX) 卷与 NVMe 协议一起使用，则单个 RWX NVMe 卷不能由 Kubernetes 群集中超过 64 个节点装载。

请勿计划在 64 个以上节点上挂载相同 RWX NVMe PersistentVolumeClaim 的工作负载。

此限制仅适用于使用 NVMe 协议的 RWX 卷。

了解 **NVMe** 子系统模型

每卷子系统模型（**Trident** 版本早于 **26.02**）

在 26.02 之前的 Trident 版本中，RWX NVMe 卷使用每个卷的子系统模型进行调配。每个 RWX NVMe 卷都映射到 ONTAP 上其自己的专用 NVMe 子系统。

此模型很简单，但具有较低的可扩展性限制。在大型 Kubernetes 集群中，由于每个 RWX 卷消耗一个专用子系统，因此子系统控制器限制很快就会达到。

超级子系统模型（在 **Trident 26.02** 中引入）

从 Trident 26.02 开始，RWX NVMe 卷使用共享超级子系统模型。多个 RWX NVMe 卷共享同一个 NVMe 子系统。

每个超级子系统最多支持 1024 个命名空间（卷）。该模型显著提高了 RWX 工作负载的可扩展性，并降低了达到 ONTAP 子系统限制的可能性。

每个 RWX NVMe 卷最多支持 64 个节点。

识别错误症状

如果大规模创建或附加 RWX NVMe 卷，您可能会发现类似于以下错误：

```
Maximum number of controllers reached. No more controllers can be created.
```

此错误表示已达到 ONTAP NVMe 子系统控制器限制。

解决子系统限制错误

要超越每卷子系统的限制并利用超级子系统模型，请升级到 Trident 26.02 或更高版本。

升级 **Trident** 以应用超级子系统模型

要为 RWX NVMe 卷应用超级子系统模型：

1. 将 Trident 升级到 26.02 或更高版本。
2. 将使用 RWX NVMe 卷的所有 Pod 缩小到零副本。
3. 确认没有工作负载正在使用 RWX NVMe 卷。
4. 将 pod 重新扩容。

此重新启动顺序可确保使用超级子系统模型连接 RWX NVMe 卷。

- 此限制仅适用于使用 NVMe 协议的 RWX 卷。
- 64 节点限制适用于每个 RWX NVMe 卷。
- 其他访问模式和其他协议不受影响。

控制器可扩展性

Trident 通过提高多个存储驱动程序之间的并发性来提升控制器可扩展性。当您启用控制器

可扩展性时，Trident 控制器会并行处理存储操作，而不是对其进行序列化，从而在具有大量并发操作的 Kubernetes 环境中提高吞吐量。

在部署之前，请确定哪些 Trident 驱动程序在正式发布时支持控制器可扩展性，以及哪些驱动程序在 Trident 26.10 中作为技术预览版提供。这有助于您做出明智的部署决策并管理风险。控制器可扩展性默认处于禁用状态。

关键概念和定义

控制器可扩展性

控制器可扩展性是指 Trident 控制器并行处理多个存储操作的能力，而不是在单个锁后对其进行序列化。这些操作包括卷创建、删除和调整大小；快照创建和删除；卷发布和取消发布；以及后端管理。

启用控制器可扩展性时，不同卷和后端上的操作会同时进行。这增加了吞吐量，并减少了具有大量并发 PersistentVolumeClaim 和 VolumeSnapshot 操作的环境中的端到端操作时间。

默认行为（串行模式）

默认情况下，Trident 控制器一次处理一个操作。每个创建、删除、调整大小或快照请求在 Trident 启动下一个之前完成。串行模式是所有安装支持的默认值，无需配置。

串行模式足以满足大多数工作负载的要求。仅当操作量在串行处理下产生积压时，才启用控制器可扩展性。

控制器可扩展性支持

Trident 支持针对不同存储驱动程序的不同成熟度级别的控制器可扩展性。

正式发布

以下驱动程序在 Trident 26.10 正式上市时支持控制器可扩展性：

- `ontap-san`
- `ontap-NAS`
- `ontap-nas-economy.`
- `ontap-nas-flexgroup`
- `ontap-san-economy.`
- `google-cloud-netapp-volumes`
- `azure-netapp-files`
- `solidfire-san`

备注	The `google-cloud-netapp-volumes` 和 `google-cloud-netapp-volumes-san`驱动程序不同。`Trident` 仅支持 `google-cloud-netapp-volumes`。请勿在后端配置或示例中使用 `google-cloud-netapp-volumes-san`。

技术预览

以下驱动程序支持将控制器可扩展性作为 Trident 26.10 的技术预览功能：

- `asa-r2` (SAN 和 NVMe)

此驱动程序具有以下限制：

- 控制器并发性仅可用于评估和测试。
- 行为可能因版本不同而有所变化。
- NetApp 不建议在生产环境中使用。

enableConcurrency 的工作原理

当您将 `enableConcurrency`设置为 `true`时，Trident 会对控制器管理的所有后端应用并发处理。该设置同时应用于每个后端。您不能为单个后端或单个驱动程序启用它。

每个配置的后端都必须使用正式发布或技术预览列表中的驱动程序。如果任何后端使用不受支持的驱动程序，则 Trident 不会启动，并且 Trident 不会添加使用不受支持的驱动程序的后端。

要恢复启动，请删除或重新配置使用不受支持的驱动程序的任何后端，或禁用控制器可扩展性。

启用之前

重要说明	在启用控制器可扩展性之前，请确认每个已配置的后端均使用正式发布或技术预览列表中的驱动程序。如果任何后端使用不受支持的驱动程序，则在启用该功能后，Trident 将无法启动。

使用下表来决定是否启用控制器可扩展性。

如果您的环境	操作
仅使用受支持的驱动程序，并遇到控制器操作积压	启用控制器可扩展性。
使用任何不受支持的驱动程序	保留默认串行模式。不要启用控制器可扩展性。
处理低操作量且无积压	保留默认串行模式。

启用控制器可扩展性

`enableConcurrency` 配置选项控制控制器的可扩展性。在 `Trident` 安装期间或更新现有部署时，必须显式启用此选项。

Trident 操作员部署

要使用 Trident 运算符启用控制器可扩展性，请在 `TridentOrchestrator`自定义资源 (CR) 中将 `enableConcurrency`设置为 `true`。

新安装

创建或编辑 `TridentOrchestrator` CR 并将 `enableConcurrency`设置为 `true`：

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

应用CR：

```
kubectl apply -f tridentorchestrator_cr.yaml
```

现有安装

修补现有 `TridentOrchestrator` CR 以启用控制器可扩展性：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

验证 Trident 是否应用了设置：

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm 部署

要使用 Helm 启用控制器可扩展性，请将 `enableConcurrency`值设置为 `true`。

新安装

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

现有安装

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

或者，在自定义 `values.yaml` 文件中将 `enableConcurrency` 设置为 `true`：

```
# values.yaml
enableConcurrency: true
```

然后使用 values 文件安装或升级：

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl 部署

要使用 `tridentctl` 启用控制器可扩展性，请在安装过程中传递 `--enable-concurrency` 标志。

新安装

```
tridentctl install -n trident --enable-concurrency
```

现有安装

要在现有 `tridentctl`部署上启用控制器可扩展性，请卸载 Trident 并使用以下标志重新安装：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

验证是否已启用控制器可扩展性

启用控制器可扩展性后，验证 Trident 控制器是否在启用并发的情况下运行。检查控制器 Pod 日志：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

输出包含一个确认启用并发的日志条目。

并发行为

启用控制器可扩展性后，Trident 控制器将应用以下行为：

- Trident 将单个全局锁替换为细粒度的按资源锁定。
- Trident 对修改同一资源的操作进行序列化，以保持数据一致性。
- 仅从资源读取的操作与该资源上的其他读取操作同时进行。
- Trident 将并发 ONTAP API 请求限制为每个管理 LIF 20 个，以防止后端存储系统过载。

- 如果多个后端共享相同的管理 LIF，则它们共享此 20 个请求的限制。

注意事项和限制

以下注意事项适用于 Trident 26.10 中的控制器可扩展性：

- 控制器可扩展性仅支持一般可用性和技术预览列表中的驱动程序。有关详细信息，请参见 启用之前。
- Trident 控制器在内部管理并发。此版本不提供用户可配置的并发限制。
- 整体吞吐量取决于正在使用的存储驱动程序、后端响应能力和 Kubernetes API 服务器性能。
- 高并发可能会增加后端存储系统的负载。
- 控制器的可扩展性行为在所有驱动程序中并不相同。
- 技术预览驱动程序在高负载下可能表现出不一致的性能，并且可能在不同版本之间改变行为。
- 由于并行执行，调试并发操作可能会更加复杂。指标和日志可以显示交错操作输出。

建议

在启用控制器可扩展性之前，请完成以下步骤：

1. 确认每个配置的后端都使用一般可用性或技术预览列表中的驱动程序。
2. 在将更改应用于生产之前，请先在非生产集群中测试更改。
3. 应用更改后，验证是否已启用控制器可扩展性。

使用控制器可扩展性操作时，请应用以下一般建议：

- 对于需要高可扩展性的生产环境，请使用正式发布版驱动程序。
- 仅在非生产环境中评估技术预览驱动程序。
- 大规模运行时监控后端和控制器性能。
- 请勿在自动化脚本中假设操作顺序。

自动卷扩展

自动卷扩展使 Trident 配置的持久卷能够在使用的容量达到定义的阈值时自动增长。此功能可降低运营开销，并有助于防止因容量耗尽而导致的应用程序中断。使用 Autogrow Policies 实现自动卷扩展。Autogrow Policy 定义：

- 触发扩展的利用率阈值
- 卷增长的量
- 卷可以达到的最大大小

Trident Autogrow 策略扩展了 PVC。这与 "ONTAP 卷自动调整大小" 不同，后者会扩展底层 FlexVol，而不改变 PVC 大小。

备注	当超过定义的利用率阈值时，卷的大小会自动增加。卷永远不会自动缩减。

要求

在配置自动卷扩展之前，请确保满足以下要求：

- Trident 26.02 或更高版本
- 用于创建 `TridentAutogrowPolicy`自定义资源的基于角色的访问控制权限
- StorageClasses 配置为 `allowVolumeExpansion: true`
- 支持的 ONTAP 协议：
  - 网络文件系统 (NFS)
  - Internet 小型计算机系统接口 (iSCSI)
  - 非易失性内存快速通道 (NVMe)
  - 光纤通道协议 (FCP)

限制

- 早于 ONTAP 9.16.1 的 ONTAP Non-Volatile Memory Express 原始块卷不支持自动扩展。
- 对于存储区域网络卷，如果 `growthAmount`小于或等于 `50 mebibytes`，则 `Trident` 会在调整大小之前自动将值增加到 `51 mebibytes`，前提是生成的大小不超过 `maxSize`。
- 在棕地环境中，由于卷发布迁移行为，某些现有卷可能无法自动扩展。
- 当卷达到 `maxSize`时，不会发生进一步的扩展。
- 支持自动卷扩展的协议：
  - 网络文件系统 (NFS)
  - Internet 小型计算机系统接口 (iSCSI)
  - 非易失性内存快速通道 (NVMe)
  - 光纤通道协议 (FCP)

使用自动增长策略配置卷

可以在两个级别配置自动增长策略：

- 存储类级别：为所有卷设置默认值（使用注释）
- PVC 级别：覆盖存储类默认值（使用注释）

创建 **Autogrow** 策略

当卷达到定义的容量阈值时，Autogrow Policies 启用自动卷扩展。

请确保您拥有：

- 已安装 Trident 26.02 或更高版本
- 基于角色的访问控制权限以创建 `TridentAutogrowPolicy` 资源
- 了解工作负载增长需求

Autogrow Policy 定义了卷在达到定义的容量阈值时如何自动扩展。

您可以在工作流程中随时创建 Autogrow Policies：

- 在创建 StorageClasses 和卷之前
- 在 StorageClasses 存在后
- 配置卷后

这种灵活性使您能够引入自动扩展，而无需重新创建现有资源。

Autogrow 策略规范

Autogrow Policies 是 Kubernetes 自定义资源，定义如下：

字段	Description	格式。	Required	示例	Default
name	唯一策略标识符	string	是的。	production-db-policy	无
usedThreshold	触发扩展的容量百分比	百分比字符串	是的。	"80%"	无
growthAmount	达到阈值时要增长的金额	百分比或大小	否	"10%" 或 "5Gi"	"10%"
maxSize	最大卷大小限制	Kubernetes 数量	否	"500Gi"	无限制

创建 Autogrow 策略

步骤

1. 创建定义自动增长策略的 YAML 文件：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. 将策略应用于您的集群：

```
kubectl apply -f autogrow-policy.yaml
```

3. 验证策略是否已创建：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

预期输出

```
NAME                USED THRESHOLD   GROWTH AMOUNT   STATE
standard-autogrow   80%              10%             Success
```

策略状态

创建策略后，Trident 验证规范并分配以下状态之一：

状态	Description	需采取行动
成功	策略已验证并可供使用。	无。
失败	检测到验证错误。	查看并修复规范。
正在删除	正在执行删除。	等待完成。

将策略与 **StorageClass** 关联

您可以使用 `trident.netapp.io/autogrowPolicy`批注将自动增长策略与StorageClass关联起来。从该StorageClass配置的所有卷都会继承该策略。

> 备注 StorageClass 必须具有 `allowVolumeExpansion: true`。

步骤

1. 使用 Autogrow Policy 注释创建或修改 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. 应用 StorageClass：

```
kubectl apply -f storageclass.yaml
```

3. 验证标注：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
production-db-policy
```

策略优先级

当在 StorageClass 和 PVC 上设置 Autogrow Policy 注释时，Trident 应用以下优先级规则：

1. *PVC 注释优先。*如果 PVC 设置 `trident.netapp.io/autogrowPolicy`，则始终使用该值。
2. **StorageClass** 注释仅适用于 **PVC** 没有注释的情况。
3. 如果两者都没有注释， 则不会应用 Autogrow Policy。

StorageClass 标注	**PVC** 标注	有效行为
`trident.netapp.io/autogrowPolicy: standard-agp`	未设置	用途 `standard-agp`。
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: logs-policy`	使用 `logs-policy`（PVC 覆盖 StorageClass）。
`trident.netapp.io/autogrowPolicy: standard-agp`	`trident.netapp.io/autogrowPolicy: "none"`	无自动增长策略（PVC 禁用自动增长）。
未设置	`trident.netapp.io/autogrowPolicy: dev-policy`	用途 `dev-policy`。
未设置	未设置	无自动增长策略。

配置示例

以下示例显示了不同用例的常见 Autogrow Policy 配置。

生产数据库的保守策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
```

```
  usedThreshold: "75%"
  growthAmount: "20%"
  maxSize: "5Ti"
```

具有固定增长增量的日志存储

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

具有默认值的最小策略

当您忽略 `growthAmount`和 `maxSize`时，Trident 使用默认值(`10%`增长，无限大小）：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

具有自定义 **maxSize** 和默认 **growthAmount** 的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

无限制 **maxSize** 的激进增长

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
```

```
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

具有分数百分比的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```

备注 支持分数百分比。如果指定的小数位数超过三位，Trident 会将值舍入到小数点后三位。

具有 **Autogrow** 的 **NAS StorageClass**

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass 使用 **Autogrow**

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
```

```
allowVolumeExpansion: true
```

配置 ONTAP 卷自动调整大小

配置 Trident 以在支持持久卷的 FlexVol 卷上启用 ONTAP 卷自动调整大小。当已用空间达到配置的阈值时，ONTAP 会自动将 FlexVol 增加到最大大小。这为快照和存储开销提供了额外的容量，而无需手动调整 FlexVol 的大小。

ONTAP volume autosize 仅增加后备 FlexVol 的大小；它不会更改 PersistentVolumeClaim（PVC）的大小。这与 "Trident 自动增长策略" 不同，后者会扩展 PVC。

Trident 在创建、克隆或调整卷大小，以及导入其管理的卷时，会应用后端自动调整大小设置。

要求

- Trident 26.10 或更高版本。
- 以下驱动程序之一：
  - `ontap-san` 使用 iSCSI、NVMe 或光纤通道协议（FCP）。
  - `ontap-san-economy` 使用 iSCSI。
- 本地 ONTAP：
  - 对于使用 ONTAP REST API 的后端，需要 ONTAP 9.12.1 或更高版本。NVMe 后端必须使用 ONTAP REST API。
  - 对于使用 ONTAPI (ZAPI) 的后端，Trident 支持的任何 ONTAP 版本。请参阅 "Trident要求"。

限制

- ONTAP 卷自动调整大小不会缩小 FlexVols。
- 自动调整大小是按后端配置的。创建单独的后端，以便为不同的工作负载使用不同的设置。
- ONTAP NAS 驱动程序、ASA r2 系统、NetApp AFX 或其他分解的 ONTAP 系统，或者 Google Cloud NetApp Volumes 不支持 ONTAP 卷自动调整大小。这些配置的后端创建失败。
- ONTAP 卷自动调整大小不支持 TridentMirrorRelationship 复制和 MetroCluster 配置。Trident 不会阻止这些配置。
- 对于 `ontap-san-economy`， `maximumSize`必须是百分比。
- 对于 `ontap-san`，绝对 `maximumSize`充当大小限制。如果底层 FlexVol 必须超过该限制，则创建或扩展 PVC 将失败。
- 一个 `ontap-san-economy` FlexVol 可以包含多个 PVC 的 LUN。每个卷操作都会将当前后端的自动调整大小设置应用于整个 FlexVol。如果多个后端共享 FlexVols，则由最近的操作确定其设置。对于 `ontap-san-economy`，仅配置 `mode: grow`并省略 `growThresholdPercent`和 `maximumSize`，以便所有后端都使用相同的默认值。

ONTAP 卷自动调整大小选项

在后端配置的顶级 `ontapAutosize`块中设置这些选项。

选项	Description	格式。	Required	当 `ontapAutosize`存在时的默认值	示例
`mode`	启用或禁用自动调整大小。 `grow`启用自动调整大小； `off`禁用它。	string	否	`off`	`grow` 或 `off`
`growThresholdPercent`	ONTAP 扩展 FlexVol 时的已用空间百分比。有效值介于 2 到 99 之间。	整型	否	`85`	`90`
`maximumSize`	最大 FlexVol 大小，以其大小的百分比或绝对大小表示。仅对 `ontap-san`支持绝对大小。	百分比或 Kubernetes 数量	否	`"20%"`	`"20%"` 或 `"2Ti"`

当 `ontapAutosize`块存在时，默认值适用。如果忽略该块，Trident 不会更改 FlexVols 上的自动调整大小设置。

Trident 将百分比 `maximumSize`转换为绝对大小，并在每次调整卷大小时重新计算。例如，使用 `maximumSize: "20%"`，100 GiB 的 FlexVol 可以增长到 120 GiB。

配置 **ONTAP** 卷自动调整大小

1. 将 `ontapAutosize`块添加到后端配置：

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc-ontap-san
  namespace: trident
spec:
  version: 1
  storageDriverName: ontap-san
  backendName: ontap-san-autosize
  managementLIF: 10.0.0.1
  svm: svm_san
  sanType: iscsi
  ontapAutosize:
    mode: grow
    growThresholdPercent: 85
    maximumSize: "20%"
  credentials:
    name: backend-tbc-ontap-san-secret
```

2. 创建后端：

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

3. 从此后端配置卷后，请验证应用于其 Trident 卷的自动调整大小设置：

```
kubectl get tridentvolume <pv-name> -n trident \
  -o jsonpath='{.config.ontapAutosizeMode}
{.config.ontapAutosizeGrowThresholdPercent}
{.config.ontapAutosizeMaximumSize}'
```

对于上述配置，预期输出为：

```
grow 85 20%
```

在后端更新自动调整大小设置

您可以通过更新后端来添加、更改或删除 `ontapAutosize` 块。更改不会立即更新现有 FlexVols：

- 对于 `ontap-san`，新卷、克隆和受管导入将使用新设置。现有卷保留其当前设置，包括调整大小时。
- 对于 `ontap-san-economy`，下次 Trident 创建、调整大小、克隆或导入其中的卷时，共享 FlexVol 会收到新设置。

配置示例

`ontap-san` 使用 FCP 和绝对最大大小

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_san",
  "username": "vsadmin",
  "password": "<password>",
  "sanType": "fcp",
  "ontapAutosize": {
    "mode": "grow",
    "growThresholdPercent": 90,
    "maximumSize": "2Ti"
  }
}
```

`ontap-san-economy`.

此示例仅设置 `mode`，因此 Trident 使用 `growThresholdPercent`和 `maximumSize`的默认值：

```
{
```

```
    "version": 1,
    "storageDriverName": "ontap-san-economy",
    "managementLIF": "10.0.0.1",
    "svm": "svm_san",
    "username": "vsadmin",
    "password": "<password>",
    "ontapAutosize": {
      "mode": "grow"
    }
}
```

常见问题解答

我可以将 **ONTAP** 卷自动调整大小与 **Trident** 自动增长策略一起使用吗?

是的。Trident Autogrow 策略会扩展 PVC，并且 Trident 会在每次调整大小时重新计算 ONTAP autosize 最大值。如果将绝对 `maximumSize`与 `ontap-san`结合使用，请将策略的 `maxSize`设置得足够低，以使底层 FlexVol 保持在该限制范围内。FlexVol 比 PVC 大，因为 Trident 为快照预留和元数据增加了空间。

省略 **`ontapAutosize`**和设置 **`mode: off`**之间有什么区别?

如果省略 `ontapAutosize`，Trident 不会更改 ONTAP FlexVols 上的自动调整大小设置。如果设置 `mode: off`，Trident 会在其创建或更新的 FlexVols 上禁用自动调整大小功能，包括从其源继承了自动调整大小设置的克隆。

为什么我的 **PVC** 扩展失败并出现 **`ontapAutosize maximum … is below required FlexVol size`** 错误?

该卷使用绝对 `maximumSize`，并且扩展的 PVC 需要更大的 FlexVol。 `ontap-san` 卷保留其创建时的最大大小，因此更新后端不会更改现有卷上的设置。对于新卷，请增加 `maximumSize`或使用百分比。

为什么卷创建失败并出现自动调整大小错误?

ONTAP 拒绝了自动调整大小设置，并且 PVC 事件包含来自 ONTAP 的原因。例如，最大大小可能超过 ONTAP 系统支持的最大 FlexVol 大小。在后端降低 `maximumSize`；Trident 在下次尝试创建卷时将使用更新后的值。

相关信息

- "使用 Trident Autogrow Policies 自动扩展卷"
- "ONTAP 命令参考：volume autosize"

管理 Autogrow 策略

创建 Autogrow Policies 后，可以查看、更新和删除它们。您还可以监视使用给定策略的卷。

查看 **Autogrow** 策略

列出所有策略

使用 `kubectl` 列出集群中的所有 Autogrow Policies：

```
kubectl get tridentautogrowpolicy
```

或者，也可以使用 `tridentctl`：

```
tridentctl get autogrowpolicy
```

查看策略详细信息

要查看策略的完整规范和状态：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

要以 YAML 格式查看策略及其关联的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

更新 **Autogrow** 策略

您可以修改现有策略以更改其阈值、增长量或最大大小。更改将立即对使用该策略的所有卷生效。

重要说明	更改会影响使用此策略的所有卷。如果可能，请首先在非生产环境中测试更改。

步骤

1. 编辑策略：

   ```
   kubectl edit tridentautogrowpolicy production-db-policy
   ```

2. 根据需要修改 `spec`字段：

   ```
   spec:
     usedThreshold: "75%"     # Changed from 80%
     growthAmount: "20%"      # Changed from 10%
     maxSize: "1Ti"           # Changed from 500Gi
   ```

3. 保存并退出。更改立即生效。

更新注意事项

- 即时效果： 所有使用该策略的卷将在下一次增长评估时采用新参数。
- *无需重新启动卷：*更改适用于下一次增长操作。
- 先测试： 尽可能在非生产环境中验证更改。
- *沟通更改：*修改共享策略时通知团队。

删除 **Autogrow** 策略

Autogrow Policies 使用终结器保护，以防止在卷主动使用时意外删除。

步骤

1. 删除策略：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. 如果卷仍在使用策略，则删除将进入 `Deleting`状态。检查受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 从每个受影响的卷中删除策略。从下列选项中选择一项：

   - 选项 **A**：通过将注释设置为 `**"none"**`来显式禁用自动增长：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

   - 选项 **B**：完全删除注释：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

删除行为

场景	行为
没有卷使用该策略	策略将立即删除。
卷使用此策略	策略进入 `Deleting`状态。终结器阻止完成，直到删除所有卷。
所有卷均已从策略中删除	终结器将被删除，策略也将被删除。

监控 Autogrow 策略使用情况

使用策略检查卷

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

查找卷使用的策略

```
kubectl get pvc database-pvc -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

监控策略事件

```
kubectl get events --field-selector
involvedObject.kind=TridentAutogrowPolicy
```

支持的协议

Autogrow 支持以下存储协议：

- NFS
- iSCSI
- FCP
- NVMe

备注	对于 SAN 卷，如果配置的 `growthAmount`为 `50 MiB` 或更少，`Trident` 会自动将调整大小操作的增长量增加到 `51 MB`，前提是结果大小不超过 `maxSize`。

支持的后端

Autogrow 支持 ONTAP 后端。Autogrow 还支持 NAS 和 SAN 卷的 Google Cloud NetApp Volumes (GCNV)。

已知限制

- **ONTAP NVMe** 原始块卷： 使用早于 9.16.1 的 ONTAP 版本创建的卷不支持自动增长。
- 现有卷（棕地部署）： 即使应用了有效的自动增长策略，自动增长可能也不适用于现有卷。此限制是由于卷发布的持续迁移所致。要确认迁移已完成，请检查 Trident 控制器日志中的 `"Migration completed"`消息。

常见问题解答

Trident 何时评估阈值?

Trident 持续监控卷使用情况。当使用的容量超过 `usedThreshold`时，Trident 会创建一个内部调整大小请求，并按配置的 `growthAmount`扩展卷。

例如，此策略将以 80% 的容量触发扩展，并每次将卷增长 10%，最多可达 500 GiB：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

是否可以在已配置卷后应用策略?

可以。您可以随时创建 Autogrow Policy，并通过添加或更新 `trident.netapp.io/autogrowPolicy`注释将其应用于现有 PVC。您无需重新创建 PVC 或 StorageClass。

将策略应用于现有 PVC：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

要将策略应用于现有 StorageClass：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

如果我同时在 **StorageClass** 和 **PVC** 上设置自动增长策略，会发生什么?

PVC 注释始终优先。如果 PVC 具有 `trident.netapp.io/autogrowPolicy`注释，则无论StorageClass指定什么，Trident 都会使用该值。有关详细信息，请参见 "策略优先级"。

例如，假设此 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

此 PVC 覆盖 StorageClass 策略：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Trident 使用 `logs-policy`用于 `database-pvc`，而不是 `standard-agp`。

如何禁用特定卷的自动增长？

将 PVC 注释设置为 `"none"`。此设置将覆盖该卷的任何 StorageClass 级别策略：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

您可以验证是否已禁用自动增长：

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
none
```

当卷达到 **maxSize** 时会发生什么？

Trident 停止扩展该卷。即使使用量继续增加超过 `usedThreshold`，它也不会为该卷创建进一步的调整大小请求。

例如，使用此策略，Trident 在达到 100 GiB 后停止扩展卷：

```
apiVersion: trident.netapp.io/v1
```

```
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

要允许无限增长，请忽略 `maxSize` 或将其设置为 `0`：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

我可以在不重新启动卷的情况下更改策略吗?

可以。更新策略时，使用该策略的所有卷将在下一次增长评估时采用新参数。不需要重新启动卷。

要就地更新策略：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

根据需要修改字段：

```
spec:
  usedThreshold: "75%"     # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

保存并退出。验证更新后的策略：

```
kubectl get tridentautogrowpolicy production-db-policy
```

预期输出

```
NAME                   USED THRESHOLD   GROWTH AMOUNT   STATE
production-db-policy   75%              20%             Success
```

为什么我的策略处于失败状态?

`Failed` 状态表示策略规范包含验证错误。运行以下命令来查看错误详细信息：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

常见原因包括无效的 `usedThreshold`（必须为 1–99%）、 `growthAmount` 超过 `maxSize`，或无效的 Kubernetes 数量格式。请更正规范并重新应用：

```
kubectl apply -f autogrow-policy.yaml
```

为什么我无法删除策略?

策略使用终结器保护。如果卷仍在使用该策略，则删除将进入 `Deleting`状态并等待您从策略中删除所有卷。

识别受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

然后从每个 PVC 中删除注释：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

删除所有卷后，Trident 释放终结器并删除此策略。

Autogrow 支持哪些后端?

Autogrow 支持 ONTAP 后端。它还支持 NAS 和 SAN 卷的 Google Cloud NetApp Volumes (GCNV)。

Autogrow 支持 NFS、iSCSI、FCP 和 NVMe 协议。NVMe 原始块卷需要 ONTAP 9.16.1 或更高版本。

棕色字段部署中的现有卷可能需要在自动扩展生效之前完成卷发布迁移。要验证迁移状态，请检查 Trident 控制器日志：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration
```

```
completed"
```

以下 StorageClass 示例显示为 NAS 和 SAN 后端配置的自动增长：

NAS 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 卷的最低增长量是多少?

对于 SAN 卷，有效最低增长量为 51 MB。如果将 `growthAmount`配置为 50 MiB 或更少，Trident 会自动将调整大小操作的增长增加到 51 MB。

例如，此策略设置 `growthAmount`的 `"40Mi"`，但 Trident 对使用它的任何 SAN 卷应用 51 MB 增长：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
```

```
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

要避免这种自动调整，请将 `growthAmount`设置为大于 50 MiB 的值：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

导入 **Trident** 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

您可以将卷导入到Trident中、以便：

- 将应用程序容器化并重复使用其现有数据集
- 对一个应用程序使用数据集的克隆
- 重建发生故障的Kubrenetes集群
- 在灾难恢复期间迁移应用程序数据

注意事项

导入卷之前、请查看以下注意事项。

- Trident只能导入RW (读写)类型的ONTAP卷。DP (数据保护)类型的卷是SnapMirror目标卷。在将卷导入Trident之前、应先中断镜像关系。
- 我们建议导入没有活动连接的卷。要导入当前使用的卷、请克隆此卷、然后执行导入。

警告	这对于块卷尤其重要、因为Kubnetes不会意识到先前的连接、并且可以轻松地将活动卷连接到Pod。这可能会导致数据损坏。

- 虽然 `StorageClass`必须在PVC上指定、但Trident在导入期间不使用此参数。创建卷期间会使用存储类根据存储特征从可用池中进行选择。由于卷已存在、因此导入期间不需要选择池。因此、即使卷位于与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。存储驱动程序导入卷后，系统将创建 PV ，并为其创建一个 Claims Ref 。
  - 回收策略最初设置为 `retain` 在PV中。Kubernetes 成功绑定 PVC 和 PV 后，将更新回收策略以匹配存储类的回收策略。

- 如果 Storage Class 的回收策略为 `delete`，Trident 会在删除 PV 时删除存储卷。

- 默认情况下，Trident 管理 PVC 并重命名 FlexVol 卷以及后端上的 LUN 或 qtree。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`标志表明， Trident在对象的生命周期内不会对 PVC 或 PV 执行任何额外的操作。删除 PV 时，存储卷不会被删除，其他操作（如卷克隆和卷调整大小）也会被忽略。
  - `--no-rename`* - 如果使用 `--no-rename` 标志，Trident 将保留现有卷名并管理卷的生命周期。此选项仅支持 `ontap-nas`、 `ontap-san`（包括 ASA r2 系统）、 `ontap-san-economy` 和 `ontap-nas-economy`（qtree 导入）驱动程序。

提示	如果您想使用 Kubernetes 进行容器化工作负载，但又想在 Kubernetes 之外管理存储卷的生命周期，那么这些选项非常有用。

- PVC 和 PV 中会添加一个标注，用于指示卷已导入以及 PVC 和 PV 是否已管理。不应修改或删除此标注。

导入卷

您可以使用 `tridentctl import`或通过创建带有 Trident 导入注释的 PVC 来导入卷。

备注	如果使用 PVC 注释，则无需下载或使用 `tridentctl`来导入卷。

使用 **tridentctl**

步骤

1. 创建一个 PVC 文件（例如， `pvc.yaml`），该文件将用于创建 PVC。PVC 文件应包括 `name`、`namespace`、 `accessModes`和 `storageClassName`。或者，您可以在 PVC 定义中指定 `unixPermissions`。

   以下是最低规格示例：

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

备注	仅包括所需的参数。PV 名称或卷大小等附加参数可能导致导入命令失败。

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

```
tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-file>
```

使用 **PVC** 注释

步骤

1. 使用所需的 Trident 导入注释创建 PVC YAML 文件（例如， `pvc.yaml`）。PVC 文件应包括：
   - `name` 和 `namespace` 在元数据中
   - `accessModes`、 `resources.requests.storage` 和 `storageClassName` 在规格中
   - 标注：
     - `trident.netapp.io/importOriginalName`: 后端上的卷名称
     - `trident.netapp.io/importBackendUUID`：卷存在的后端 UUID
     - `trident.netapp.io/notManaged` (*Optional*)：为非托管卷设置为 `"true"`。默认值为 `"false"`。

     下面是导入托管卷的示例规范：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

2. 将 PVC YAML 文件应用到 Kubernetes 集群：

```
kubectl apply -f <pvc-file>.yaml
```

Trident 将自动导入卷并将其绑定到 PVC。

示例

查看以下卷导入示例、了解受支持的驱动程序。

ONTAP NAS和ONTAP NAS FlexGroup

Trident支持使用和 `ontap-nas-flexgroup`驱动程序导入卷 `ontap-nas。

备注

- 要使用 `ontap-nas-economy`驱动程序导入 qtree，请参阅导入 NFS qtree。当 Trident 导入 qtree 时，它也会将父 FlexVol 作为包含 qtree 的存储桶导入。必须单独导入 qtree。
- 。 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名称。

使用驱动程序创建的每个卷 `ontap-nas`都是ONTAP集群上的一个FlexVol volume。使用驱动程序导入FlexVol卷的 `ontap-nas`工作原理相同。可以将ONTAP集群上已存在的FlexVol卷作为PVC导入 `ontap-nas。同样、FlexGroup vols也可以作为PVC导入 `ontap-nas-flexgroup。

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。

受管卷

以下示例将导入名为的卷 `managed_volume` 位于名为的后端 `ontap_nas`：

```
tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
|                    NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

非受管卷

使用参数时 `--no-manage`、Trident不会重命名卷。

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

```
tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-
file> --no-manage

+------------------------------------------+---------+---------------
+----------+------------------------------------------+--------+---------+
```

```
|                    NAME                    |  SIZE   | STORAGE CLASS |
PROTOCOL |              BACKEND UUID              | STATE  | MANAGED |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+----------------------------------------+---------+---------------
+----------+----------------------------------------+--------+---------+
```

使用 **PVC** 注释的 **ONTAP NAS** 示例

以下示例演示如何使用 PVC 注释导入托管和非托管卷。

受管卷

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>
```

导入 **qtree**

备注	Qtree 导入是 Trident 26.10 中的技术预览功能。技术预览功能不支持用于生产环境。

您可以使用 `ontap-nas-economy`驱动程序将现有的 ONTAP qtree 导入到 Trident 中。通过 `flexvol/qtree`格式的 FlexVol volume 和 qtree 名称，以及包含该 qtree 的后端的 UUID 来标识 qtree。[概述和注意事项]中的一般导入注意事项也适用于 qtree 导入。

使用 **tridentctl** 导入 **qtree**

以 `flexvol/qtree` 格式将 qtree 指定为卷名称。以下示例从后端 `ontap_nas_economy` 上的 FlexVol volume `flexvol1` 导入 qtree `qtree1`：

```
tridentctl import volume ontap_nas_economy flexvol1/qtree1 -f <path-to-
pvc-file>
```

要导入 qtree 而不重命名它，请添加 `--no-rename`标志。要导入 qtree 而不让 Trident 管理其生命周期，请添加 `--no-manage`标志。

使用 **PVC** 注释导入 **qtree**

使用以下注释创建 PVC：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "flexvol1/qtree1"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
```

```
    - ReadWriteMany
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

要保留现有 FlexVol 卷和 qtree 名称，请添加 `trident.netapp.io/importNoRename: "true"`注释。要将 qtree 导入为非托管，请添加 `trident.netapp.io/notManaged: "true"`注释。

命名和管理行为

托管导入(默认)

Trident 重命名 qtree 以匹配 PVC 名称，并将 FlexVol 卷重命名为 `ontap-nas-economy`命名约定。接合路径不变。

--no-rename

Trident 保留现有的 FlexVol volume 和 qtree 名称。名称必须已遵循 Trident 命名约定，否则 Trident 无法管理 qtree 的删除或挂载操作。

--no-manage

Trident 引用 qtree 但不管理其生命周期。名称必须遵循 Trident 命名约定，Trident 不使用或管理 qtree 的自动导出策略。

导出策略行为

如果后端使用自动导出策略，则导入的 qtree 使用每 qtree 导出策略模型，其中每个 qtree 都有自己的导出策略。Trident 在导入时为 qtree 分配一个空导出策略。在发布时，如果启用了自动导出且现有 FlexVol 导出策略为 `default`或为空，则 Trident 会将 FlexVol 卷更新为每个后端导出策略。

注意	更新 FlexVol 卷导出策略可能会中断当前挂载的同一 FlexVol 卷中的其他 qtree。仅在 FlexVol 卷中的 qtree 已卸载时才导入 qtree。
警告	不要将两个 Kubernetes 集群指向同一个活动 SVM，也不要导入另一个集群仍在管理的 qtree。如果两个集群管理同一个卷，冲突的操作可能会损坏或破坏数据。

ONTAP SAN

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

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

以下示例显示了如何导入托管卷和非托管卷：

受管卷

对于受管卷，Trident会将FlexVol volume重命名为格式，并将FlexVol volume中的LUN重命名 `pvc-<uuid>`为 `lun0`。

以下示例将导入 ontap-san-managed`后端上的FlexVol volume `ontap_san_default：

```
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

以下示例导入 unmanaged_example_volume 在上 ontap_san 后端：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+---------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+---------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+---------+-----------------------------------------+--------+---------+
```

如果您将LUN映射到与Kubornetes节点IQN共享IQN的igroux、如以下示例所示、您将收到错误： `LUN already mapped to initiator(s) in this group`。您需要删除启动程序或取消映射LUN才能导入卷。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

ONTAP SAN-economy 示例

以下示例演示如何为 `ontap-san-economy` 后端导入托管和非托管卷。

受管卷

当您导入托管卷时， Trident 将获得 FlexVol 的所有权并重命名它。当您从同一个 FlexVol 导入多个 LUN 时，您必须考虑此重命名。

以下示例将 `lun1`从FlexVol `toimport`导入为名为 `vol-managed-saneco`的受管卷：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

导入 `lun1`后，`Trident` 将 `FlexVol` 重命名（例如，重命名为 `trident_lun_pool_xyz`）。要从相同的 FlexVol 导入其他 LUN，请使用新的 FlexVol 名称：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```

备注	`ontap-san-economy` 后端一次导入一个 LUN。您可以使用脚本自动执行多个导入。

非受管卷

当您导入非托管卷时，Trident 不拥有 FlexVol。但是，FlexVol 和 LUN 必须遵循 Trident 命名约定。

FlexVol 命名格式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- `STORAGEPREFIX` 是后端配置中 `storagePrefix` 的值。默认值为 `trident`。
- `RANDOMSTRING` 是您选择的任何字符串。

LUN 命名要求

必须为 LUN 命名 `lun0`。

示例

如果您的 `storagePrefix`是 `xyz`，则 LUN 的完整路径为：

```
trident_lun_pool_xyz_randomstring/lun0
```

Element

Trident支持使用驱动程序导入NetApp Element软件和NetApp HCI卷 `solidfire-san`。

备注	Element 驱动程序支持重复的卷名称。但是、如果存在重复的卷名称、Trident将返回错误。作为临时解决策、克隆卷、提供唯一的卷名称并导入克隆的卷。

以下示例将导入 `element-managed` 后端上的卷 `element_default`。

```
tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
|                   NAME                 |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+----------------------------------------+--------+---------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident支持使用驱动程序导入卷 `azure-netapp-files`。

备注	要导入Azure NetApp Files卷、请按卷路径确定该卷。卷路径是卷的导出路径的一部分、位于之后 `:/`。例如、如果挂载路径为 `10.0.0.2:/importvol1`、卷路径为 `importvol1`。

以下示例将导入 `azure-netapp-files` 后端上的卷 `azurenetappfiles_40517` 卷路径 `importvol1`。

```
tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident
```

```
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

Google Cloud NetApp卷

Trident支持使用驱动程序导入卷 `google-cloud-netapp-volumes`。

以下示例在后端 `backend-tbc-gcnv1` 上导入卷 `testvoleasiaeast1`。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+-----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+-----------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

以下示例将在两个卷位于同一区域时导入 `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 **Trident** 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地将其映射到其各自的Kubnetes资源(PVC)。您还可以在后端级别定义用于创建自定义卷名称和自定义标签的模板；您创建、导入或克隆的任何卷都将遵循这些模板。

开始之前

可自定义的卷名称和标签支持：

- 卷创建、导入和克隆操作。
- 就此而言 `ontap-nas-economy` 驱动程序中，只有 Qtree 卷的名称符合名称模板。
- 就此而言 `ontap-san-economy` 驱动程序中，只有 LUN 名称符合名称模板。

限制

- 自定义卷名称仅与ONTAP本地驱动程序兼容。
- 仅对以下情况支持自定义标签： `ontap-san`， `ontap-nas`， 和 `ontap-nas-flexgroup` 司机。
- 自定义卷名称不适用于现有卷。

可自定义卷名称的关键行为

- 如果因名称模板中的语法无效而导致失败、则后端创建将失败。但是、如果模板应用程序失败、则会根据现有命名约定对卷进行命名。
- 如果使用后端配置中的名称模板来命名卷、则存储前缀不适用。任何所需的前缀值都可以直接添加到模板中。

具有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}}"
  }
}
```

池级别示例

```
{
  "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、只有当现有卷具有特定格式的标签时、才会更新标签。例如：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 对于受管卷导入、卷名称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分区操作符。
4. 如果这些模板不会生成唯一的卷名称、则Trident将附加一些随机字符来创建唯一的卷名称。
5. 如果NAS经济型卷的自定义名称长度超过64个字符、则Trident将根据现有命名约定为卷命名。对于所有其他ONTAP驱动程序、如果卷名称超过名称限制、则卷创建过程将失败。

跨命名空间共享 **Trident NFS** 卷

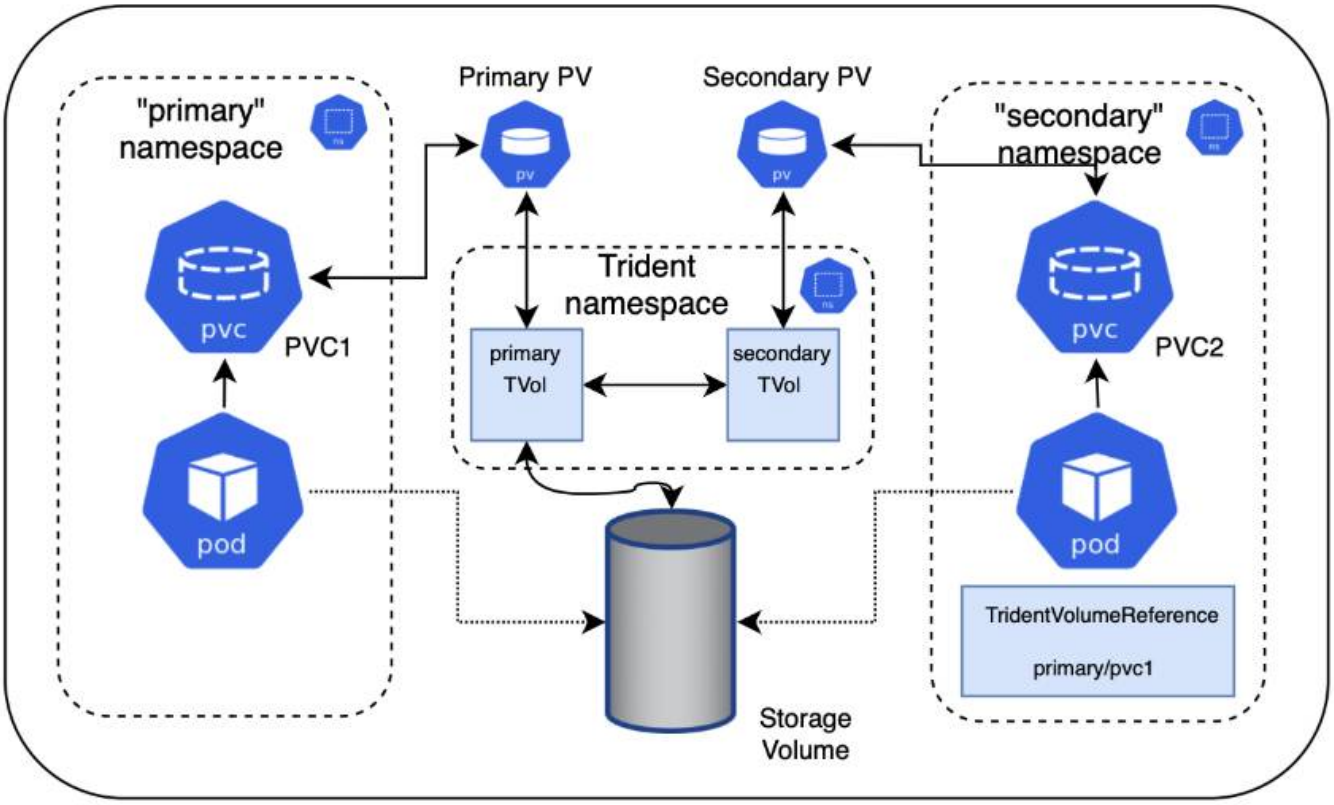
使用Trident、您可以在主命名空间中创建卷、并在一个或多个二级命名空间中共享该卷。

功能

通过TridentvolumeReference CR、您可以在一个或多个Kubernetes名空间之间安全地共享ReadwriteMany (rwx) NFS卷。此Kubernetes本机解决方案 具有以下优势：

- 可通过多个级别的访问控制来确保安全性
- 适用于所有Trident NFS卷驱动程序
- 不依赖于tridentctl或任何其他非本机Kubernetes功能

此图显示了两个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中数据的权限。

2 授予在目标命名空间中创建**CR**的权限

集群管理员向目标命名空间的所有者授予创建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将创建TridentVolumeReference CR以引用源PVC。

4 在目标命名空间中创建从属**PVC**

目标命名空间的所有者创建从属PVC以使用源PVC中的数据源。

配置源和目标命名空间

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。每个步骤都会指定用户角色。

步骤

1. *源命名空间所有者：*创建PVC (`pvc1`)、以授予与目标命名空间共享的权限 (`namespace2`) `shareToNamespace` 标注。

```
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将创建PV及其后端NFS存储卷。

备注

- 您可以使用逗号分隔列表将PVC共享给多个命名空间。例如：`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用共享到所有命名空间 `*`。例如：`trident.netapp.io/shareToNamespace: *`
- 您可以更新PVC以包括 `shareToNamespace` 随时添加标注。

2. *集群管理员：*确保已建立适当的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限。
3. *目标命名空间所有者：*在目标命名空间中创建引用源命名空间的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：*创建PVC (`pvc2`) (`namespace2`) `shareFromPVC` 用于指定源PVC的标注。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

备注	目标PVC的大小必须小于或等于源PVC。

结果

Trident会读取 `shareFromPVC`目标PVC上的标注、并将目标PV创建为其自身没有指向源PV的存储资源的子卷、同时共享源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的卷的访问权限、并保留共享该卷的其他命名空间的访问权限。删除引用卷的所有名称空间后、Trident将删除该卷。

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]
```

flags

- `-h, --help`：卷帮助。
- `--parentOfSubordinate string`：将查询限制为从源卷。
- `--subordinateOf string`：将查询限制为卷的下属。

限制

- Trident无法阻止目标名称空间写入共享卷。您应使用文件锁定或其他进程来防止覆盖共享卷数据。
- 您不能通过删除来撤消对源PVC的访问 `shareToNamespace` 或 `shareFromNamespace` 标注或删除

`TridentVolumeReference` CR.要撤消访问、必须删除从属PVC。

- 无法在从属卷上执行快照、克隆和镜像。

有关详细信息 ...

要了解有关跨命名空间卷访问的详细信息、请执行以下操作：

- 观看上的演示 "NetAppTV"。

跨命名空间克隆 Trident 卷

通过使用Trident、您可以使用同一个Kubornetes集群中不同命名空间的现有卷或卷快照创建新卷。

前提条件

克隆卷之前、请确保源后端和目标后端的类型相同、并且具有相同的存储类。

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

快速入门

只需几个步骤即可设置卷克隆。

1. 配置源**PVC**以克隆卷

源命名空间所有者授予访问源PVC中数据的权限。

2. 授予在目标命名空间中创建**CR**的权限

集群管理员向目标命名空间的所有者授予创建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将创建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进行协作并采取相应操作。每个步骤都会指定用户角色。

步骤

1. **Source命名空间owner:**(pvc1`在源命名空间中创建PVC(`namespace1)，用于(namespace2`使用标注授予与目标命名空间共享的权限。 `cloneToNamespace

```
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将创建PV及其后端存储卷。

备注

- 您可以使用逗号分隔列表将PVC共享给多个命名空间。例如，`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用共享到所有 *`的文件。例如、`trident.netapp.io/cloneToNamespace: *`
- 您可以随时更新PVC以包含 `cloneToNamespace`标注。

2. 集群管理员：确保已建立适当的 RBAC，以授予目标命名空间所有者在目标命名空间中创建 TridentVolumeReference CR 的权限(namespace2）。
3. 目标命名空间所有者： 在目标命名空间中创建一个引用源 PVC 的 TridentVolumeReference CR pvc1。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination命名空间owner:**(pvc2`在目标命名空间中创建PVC(`namespace2)，使用 `cloneFromPVC`或`cloneFromSnapshot`和 `cloneFromNamespace`标注指定源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PV-NAS经济型驱动程序配置的ONTAP、不支持只读克隆。

使用SnapMirror复制卷

Trident支持在一个集群上的源卷与对等集群上的目标卷之间建立镜像关系、以便为灾难恢复复制数据。 您可以使用名为 Trident 镜像关系 (TMR) 的命名空间自定义资源定义 (CRD) 来执行以下操作：

- 在卷之间创建镜像关系(PVC)
- 删除卷之间的镜像关系
- 中断镜像关系
- 在灾难情况下提升二级卷(故障转移)
- 在集群之间执行应用程序无中断过渡(在计划内故障转移或迁移期间)

复制前提条件

开始之前、请确保满足以下前提条件：

ONTAP 集群

- **Kubernet**：使用ONTAP作为后端的源和目标Trident集群上必须存在Trident版本22.10或更高版本。
- 许可证：必须在源和目标ONTAP集群上启用使用数据保护包的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、所有许可证均以NetApp许可证文件(NLG)的形式提供、NLG是一个支持多项功能的文件。有关详细信息、请参见 "ONTAP One附带的许可证" 。

备注	仅支持 SnapMirror 异步保护。

对等

- 集群和**SVM**：ONTAP存储后端必须建立对等状态。有关详细信息、请参见 "集群和 SVM 对等概述" 。

重要说明	确保两个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必须可供目标集群上的Trident使用。

支持的驱动程序

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

备注	ASA r2 系统不支持使用 SnapMirror 进行卷复制。有关 ASA r2 系统的信息，请参阅"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Kubbernetes集群上执行以下步骤：
   a. 使用参数创建StorageClass对象 `trident.netapp.io/replication: true` 。

      示例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 使用先前创建的StorageClass创建PVC。

      示例

      ```
      kind: PersistentVolumeClaim
      apiVersion: v1
      metadata:
        name: csi-nas
      spec:
      ```

```
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-nas
```

c. 使用本地信息创建镜像关系CR。

示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident会提取卷的内部信息以及卷的当前数据保护(DP)状态、然后填充镜像关系的状态字段。

d. 获取TridentMirorRelationship CR以获取PVC的内部名称和SVM。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. 在二级Kubbernetes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. 使用目标和源信息创建镜像关系CR。

      示例

      ```
      kind: TridentMirrorRelationship
      apiVersion: trident.netapp.io/v1
      metadata:
        name: csi-nas
      spec:
        state: established
        replicationPolicy: MirrorAllSnapshots
        replicationSchedule: 5min
        volumeMappings:
        - localPVCName: csi-nas
          remoteVolumeHandle:
      "datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
      ```

      TMR spec 有四个顶级字段： state，用于设置所需的关系状态（请参阅 [卷复制状态]）； volumeMappings，用于将本地 PVC 与其远程卷句柄相关联；以及下面描述的可选 replicationPolicy 和 replicationSchedule 字段。

      ```
      `replicationPolicy` 和 `replicationSchedule` 字段是可选的。将 `replicationPolicy` 设置为 ONTAP 上的异步 SnapMirror 策略的名称，并将 `replicationSchedule` 设置为 ONTAP 计划的名称。如果省略任一字段，Trident 将使用在后端配置的相应复制策略或计划。在创建关系时设置这些字段；建立关系后无法更改策略或计划。要使用不同的设置，请删除并重新创建 TMR。
      ```

Trident 使用配置的复制策略和计划创建并初始化 SnapMirror 关系。

1. 使用先前创建的StorageClass创建一个PVC以用作二级(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将检查是否存在TridentMirorRelationship CRD、如果此关系不存在、则无法创建卷。如果存在此关系、Trident将确保将新FlexVol volume放置到与镜像关系中定义的远程SVM建立对等关系的SVM上。

卷复制状态

三级镜像关系(TCR)是一种CRD、表示PVC之间复制关系的一端。目标T关系 管理器具有一个状态、此状态会告知Trident所需的状态。目标T关系 管理器具有以下状态：

- 已建立：本地PVC是镜像关系的目标卷、这是一个新关系。
- 提升：本地PVC可读写并可挂载、当前未建立任何有效的镜像关系。
- 重新建立：本地PVC是镜像关系的目标卷、以前也位于该镜像关系中。
  - 如果目标卷曾经与源卷建立关系、因为它会覆盖目标卷的内容、则必须使用重新建立的状态。
  - 如果卷之前未与源建立关系、则重新建立的状态将失败。

在计划外故障转移期间提升辅助**PVC**

在二级Kubbernetes集群上执行以下步骤：

- 将TridentMirorRelationship的_spec.state_字 段更新到 `promoted`。

在计划内故障转移期间提升辅助**PVC**

在计划内故障转移(迁移)期间、执行以下步骤以提升二级PVC：

步骤

1. 在主Kubbernetes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 TridentTridentSnapshotInfoCR 以获取内部详细信息。

示例

```
kind: TridentSnapshotInfo
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  snapshotName csi-nas-snapshot
```

a. 在辅助 Kubernetes 集群上，将 *TridentMirrorRelationship* CR 的 *spec.state* 字段更新为 *promoted*，并将 *spec.volumeMappings[].promotedSnapshotHandle* 更新为快照的 internalName。

3. 在二级Kubernetes集群上、确认Trident镜像 关系的状态(stats.state字段)为已提升。

在故障转移后还原镜像关系

在还原镜像关系之前、请选择要用作新主卷的那一端。

步骤

1. 在二级Kubernetes集群上、确保已更新TundentMirorRelationship上的_spic.netVolumeHandle_字段的值。
2. 在辅助 Kubernetes 集群上，将 TridentMirrorRelationship 的 *spec.state* 字段更新为 `reestablished`。

其他操作

Trident支持对主卷和二级卷执行以下操作：

将主**PVC**复制到新的二级**PVC**

确保您已有一个主PVC和一个次要PVC。

步骤

1. 从已建立的二级(目标)集群中删除PerbestentVolumeClaim和TridentMirorRelationship CRD。
2. 从主(源)集群中删除TridentMirorRelationship CRD。
3. 在主(源)集群上为要建立的新二级(目标) PVC创建新的TridentMirorRelationship CRD。

调整镜像、主**PVC**或二级**PVC**的大小

可以正常调整PVC的大小、如果数据量超过当前大小、ONTAP将自动扩展任何目标flevxvol。

从**PVC**中删除复制

要删除复制、请对当前二级卷执行以下操作之一：

- 删除次要PVC上的镜像关系。此操作将中断复制关系。
- 或者、将spec.state字段更新为_promoted_。

删除**PVC (**之前已镜像**)**

Trident会检查是否存在复制的PVC、并在尝试删除卷之前释放复制关系。

删除**TTr**

删除镜像关系一端的T磁 还原会导致剩余的T磁 还原在Trident完成删除之前过渡到_promoted 状态。如果选择删除的*TMirror*已处于*_Promved"*状态、则不存在现有镜像关系、此时*TMirror*将被删除、*Trident*会将本地*PVC*提升为*_ReadWrite*。此删除操作将释放ONTAP中本地卷的SnapMirror元数据。如果此卷将来要在镜像关系中使用、则在创建新镜像关系时、它必须使用具有_re设立_卷复制状态的新TMirror。

在**ONTAP**联机时更新镜像关系

建立镜像关系后、可以随时更新这些关系。您可以使用 `state: promoted` 或 `state: reestablished` 字段更新关系。将目标卷提升为常规ReadWrite卷时、可以使用_promotedSnapshotHandle_指定要将当前卷还原到的特定快照。

在**ONTAP**脱机时更新镜像关系

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TridentAction镜像 更新的示例格式：

示例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` 反映TridentAction镜像 更新CRD的状态。它可以从_suced_、_in Progress _或_failed中获取值。

使用 **CSI** 拓扑

Trident可以通过使用有选择地创建卷并将其连接到Kubbernetes集群中的节点 "CSI 拓扑功能"。

概述

使用 CSI 拓扑功能，可以根据区域和可用性区域将对卷的访问限制为一小部分节点。如今，借助云提供商，Kubernetes 管理员可以生成基于分区的节点。节点可以位于一个区域内的不同可用性区域中，也可以位于不同区域之间。为了便于在多区域架构中为工作负载配置卷、Trident使用CSI拓扑。

提示	了解有关 CSI 拓扑功能的更多信息 "此处"。

Kubernetes 提供了两种唯一的卷绑定模式：

- 将设置为 `Immediate`时，`Trident`创建卷时 `VolumeBindingMode`不具有任何拓扑感知功能。创建 PVC 时会处理卷绑定和动态配置。这是默认设置 `VolumeBindingMode`、适合不强制实施拓扑限制的集群。创建永久性卷时、不会依赖于发出请求的POD的计划要求。
- 如果将 `VolumeBindingMode` 设置为 `WaitForFirstConsuming` ，则为 PVC 创建和绑定永久性卷的操作将延迟，直到计划并创建使用 PVC 的 Pod 为止。这样，卷就会根据拓扑要求强制实施的计划限制来创建。

备注	`WaitForFirstConsumer" 绑定模式不需要拓扑标签。此功能可独立于 CSI 拓扑功能使用。`

您需要的内容

要使用 CSI 拓扑，您需要满足以下条件：

- 运行的Kubernetes集群 "支持的Kubernetes版本"

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 集群中的节点应具有引入拓扑感知的标签 `topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone`）。 `topology.kubernetes.io/region` 标签对于 Trident 中的拓扑感知配置是必需的。在安装 Trident 之前，这些标签应存在于节点上。如果在安装后添加或更改拓扑标签，只需重新启动 Trident 节点 Pod 即可让 Trident 获取这些更改。

```
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"}]
```

第 1 步：创建可感知拓扑的后端

Trident存储后端可以设计为根据可用性区域选择性地配置卷。每个后端都可以包含一个可选 `supportedTopologies` 块、该块代表受支持的分区和区域列表。对于使用此后端的 StorageClasses ，只有在受支持区域 / 区域中计划的应用程序请求时，才会创建卷。

下面是一个后端定义示例：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
supportedTopologies:
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-a
  - topology.kubernetes.io/region: us-east1
    topology.kubernetes.io/zone: us-east1-b
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "supportedTopologies": [
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-a"
    },
```

```
    {
      "topology.kubernetes.io/region": "us-east1",
      "topology.kubernetes.io/zone": "us-east1-b"
    }
  ]
}
```

备注	`supportedTopologies`用于提供每个后端的区域和分区列表。这些区域和分区表示可在StorageClass 中提供的允许值列表。对于包含后端提供的部分区域和分区的StorageClasses、Trident会在后端创建一个卷。

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` 标签表示存储池的位置。`topology.Kubernees.io/zone` 和 `topology.Kubernees.io/zone` 指定存储池的使用位置。

第 2 步：定义可识别拓扑的 StorageClasses

根据为集群中的节点提供的拓扑标签，可以将 StorageClasses 定义为包含拓扑信息。这将确定用作 PVC 请求

候选对象的存储池，以及可使用 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
```

在上述StorageClass定义中， `volumeBindingMode`将设置为 `WaitForFirstConsumer`。在此存储类中请求的 PVC 在 Pod 中引用之前不会执行操作。和 `allowedTopologies`提供了要使用的分区和区域。`StorageClass`会 `netapp-san-us-east1`在上述定义的后端创建`PVC` `san-backend-us-east1`。

第 **3** 步：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：

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
```

```
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

此 podSpec 指示 Kubernetes 在 `us-East1` 区域中的节点上计划 Pod ，并从 `us-East1-a` 或 `us-East1-b` 区域中的任何节点中进行选择。

请参见以下输出：

```
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**

可以使用 `tridentctl backend update` 更新原有后端，以包含 `supportedTopologies` 列表。这不会影

响已配置的卷，并且仅用于后续的 PVC 。

了解更多信息

- "管理容器的资源"
- "节点选择器"
- "关联性和反关联性"
- "损害和公差"

使用 Trident 快照

持久卷(PVs)的Kubbernetes卷快照支持卷的时间点副本。您可以为使用Trident创建的卷创建快照、导入在Trident外部创建的快照、从现有快照创建新卷以及从快照恢复卷数据。

概述

卷快照受支持 `ontap-nas`， `ontap-nas-flexgroup` ， `ontap-san` ， `ontap-san-economy` ， `solidfire-san` ， `azure-netapp-files` ， 和 `google-cloud-netapp-volumes`司机。

开始之前

要使用快照、您必须具有外部快照控制器和自定义资源定义(CRD)。这是Kubernetes流程编排程序(例如：Kubeadm、GKE、OpenShift)的职责。

如果您的Kubernetes分发版不包含快照控制器和CRD、请参见 [部署卷快照控制器]。

> 备注 | 如果在GKE环境中创建按需卷快照、请勿创建快照控制器。GKE-使用内置的隐藏快照控制器。

创建卷快照

步骤

1. 创建 `VolumeSnapshotClass`。有关详细信息，请参见 "VolumeSnapshotClass"。
   - `driver`指向Trident CSI驱动程序。
   - `deletionPolicy` 可以是 `Delete` 或 `Retain`。设置为时 `Retain`、存储集群上的底层物理快照会保留、即使在使用时也是如此 `VolumeSnapshot` 对象已删除。

   示例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   ```

```
deletionPolicy: Delete
```

2. 创建现有PVC的快照。

   示例

   - 此示例将创建现有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
```

   - 此示例将为名为的PVC创建卷快照对象 `pvc1` 快照的名称设置为 `pvc1-snap`。VolumeSnapshot类似于PVC、并与关联 `VolumeSnapshotContent` 表示实际快照的对象。

```
kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

kubectl get volumesnapshots
NAME                    AGE
pvc1-snap               50s
```

   - 您可以确定 `VolumeSnapshotContent` 的对象 `pvc1-snap` VolumeSnapshot的说明。。 `Snapshot Content Name` 标识提供此快照的VolumeSnapshotContent对象。。 `Ready To Use` 参数表示快照可用于创建新PVC。

```
kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
...
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-
525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
```

```
    Name:        pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
...
```

从卷快照创建**PVC**

您可以使用 `dataSource` 使用名为的卷快照创建PVC `<pvc-name>` 作为数据源。创建 PVC 后，可以将其附加到 Pod 上，并像使用任何其他 PVC 一样使用。

警告	PVC将与源卷在同一后端创建。请参见 "知识库文章：无法在备用后端创建从三端PVC Snapshot创建PVC"。

以下示例将使用创建PVC `pvc1-snap` 作为数据源。

```
cat pvc-from-snap.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

导入卷快照

Trident支持通过"Kubbernetes预配置快照过程"、集群管理员可以创建 `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 `VolumeSnapshotContent`引用后端快照的对象。这将在Trident中启动快照工作流。

   - 在中指定后端快照的名称 `annotations` 作为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 在中指定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`。这是调用中 `snapshotHandle`外部快照程序向Trident提供的唯一信息。 `ListSnapshots`。`<volumeSnapshotContentName>` 由于CR命名限制、不能始终与后端快照名称匹配。

备注

   示例

   以下示例将创建 `VolumeSnapshotContent` 引用后端快照的对象 `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. *集群管理员：*创建 `VolumeSnapshot` 引用的CR `VolumeSnapshotContent` 对象。此操作将请求访问以使用 `VolumeSnapshot` 在给定命名空间中。

   示例

   以下示例将创建 `VolumeSnapshot` CR已命名 `import-snap` 引用的 `VolumeSnapshotContent` 已命名 `import-snap-content`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
```

```
  source:
    volumeSnapshotContentName: import-snap-content
```

3. *内部处理(无需执行任何操作)：*外部快照程序识别新创建的 `VolumeSnapshotContent`并运行 `ListSnapshots`调用。`Trident`将创建 `TridentSnapshot`。
   - 外部快照程序用于设置 `VolumeSnapshotContent` to `readyToUse` 和 `VolumeSnapshot` to `true`。
   - TRIdent返回 `readyToUse=true`。
4. *任何用户:*创建一个 `PersistentVolumeClaim` 以引用新的 `VolumeSnapshot`、其中 `spec.dataSource` (或 `spec.dataSourceRef`)名称为 `VolumeSnapshot` 名称。

   示例

   以下示例将创建一个引用的PVC `VolumeSnapshot` 已命名 `import-snap`。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

使用快照恢复卷数据

默认情况下、快照目录处于隐藏状态、以便最大程度地提高使用配置的卷的兼容性 `ontap-nas` 和 `ontap-nas-economy` 驱动程序。启用 `.snapshot` 目录以直接从快照恢复数据。

使用volume Snapshot restore ONTAP命令行界面将卷还原到先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

备注	还原Snapshot副本时、现有卷配置将被覆盖。创建Snapshot副本后对卷数据所做的更改将丢失。

从快照原位还原卷

Trident可使用(TSR) CR从快照快速原位还原卷 `TridentActionSnapshotRestore`。此CR用作要务Kubbernetes操作、在操作完成后不会持久保留。

Trident支持快照恢复 `ontap-san`， `ontap-san-economy` ， `ontap-nas` ， `ontap-nas-flexgroup` ， `azure-netapp-files` ， `google-cloud-netapp-volumes` ， 和 `solidfire-san`司机。

开始之前

您必须具有绑定的PVC和可用的卷快照。

- 验证PVC状态是否已绑定。

```
kubectl get pvc
```

- 确认卷快照已准备就绪、可以使用。

```
kubectl get vs
```

步骤

1. 创建TSR CR。此示例将为PVC和卷快照创建CR `pvc1` `pvc1-snapshot`。

   备注 | TSR CR必须位于PVC和VS所在的命名空间中。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. 应用CR以从快照还原。此示例将从Snapshot恢复 `pvc1`。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

结果

Trident将从快照还原数据。您可以验证快照还原状态：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

备注

- 在大多数情况下、如果出现故障、Trident不会自动重试此操作。您需要再次执行此操作。
- 没有管理员访问权限的Kubbernetes用户可能必须获得管理员授予的权限、才能在其应用程序命名空间中创建TSR CR。

删除具有关联快照的**PV**

删除具有关联快照的永久性卷时、相应的Trident卷将更新为"正在删除"状态。删除卷快照以删除Trident卷。

部署卷快照控制器

如果您的Kubernetes分发版不包含快照控制器和CRD、则可以按如下所示进行部署。

步骤

1. 创建卷快照CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

备注	如有必要、打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` 并更新 `namespace` 命名空间。

相关链接

- "卷快照"
- "VolumeSnapshotClass"

使用 **Trident** 卷组快照

Kubernetes 持久卷 (PV) 卷组快照 NetApp Trident 提供了创建多个卷（一组卷快照）的功能。此卷组快照代表在同一时间点创建的多个卷的副本。

备注	VolumeGroupSnapshot 是 Kubernetes 中的一个 Beta 功能，包含 Beta 版 API 。VolumeGroupSnapshot所需的最低版本为 Kubernetes 1.32。

创建卷组快照

以下存储驱动程序支持卷组快照：

- `ontap-san`驱动程序 - 仅适用于 iSCSI 和 FC 协议，不适用于 NVMe/TCP 协议。
- `ontap-san-economy` - 仅适用于 iSCSI 协议。
- `ontap-NAS`

备注	NetApp ASA r2 或 AFX 存储系统不支持卷组快照。

开始之前

- 确保您的 Kubernetes 版本是 K8s 1.32 或更高版本。
- 要使用快照、您必须具有外部快照控制器和自定义资源定义(CRD)。这是Kubernetes流程编排程序(例如：Kubeadm、GKE、OpenShift)的职责。

  如果您的 Kubernetes 发行版不包含外部快照控制器和 CRD，请参阅[部署卷快照控制器] 。

备注	如果在 GKE 环境中创建按需卷组快照，请不要创建快照控制器。GKE-使用内置的隐藏快照控制器。

- 在快照控制器 YAML 中，设置 `CSIVolumeGroupSnapshot`功能门控设置为“true”，以确保卷组快照已启用。
- 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 之前创建一个 VolumeGroupSnapshotClass。有关详细信息，请参阅 "卷组快照类"。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
  name: csi-group-snap-class
  annotations:
    kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

- 使用现有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。

  以下示例使用以下方式创建 PVC `pvc1-group-snap`作为数据源和标签 `consistentGroupSnapshot: groupA`.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
```

- 创建具有相同标签的 VolumeGroupSnapshot (`consistentGroupSnapshot: groupA`) 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

您可以使用作为卷组快照的一部分创建的各个快照来恢复各个持久卷。您无法将卷组快照作为一个整体进行恢复。

使用volume Snapshot restore ONTAP命令行界面将卷还原到先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

备注	还原Snapshot副本时、现有卷配置将被覆盖。创建Snapshot副本后对卷数据所做的更改将丢失。

从快照原位还原卷

Trident可使用(TSR) CR从快照快速原位还原卷 `TridentActionSnapshotRestore`。此CR用作要务Kubbernetes操作、在操作完成后不会持久保留。

有关详细信息，请参见 "从快照原位还原卷"。

删除与组快照关联的 **PV**

删除组卷快照时：

- 您可以删除整个 VolumeGroupSnapshots，而不是删除组中的单个快照。
- 如果在持久卷存在快照的情况下删除了该持久卷，Trident 会将该卷移至“删除”状态，因为必须先删除快照，然后才能安全删除该卷。
- 如果已使用分组快照创建了克隆，然后要删除该组，则会开始克隆拆分操作，并且在拆分完成之前无法删除该组。

部署卷快照控制器

如果您的Kubernetes分发版不包含快照控制器和CRD、则可以按如下所示进行部署。

步骤

1. 创建卷快照CRD。

```
cat snapshot-setup.sh
```

```
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl
asses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotco
ntents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.
yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-
```

```
controller/setup-snapshot-controller.yaml
```

备注	如有必要、打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` 并更新 `namespace` 命名空间。

相关链接

- "卷组快照类"
- "卷快照"

管理和监控Trident

升级Trident

升级Trident

从24.02版开始、Trident按四个月的发布节奏、每年发布三个主要版本。每个新版本都是在先前版本的基础上构建的、并提供了新功能、性能增强功能、错误修复和改进。我们建议您至少每年升级一次、以利用Trident中的新功能。

升级前的注意事项

升级到最新版本的Trident时、请考虑以下事项：

- 在给定的Kubnetes集群中的所有名称区中只应安装一个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、请务必在"由Trident使用"中 `StorageClasses`提供 `parameter.fsType`。您可以在不中断已有卷的情况下删除和重新创建 `StorageClasses`卷。
  - 这是执行的一项 * 要求 * "安全上下文" SAN 卷。
  - sample输入目录包含https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-basic.yaml.templ等示例[`storage-class-basic.yaml.templ`)和链接：https://github.com/NetApp/trident/blob/master/trident-installer/sample-input/storage-class-samples/storage-class-bronze-default.yaml[`storage-class-bronze-default.yaml`^。
  - 有关详细信息，请参见 "已知问题"。

第**1**步：选择版本

Trident版本遵循基于日期的 `YY.MM`命名约定、其中"YY"是一年的最后两位数字、"MM"是月份。DOT版本遵循 `YY.MM.X`惯例、其中"X"是修补程序级别。您将根据要从中升级的版本选择要升级到的版本。

- 您可以直接升级到已安装版本的四个版本窗口中的任何目标版本。例如、您可以直接从24.06 (或任何24.06 DOT版本)升级到25.06。
- 如果要从四个版本窗口之外的版本进行升级、请执行多步骤升级。按照您要升级的的升级说明升级到适合四个版本窗口的 "早期版本" 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。

第**2**步：确定原始安装方法

要确定最初安装Trident时使用的版本、请执行以下操作：

1. 使用 ... `kubectl get pods -n trident` 检查Pod。
   - 如果没有操作员POD，则使用安装了Trident `tridentctl`。

- 如果有操作员POD、则Trident是使用Trident操作员手动或使用Helm安装的。

2. 如果有操作员POD、请使用 `kubectl describe torc`确定Trident是否是使用Helm安装的。
   - 如果有Helm标签、则Trident是使用Helm安装的。
   - 如果没有Helm标签、则Trident是使用Trident操作员手动安装的。

第**3**步：选择升级方法

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"在安装方法之间切换"。升级 Trident 有两种选择。

- "使用Trident操作符进行升级"

> 提示　我们建议您查看 "了解操作员升级工作流" 在使用操作员进行升级之前。

*

使用操作员升级

了解 **Trident Operator** 升级工作流程

在使用Trident运算符升级Trident之前、您应了解升级期间发生的后台进程。其中包括对支持滚动更新的三项技术控制器、控制器Pod和节点Pod以及节点DemonSet进行的更改。

TRIdent操作员升级处理

安装和升级 Trident 的众多"使用啮合式操作员的优势"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：

- 删除并重新创建三级控制器部署和节点DemonSet。
- 使用新版本更换TRIDent控制器Pod和TRIDent节点Pod。
  - 如果节点未更新、则不会阻止更新其余节点。
  - 只有运行了三项节点Pod的节点才能挂载卷。

> 提示　有关Kubnetes集群上Trident架构的详细信息，请参阅"Trident架构"。

操作员升级工作流

使用三端修复操作符启动升级时：

1. 三端运算符*：
   a. 检测当前安装的Trident版本(版本_n_)。
   b. 更新所有Kubnetes对象、包括CRD、RBAC和三项服务。
   c. 删除版本为_n_的TRIdent控制器部署。
   d. 创建版本为_n+1_的三项控制器部署。

2. *Kubernetes*为_n+1_创建了三项控制器Pod。
3. 三端运算符*：
   a. 删除_n_的三项目标节点演示集。操作员不会等待节点Pod终止。
   b. 为_n+1_创建三项目标节点演示。
4. *Kubernetes*会在未运行三端节点Pod _n_的节点上创建三端节点Pod。这样可以确保一个节点上的任何版本的三端存储节点Pod不会超过一个。

使用Trident Operator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

您可以从集群范围的Trident操作员安装升级到另一个集群范围的Trident操作员安装。所有Trident版本都使用集群范围的操作符。

备注	要从使用命名空间范围的运算符(版本20.07到20.10)安装的Trident升级、请按照Trident的升级说明进行操作"您安装的版本"。

关于此任务

{\f270通过} {\f270 {\f151、} {\f270} {\f270} {\f151、} {\f270} {\f270} {\f151、} {\f270通过} {\f151、} {\f270} {\f270} {\f151、} {\f270}

- 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

确保您使用的是正在运行的Kubbernetes集群 "支持的Kubernetes版本"。

步骤

1. 验证Trident版本：

```
./tridentctl -n trident version
```

2. 更新 `operator.yaml`， `tridentorchestrator_cr.yaml` ， 和 `post_1_25_bundle.yaml`使用您要升级到的版本（例如 25.06）的注册表和图像路径以及正确的密钥。
3. 删除用于安装当前Trident实例的Trident操作员。例如，如果您从 25.02 升级，请运行以下命令：

```
kubectl delete -f 25.02.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

4. 如果您使用自定义了初始安装 `TridentOrchestrator` 属性、您可以编辑 `TridentOrchestrator` 用于修改安装参数的对象。其中可能包括为脱机模式指定镜像Trident和CSI映像注册表、启用调试日志或指定映

像提取密钥所做的更改。

5. 使用适合您环境的正确 bundle YAML 文件安装Trident ，其中 *<bundle.yaml>* 是 `bundle_pre_1_25.yaml`或者 `bundle_post_1_25.yaml`根据您的 Kubernetes 版本。例如，如果您正在安装Trident 25.06.0，请运行以下命令：

```
kubectl create -f 25.06.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

6. 编辑三叉戟项圈以包含图像 25.06.0。

升级**Helm**安装

您可以升级Trident Helm安装。

警告	在将安装了Trident的Kubernetes集群从1.24升级到1.25或更高版本时、您必须 `helm upgrade` 先更新`values.yaml`以设置为或添加到 `true`命令、 `--set excludePodSecurityPolicy=true`然后才能升级集群。 `excludePodSecurityPolicy`

如果您已将Kubnetes集群从1.24升级到1.25、但未升级Trident Helm、则Helm升级将失败。要完成Helm升级、请作为前提条件执行以下步骤：

1. 从安装helm-mapkubeapis插件 https://github.com/helm/helm-mapkubeapis。
2. 在安装了Trident的命名空间中对Trident版本执行演练。此操作将列出要清理的资源。

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. 使用Helm执行完整运行以执行清理。

```
helm mapkubeapis trident  --namespace trident
```

步骤

1. 如果您"已使用Helm安装Trident"，您可以使用 `helm upgrade trident netapp-trident/trident-operator --version 100.2610.0`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. 从下载最新的Trident版本"GitHub上的_assets_部分"。
   b. 请使用 `helm upgrade` 命令，其中 `trident-operator-26.10.1.tgz` 反映要升级到的版本。

```
helm upgrade <name> trident-operator-26.10.1.tgz
```

备注	如果您在初始安装期间设置了自定义选项(例如、为三端映像和CSI映像指定专用、镜像注册表)、请附加 `helm upgrade` 命令 `--set` 为了确保这些选项包含在upgrade命令中、否则这些值将重置为默认值。

2. 运行 `helm list` 验证图表和应用程序版本均已升级。运行 `tridentctl logs` 查看任何调试消息。

从升级 `tridentctl` 安装到TRIdent操作员

您可以从升级到最新版本的Trident操作员 `tridentctl` 安装。现有后端和PVC将自动可用。

备注	在切换安装方法之前，请查看 "在安装方法之间移动"。

步骤

1. 下载最新版本的Trident。

```
# Download the release required [26.10.1]
mkdir 26.10.1
cd 26.10.1
wget
https://github.com/NetApp/trident/releases/download/v26.10.1/trident-
installer-26.10.1.tar.gz
tar -xf trident-installer-26.10.1.tar.gz
cd trident-installer
```

2. 从清单中创建 `tridentorchator` CRD 。

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. 将集群范围的运算符部署在同一命名空间中。

```
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. 确认已将三项功能升级到预期版本。

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v26.10.0
```

使用 **tridentctl** 升级 **Trident**

您可以使用轻松升级现有的Trident安装 `tridentctl`。

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部署所使用的永久性卷请求(PVC)和永久性卷(PV)。已配置的PV在Trident脱机时仍可用、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. 在中运行卸载命令 `tridentctl`以删除与Trident关联的所有资源、CRD和相关对象除外。

```
./tridentctl uninstall -n <namespace>
```

2. 重新安装 Trident。请参阅 "使用trdentcl安装Trident"。

重要说明	请勿中断升级过程。确保安装程序运行完毕。

使用trdentctrd管理Trident

`https://github.com/NetApp/trident/releases["Trident 安装程序包"^]`包含`tridentctl`命令行实用程序、可用于轻松访问`Trident`。具有足够`Privileges`的`Kubornetes`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` 以获取可用命令的列表 `tridentctl` 或附加 `--help` 用于任何命令的标志、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

将资源添加到Trident。

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所需容器映像的表。

import

将现有资源导入到Trident。

install

安装 Trident 。

logs

从Trident打印日志。

send

从Trident发送资源。

卸载

卸载Trident。

update

在Trident中修改资源。

update backend state

暂时暂停后端操作。

upgrade

在Trident中升级资源。

ve版本

打印Trident版本。

▼ 全局标志

-d，--debug

调试输出。

-h，--help

帮助 `tridentctl`。

-k，--kubeconfig string

指定 `KUBECONFIG` 在本地或从一个Kubornetes集群到另一个集群运行命令的路径。

备注	或者、您也可以导出 `KUBECONFIG` 变量、用于指向特定的Kubbernetes集群和问题描述 `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。

警告	可以将 Trident REST 接口配置为仅以 127.0.0.1 （对于 IPv4 ）或（：： 1 ）（对于 IPv6 ）侦听和提供服务。

命令选项和标志

创建

使用 `create`命令将资源添加到Trident。

```
tridentctl create [option]
```

选项

backend：将后端添加到Trident。

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。

标志

-h，--help：卷帮助。
--parentOfSubordinate string：将查询限制为从源卷。
--subordinateOf string：将查询限制为卷的下属。

映像

使用 `images`标志打印Trident所需容器映像的表。

```
tridentctl images [flags]
```

标志

-h，--help：图像帮助。
-v，--k8s-version string：Kubornetes集群的语义版本。

导入卷

使用 `import volume`命令将现有卷导入Trident。

```
tridentctl import volume <backendName> <volumeName> [flags]
```

别名

volume， v

标志

-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。不要假定卷生命周期管理。

安装

使用 `install`标志安装Trident。

```
tridentctl install [flags]
```

标志

--autosupport-image string：自动支持遥测的容器映像（默认“netapp/trident autosupport:<current-version>”）。
--autosupport-proxy string：用于发送自动支持遥测的代理的地址/端口。
--enable-node-prep：尝试在节点上安装所需的软件包。
--generate-custom-yaml：无需安装任何东西即可生成 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 1 分 30 秒）。
--image-registry string：内部图像注册表的地址/端口。
--k8s-timeout duration：所有 Kubernetes 操作的超时时间（默认 3m0s）。
--kubelet-dir string：kubelet 内部状态的主机位置（默认为“/var/lib/kubelet”）。
--log-format string： Trident日志格式（文本、json）（默认“文本”）。
--node-prep：使Trident能够准备 Kubernetes 集群的节点，以使用指定的数据存储协议管理卷。现在，**`iscsi`是唯一支持的值。从 `OpenShift 4.19` 开始，此功能支持的最低`Trident`版本为 `25.06.1`。**
`--pv string`： Trident使用的旧版 PV 的名称，确保其不存在（默认为“trident”）。
--pvc string： Trident使用的旧式 PVC 的名称，确保其不存在（默认为“trident”）。
--silence-autosupport：不要自动将自动支持包发送到NetApp （默认为 true）。
--silent：安装期间禁用大部分输出。
--trident-image string：要安装的Trident图像。
--k8s-api-qps：Kubernetes API 请求的每秒查询数 (QPS) 限制（默认 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 的通信。

日志

使用 `logs`标志从Trident打印日志。

```
tridentctl logs [flags]
```

标志

-a --archive：创建包含所有日志的支持归档文件，除非另有说明。
-h --help：日志帮助。
-l --log string：要显示的Trident日志。Trident |自动|Auto-operator|all之一Trident (默认值为"auto")。
--node string：要从中收集节点Pod日志的Kubbernetes节点名称。
-p --previous：获取上一个容器实例(如果存在)的日志。
--sidecars：获取此容器的日志。

发送

使用 `send`命令从Trident发送资源。

```
tridentctl send [option]
```

选项

`autosupport`：将AutoSupport 归档发送给NetApp。

卸载

使用 `uninstall`标志卸载Trident。

```
tridentctl uninstall [flags]
```

标志

`-h, --help`：卸载帮助。
`--silent`：卸载期间禁用大多数输出。

更新

使用 `update`命令修改Trident中的资源。

```
tridentctl update [option]
```

选项

`backend`：在Trident中更新后端。

更新后端状态

使用 `update backend state` 用于暂停或恢复后端操作的命令。

```
tridentctl update backend state <backend-name> [flag]
```

需要考虑的要点

- 如果使用TridentBackendConfig (tbc)创建后端、则无法使用文件更新后端 `backend.json` 。
- 如果 `userState` 已在tbc中设置、则无法使用命令修改 `tridentctl update backend state <backend-name> --user-state suspended/normal` 。
- 要在通过tbc设置后重新能够通过tldentcdt设置 `userState`、必须从tbc中删除此字段。 `userState`可以使用命令来完成此操作 `kubectl edit tbc` 。删除此字段后 `userState`、您可以使用 `tridentctl update backend state`命令更改 `userState`后端的。
- 使用 `tridentctl update backend state` 更改 `userState`。您还可以使用或文件更新 `userState` `TridentBackendConfig` `backend.json` ；这会触发后端的完全重新初始化、并且可能会非常耗时。

  标志

  `-h, --help`：后端状态帮助。
  `--user-state`：设置为 `suspended` 暂停后端操作。设置为 `normal` 恢复后端操作。设置为时 `suspended`：

- AddVolume 和 Import Volume 已暂停。
- CloneVolume、 ResizeVolume PublishVolume、、 UnPublishVolume、 CreateSnapshot GetSnapshot RestoreSnapshot、 DeleteSnapshot、、 RemoveVolume、 GetVolumeExternal ReconcileNodeAccess 保持可用。

您也可以使用后端配置文件或中的字段更新后端状态 userState TridentBackendConfig backend.json。有关详细信息，请参阅 "用于管理后端的选项" 和 "使用 kubectl 执行后端管理"。

- 示例： *

JSON

按照以下步骤使用文件更新 userState backend.json：

1. 编辑 backend.json 文件以包含 userState 字段、并将其值设置为"已附加"。
2. 使用 `tridentctl update backend`命令和更新的路径 `backend.json`文件。

   例子： tridentctl update backend -f /<path to backend JSON file>/backend.json -n trident

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "<redacted>",
  "svm": "nas-svm",
  "backendName": "customBackend",
  "username": "<redacted>",
  "password": "<redacted>",
  "userState": "suspended"
}
```

YAML

您可以在使用命令应用tbc后对其进行编辑 kubectl edit <tbc-name> -n <namespace> 。以下示例使用选项将后端状态更新为暂停 userState: suspended：

```
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` 用于打印版本的标志 `tridentctl` 以及正在运行的Trident服务。

```
tridentctl version [flags]
```

标志

`--client`：仅限客户端版本(不需要服务器)。
`-h, --help`：版本帮助。

插件支持

tridentctd支持类似于kubectd的插件。如果插件二进制文件名遵循"tridentcts-tld"方案<plugin>、并且二进制文件位于列出了路径环境变量的文件夹中、则tridentctL)将检测插件。所有检测到的插件都会在trdentctrd帮助的插件部分中列出。您也可以通过在环境变量TRIDENTCTL_plugin_path中指定plugin文件夹来限制搜索(示例：`TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`)。如果使用了变量、则trdencdld仅在指定文件夹中搜索。

监控 Trident 存储操作

Trident提供了一组Prometheus指标端点、可用于监控Trident性能。

概述

通过Trident提供的指标、您可以执行以下操作：

- 密切关注Trident的运行状况和配置。您可以检查操作的成功程度以及它是否能够按预期与后端进行通信。
- 检查后端使用情况信息，并了解在后端配置的卷数量以及占用的空间量等。
- 维护可用后端配置的卷数量的映射关系。
- 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。

备注	默认情况下，Trident 的指标会暴露在目标端口上。 `8001`在 `/metrics`端点。安装Trident时，这些指标*默认启用*。您可以配置通过 HTTPS 在端口上消费Trident指标。 `8444`也一样。

您需要的内容

- 安装了Trident的Kubnetes集群。
- 一个 Prometheus 实例。可以是 "容器化 Prometheus 部署" 或者，您也可以选择将 Prometheus 作为运行 "原生应用程序"。

第 1 步：定义 Prometheus 目标

您应该定义一个 Prometheus 目标来收集指标并获取有关Trident管理的后端、它创建的卷等的信息。看"Prometheus Operator 文档"。

第 2 步：创建 Prometheus ServiceMonitor

要使用 Trident 指标，您应创建一个 Prometheus ServiceMonitor ，该监控器可监控 `trident CSI` 服务并侦听 `mtrigics` 端口。示例 ServiceMonitor 如下所示：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
    release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
      - trident
  endpoints:
    - port: metrics
      interval: 15s
```

此 ServiceMonitor 定义检索由以下方式返回的指标： `trident-csi`服务，特别是寻找 `metrics`服务的端点。因此，Prometheus 现在已配置为能够理解 Trident 的指标。

除了直接从Trident获得的指标之外、kubelet还会通过自己的指标端点公开许多 `kubelet_volume_*`指标。Kubelet 可以提供有关已连接的卷， Pod 及其处理的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，您必须修改 ServiceMonitor 定义以包含 TLS 配置。你还需要复制 `trident-csi`秘密来自 `trident`将命名空间指向 Prometheus 运行所在的命名空间。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:
    - 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支持所有安装方法中的 HTTPS 指标：tridentctl、Helm chart 和 Operator：

- 如果您正在使用 `tridentctl install`命令，你可以传递 `--https-metrics`启用 HTTPS 指标的标志。
- 如果您使用的是 Helm Chart，则可以进行设置。 `httpsMetrics`用于启用 HTTPS 指标的参数。
- 如果您使用的是 YAML 文件，则可以添加以下内容： `--https_metrics`向…… `trident-main`容器中的 `trident-deployment.yaml`文件。

第 3 步：使用 PromQL 查询 Trident 指标

PromQL 非常适合创建返回时间序列或表格数据的表达式。

您可以使用以下 PromQL 查询：

获取 **Trident** 运行状况信息

- 来自**Trident**的**HTTP 2XX**响应的百分比

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- 通过状态代码来自**Trident**的**REST**响应的百分比

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- 由**Trident**执行的操作的平均持续时间**(**以毫秒为单位**)**

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

获取**Trident**使用情况信息

- 卷大小 * 平均值 *

```
trident_volume_allocated_bytes/trident_volume_count
```

- * 每个后端配置的卷总空间 *

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

获取单个卷的使用情况

备注	只有在同时收集 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会每天向NetApp发送Prometheus指标和基本后端信息。

- 要阻止Trident向NetApp发送Prometheus指标和基本后端信息、请在Trident安装期间传递此 `--silence-autosupport`标志。
- Trident还可以通过按需向NetApp支持发送容器日志 `tridentctl send autosupport`。您需要触发Trident来上传其日志。在提交日志之前，应接受NetApp的https://www.netapp.com/company/legal/privacy-policy/["隐私政策"^]。
- 除非指定、否则Trident将提取过去24小时的日志。
- 您可以使用标志指定日志保留时间范围 `--since`。例如： `tridentctl send autosupport --since=1h`。此信息通过随Trident一起安装的容器收集和发送 `trident-autosupport`。您可以从获取容器映像 "Trident AutoSupport"。
- Trident AutoSupport 不会收集或传输个人身份信息 (PII) 或个人信息。它附带 "EULA"，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发送的有效负载示例如下所示：

```
---
items:
  - backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
    protocol: file
    config:
      version: 1
      storageDriverName: ontap-nas
      debug: false
      debugTraceFlags: null
      disableDelete: false
      serialNumbers:
        - nwkvzfanek_SN
      limitVolumeSize: ""
    state: online
    online: true
```

- AutoSupport 消息将发送到 NetApp 的 AutoSupport 端点。如果使用私有注册表存储容器映像，则可以使用 ` -image-regRegistry` 标志。
- 您也可以通过生成安装 YAML 文件来配置代理 URL 。为此，可以使用 `tridentctl install -generate-custom-yaml` 创建 YAML 文件，并在 `trident deployment.yaml` 中为 `trident autosupport` 容器添加 ` -proxy-url` 参数。

禁用Trident指标

要 ` 报告指标，您应生成自定义 YAML （使用` -generate-custom-yaml ` 标志）并对其进行编辑，以删除为 `trident 主` 容器调用的` -metrics 标志。

卸载 Trident

您应使用与安装Trident相同的方法卸载Trident。

关于此任务

- 如果您需要修复在升级、依赖关系问题或升级失败或不完整后发现的错误，则应卸载Trident，然后按照适用于该版本的特定说明重新安装早期版本"version"。这是将_降 级_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，请参见"完全删除Trident和CRD"。

开始之前

如果要停用Kubnetes集群、则必须在卸载之前删除使用Trident创建的卷的所有应用程序。这样可以确保在删除之前、在Kubbernetes节点上未取消对这些PVC的审核。

确定原始安装方法

您应使用与安装Trident相同的方法来卸载它。卸载之前、请验证最初安装Trident时使用的版本。

1. 使用 ... `kubectl get pods -n trident` 检查Pod。
   - 如果没有操作员POD，则使用安装了Trident `tridentctl`。
   - 如果有操作员POD、则Trident是使用Trident操作员手动或使用Helm安装的。
2. 如果有操作员POD、请使用 `kubectl describe tproc trident`确定Trident是否是使用Helm安装的。
   - 如果有Helm标签、则Trident是使用Helm安装的。
   - 如果没有Helm标签、则Trident是使用Trident操作员手动安装的。

卸载TRIdent操作员安装

您可以手动卸载或使用Helm卸载TRYDent操作员安装。

卸载手动安装

如果您使用操作员安装了Trident、则可以通过执行以下操作之一将其卸载：

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`、Trident操作符将卸载Trident、但不会删除Trident Orchestrator本身。如果要重新安装 Trident ，应清理 Trident Orchestrator 并创建新的 Trident 。

2. 删除 **`TridentOrchestrator`**:删除用于部署Trident的CR后 `TridentOrchestrator`，您将指示操作员卸载Trident。操作员将处理Trident部署和守护进程的删除过程、并继续删除此部署和守护进程 `TridentOrchestrator`、同时删除其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，则可以使用将其卸载 `helm uninstall`。

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

卸载 **`tridentctl`** 安装

使用 `uninstall`中的命令 `tridentctl`删除与Trident关联的所有资源(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

您必须先在主机上安装和配置必要的协议前提条件、然后才能部署Trident。

验证要求

- 验证您的部署是否满足所有要求 "要求"。
- 验证您是否安装了受支持的 Docker 版本。如果您的 Docker 版本已过时， "安装或更新它"。

```
docker --version
```

- 验证是否已在主机上安装和配置协议前提条件。

NFS工具

使用适用于您的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
```

警告 安装NFS工具后重新启动工作节点、以防止在将卷连接到容器时失败。

iSCSI工具

使用适用于您的操作系统的命令安装iSCSI工具。

RHEL 8+

1. 安装以下系统软件包：

```
sudo yum install -y lsscsi iscsi-initiator-utils sg3_utils device-
mapper-multipath
```

2. 检查 iscsi-initiator-utils 版本是否为 6.2.0.877-2.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
```

> 备注 确保 detc/multipath.conf contains find_multipaths no under` efaults `.

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 2.0.877-5ubuntu2.10 或更高版本（对于双子系统）或 2.0.877-7.1ubuntu6.1 或更高版本（对于 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
```

备注 确保 detc/multipath.conf contains find_multipaths no under` efaults `.

5. 确保已启用并运行 open-iscsi 和 multipath-tools：

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

NVMe工具

使用适用于您的操作系统的命令安装NVMe工具。

备注

- NVMe需要RHEL 9或更高版本。
- 如果Kubelnetes节点的内核版本太旧、或者NVMe软件包不适用于您的内核版本、您可能需要将节点的内核版本更新为具有NVMe软件包的版本。

RHEL 9

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

Ubuntu

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

FC工具

使用适用于您的操作系统的命令安装FC工具。

- 如果将运行RHL/Red Hat Enterprise Linux Core-OS (RHOS)的工作节点与FC PVs结合使用、请在StorageClass中指定 `discard`mountOption以执行实时空间回收。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
```

> 备注 确保 `detc/multipath.conf` contains `find_multipaths no` under` efaults `.

3. 确保 `multipathd`正在运行：

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

Ubuntu

1. 安装以下系统软件包：

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

2. 启用多路径：

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

> 备注 确保 `detc/multipath.conf` contains `find_multipaths no` under` efaults `.

3. 确保 `multipath-tools`已启用且正在运行：

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

部署Trident

适用于Docker的Trident可与适用于NetApp存储平台的Docker生态系统直接集成。它支持从存储平台到 Docker 主机的存储资源配置和管理，并提供一个框架，用于在未来添加其他平台。

多个Trident实例可以同时在同一主机上运行。这样可以同时连接到多个存储系统和存储类型，并能够自定义用于 Docker 卷的存储。

您需要的内容

请参见"部署的前提条件"。确保满足这些前提条件后、即可部署Trident。

Docker 托管插件方法（ 1.13/17.03 及更高版本）

备注	开始之前 如果在传统守护进程方法中使用的是Trident Docker 1.3/17.03之前的版本、请确保先停止Trident进程并重新启动Docker守护进程、然后再使用托管插件方法。

1. 停止所有正在运行的实例：

```
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 引擎 17.03 （新版本 1.13 ）或更高版本。

```
docker --version
```

如果您的版本已过期， "安装或更新安装"。

步骤

1. 创建配置文件并按如下所示指定选项：
   - `config` ： 默认文件名为 `config.json` ，但您可以使用所选的任何名称，方法是使用文件名指定 `config` 选项。此配置文件必须位于主机系统上的 ` /etc/netappdvp` 目录中。
   - `log-level` ： 指定日志记录级别（ `debug` ， `info` ， `warn` ， `error` ， `fal` ）。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. 为配置文件创建一个位置：

```
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. 使用受管插件系统启动Trident。请替换 `<version>`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使用已配置系统中的存储。

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** 或更早版本）

开始之前

1. 确保您已安装 Docker 版本 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 "按照适用于您的分发版本的说明进行操作"。

2. 确保已为您的系统配置 NFS 和 / 或 iSCSI 。

步骤

1. 安装和配置 NetApp Docker 卷插件：

a. 下载并解压缩应用程序：

```
wget
https://github.com/NetApp/trident/releases/download/v26.10.1/trident-
installer-26.10.1.tar.gz
tar -xf trident-installer-26.10.1.tar.gz
cd trident-installer
```

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. 为配置文件创建一个位置：

```
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
```

> 备注　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
```

备注 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

您可以安全地升级适用于Docker的Trident、而不会对正在使用的卷产生任何影响。在升级过程中、有一段短暂的时间、 `docker volume`指向插件的命令将不会成功、应用程序将无法挂载卷、直到插件重新运行为止。在大多数情况下，这只需要几秒钟。

升级

执行以下步骤升级适用于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
```

备注	Trident 18.01版取代了nDVP。您应直接从映像升级 `netapp/ndvp-plugin`到 `netapp/trident-plugin`映像。

4. 启用插件：

```
docker plugin enable netapp:latest
```

5. 验证是否已启用此插件：

```
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
```

重要说明	如果要从旧版本Trident (20.10之前的版本)升级到Trident 20.10或更高版本、则可能会遇到错误。有关详细信息，请参阅 "已知问题"。如果遇到此错误、则应先禁用此插件、然后删除此

插件、然后再通过传递额外的配置参数来安装所需的Trident版本： `docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant-all-permissions config=config.json`

卸载

执行以下步骤卸载适用于Docker的Trident。

步骤

1. 删除插件创建的所有卷。
2. 禁用插件：

```
docker plugin disable netapp:latest
docker plugin ls
ID                  NAME                DESCRIPTION
ENABLED
7067f39a5df5        netapp:latest       nDVP - NetApp Docker Volume
Plugin   false
```

3. 删除插件：

```
docker plugin rm netapp:latest
```

使用卷

您可以使用标准命令以及根据需要指定的Trident驱动程序名称轻松创建、克隆和删除卷 `docker volume`。

创建卷

- 使用默认名称创建包含驱动程序的卷：

```
docker volume create -d netapp --name firstVolume
```

- 创建具有特定Trident实例的卷：

```
docker volume create -d ntap_bronze --name bronzeVolume
```

> 备注　如果未指定任何 "选项"，将使用驱动程序的默认值。

- 覆盖默认卷大小。请参阅以下示例，使用驱动程序创建 20 GiB 的卷：

```
docker volume create -d netapp --name my_vol --opt size=20G
```

提示	卷大小以字符串表示，该字符串包含一个包含可选单元的整数值（例如： 10 G， 20 GB，3 TiB）。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）。速率单位使用 2 的电流（ G = GiB， T = TiB，…）。

删除卷

- 像删除任何其他 Docker 卷一样删除此卷：

```
docker volume rm firstVolume
```

重要说明	使用 `solidfire-san` 驱动程序时，上述示例将删除并清除卷。

执行以下步骤升级适用于Docker的Trident。

克隆卷

使用时 `ontap-nas`， `ontap-san`， 和 `solidfire-san`存储驱动程序， Trident可以克隆卷。使用时 `ontap-nas-flexgroup`或者 `ontap-nas-economy`驱动程序不支持克隆。从现有卷创建新卷将创建一个新的快照。

- 检查卷以枚举快照：

```
docker volume inspect <volume_name>
```

- 从现有卷创建新卷。这将导致创建新快照：

```
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。此操作不会创建新快照：

```
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

如果容器没有分区、并且Trident支持其文件系统、则您可以使用Trident *仅*通过容器访问外部创建的块设备(或其克隆)(例如、 `ext4`无法通过`Trident`访问格式化的 `/dev/sdc1`)。

驱动程序专用的卷选项

每个存储驱动程序都有一组不同的选项，您可以在创建卷时指定这些选项来自定义结果。有关适用于您配置的存储系统的选项，请参见以下内容。

在卷创建操作期间使用这些选项非常简单。在命令行界面操作期间，使用 ` -o` 运算符提供选项和值。这些参数将覆盖 JSON 配置文件中的任何等效值。

ONTAP 卷选项

NFS、iSCSI和FC的卷创建选项包括以下内容：

选项	Description
s大小	卷的大小默认为 1 GiB 。
s页面预留	精简或厚配置卷，默认为精简。有效值为 `none` （精简配置）和 `volume` （厚配置）。
`sSnapshot` 策略	此操作会将 Snapshot 策略设置为所需的值。默认值为 `none`，表示不会自动为卷创建快照。除非存储管理员进行修改、否则所有ONTAP系统上都会存在一个名为"default"的策略、此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目录来恢复快照中保留的数据 `.snapshot`。
`sSnapshot` 预留	此操作会将快照预留设置为所需百分比。默认值为 no 值，这意味着如果您选择了 snapshotPolicy ， ONTAP 将选择 snapshotReserve （通常为 5% ）；如果 snapshotPolicy 为 none ，则选择 0% 。您可以在配置文件中为所有 ONTAP 后端设置默认 snapshotReserve 值，并可将其用作除 ontap-nas-economy. 以外的所有 ONTAP 后端的卷创建选项。
`splitOnClone`	克隆卷时，此操作将使发生原因 ONTAP 立即从其父卷拆分克隆。默认值为 `false`。在克隆卷的某些使用情形中，最好在创建后立即将克隆从其父卷中拆分，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、但只能节省很少的存储空间、因此最好立即拆分克隆。
加密	在新卷上启用NetApp卷加密(NVE)；默认为`false`。要使用此选项，必须在集群上获得 NVE 的许可并启用 NVE 。 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，请参阅："Trident如何与NVE和NAE配合使用"。
分层策略	设置要用于卷的分层策略。这将决定数据在变为非活动状态（冷）时是否移至云层。

以下附加选项适用于 NFS * 仅 * ：

选项	Description
`unixPermissions`	此选项用于控制为卷本身设置的权限。默认情况下，权限将设置为 ` -rwxr-xr-x ，或以数字表示法 0755 表示，并且 `root` 将为所有者。文本或数字格式均可使用。

选项	Description
snapshotDir	将其设置为 true 将创建 .snapshot 访问卷的客户端可以看到的目录。默认值为 false、表示的可见性 .snapshot 默认情况下、目录处于禁用状态。某些映像(例如官方MySQL映像)在出现时无法按预期运行 .snapshot 目录可见。
exportPolicy	设置要用于卷的导出策略。默认值为 default 。
securityStyle	设置用于访问卷的安全模式。默认值为 UNIX 。有效值为 UNIX 和 mi 已加传真 。

以下附加选项适用于 iSCSI * 仅 * ：

选项	Description
fileSystemType	设置用于格式化 iSCSI 卷的文件系统。默认值为 ext4 。有效值为 ext3 ， ext4 和 xfs 。
spaceAllocation	将其设置为 false 将关闭LUN的空间分配功能。默认值为 true、表示当卷空间用尽且卷中的LUN无法接受写入时、ONTAP 会向主机发出通知。此选项还允许 ONTAP 在主机删除数据时自动回收空间。

示例

请参见以下示例：

- 创建一个 10 GiB 卷：

```
docker volume create -d netapp --name demo -o size=10G -o
encryption=true
```

- 创建一个带有快照的 100 GiB 卷：

```
docker volume create -d netapp --name demo -o size=100G -o
snapshotPolicy=default -o snapshotReserve=10
```

- 创建启用了 setuid 位的卷：

```
docker volume create -d netapp --name demo -o unixPermissions=4755
```

最小卷大小为 20 MiB。

如果未指定快照预留且快照策略为 none，则Trident将使用0%的快照预留。

- 创建无快照策略且无快照预留的卷：

```
docker volume create -d netapp --name my_vol --opt snapshotPolicy=none
```

- 创建一个无快照策略且自定义快照预留为 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Snapshot策略创建卷、并接受ONTAP的默认Snapshot预留(通常为5%)：

```
docker volume create -d netapp --name my_vol --opt
snapshotPolicy=myPolicy
```

Element 软件卷选项

Element 软件选项会显示与卷关联的大小和服务质量（ QoS ）策略。创建卷时，将使用 ` -o type=service_level` 命名空间指定与其关联的 QoS 策略。

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	Description
s大小	卷的大小，默认为 1 GiB 或配置条目…“defaults” ：{“size”：“5G”}。
块大小	使用 512 或 4096 ，默认为 512 或配置条目 DefaultBlockSize 。

示例

请参见以下包含 QoS 定义的示例配置文件：

```
{
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
```

```
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}
```

在上述配置中，我们有三个策略定义：铜牌，银牌和金牌。这些名称是任意的。

- 创建 10 GiB 黄金卷：

```
docker volume create -d solidfire --name sfGold -o type=Gold -o size=10G
```

- 创建 100 GiB 青铜卷：

```
docker volume create -d solidfire --name sfBronze -o type=Bronze -o
size=100G
```

收集日志

您可以收集日志以帮助进行故障排除。收集日志的方法因运行 Docker 插件的方式而异。

收集日志以进行故障排除

步骤

1. 如果您正在使用建议的托管插件方法(即使用命令)运行Trident、请按如下所示查看这些插件 `docker`

plugin：

```
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，请安装启用了调试日志记录的插件：

```
docker plugin install netapp/trident-plugin:<version> --alias <alias>
debug=true
```

或者，在已安装插件的情况下启用调试日志记录：

```
docker plugin disable <plugin>
```

```
docker plugin set <plugin> debug=true
```

```
docker plugin enable <plugin>
```

3. 如果您在主机上运行二进制文件本身、则主机的中会显示日志 `/var/log/netappdvp` 目录。要启用调试日志记录、请指定 `-debug` 运行插件时。

一般故障排除提示

- 新用户遇到的最常见问题是配置不当，导致插件无法初始化。如果发生这种情况，在尝试安装或启用插件时，您可能会看到如下消息：

  守护进程的错误响应：拨打 `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，该插件已构建了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 挂载到容器时出现问题，请确保 `rpcbind` 已安装并正在运行。对主机操作系统使用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

项来检查 rpcbind 服务的状态。

管理多个Trident实例

如果希望同时提供多个存储配置，则需要多个 Trident 实例。多个实例的关键在于，在主机上实例化 Trident 时，使用容器化插件的 ` -alias` 选项或 ` -volume-driver` 选项为其提供不同的名称。

Docker 托管插件（ 1.13/17.03 或更高版本）的步骤

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，指定其他别名和配置文件。

```
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，对于银牌卷：

```
docker volume create -d silver --name ntapSilver
```

传统（ 1.12 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 启动具有 iSCSI 配置的插件：

```
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

请参见适用于您的Trident配置的配置选项。

全局配置选项

无论使用哪个存储平台、这些配置选项都适用于所有Trident配置。

选项	Description	示例
ve版本	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
s存储前缀	卷名称的可选前缀。默认值：netappdvp_。	staging_
limitVolumeSize	卷大小的可选限制。默认值：""(未强制实施)	10g

提示	不要对元素后端使用 storagePrefix(包括默认值)。默认情况下， `solidfire-san`驱动程序将忽略此设置，而不使用前缀。NetApp建议对Docker卷映射使用特定的租户ID、或者在可能已使用任何名称创建的情况下使用由Docker版本、驱动程序信息和原始名称填充的属性数据。

您可以使用默认选项来避免在创建的每个卷上指定这些选项。s 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，请参见 ONTAP 配置一节。

选项	Description	示例
s大小	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，在使用 ONTAP 时，还可以使用以下顶级选项。

选项	Description	示例
m年 月 日	ONTAP 管理 LIF 的 IP 地址。您可以指定完全限定域名（ 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。如果未提供此参数、则Trident将从SVM提取数据LIF。您可以指定要用于NFS挂载操作的完全限定域名(FQDN)、以便创建轮叫DNS来在多个dataLIF之间进行负载平衡。 **SAN ONTAP**驱动程序:不指定用于iSCSI或FC。Trident使用"ONTAP选择性LUN映射"发现建立多路径会话所需的iSCSI或FC LUN。如果明确定义、则会生成警告 dataLIF。	10.0.0.2
sVM	要使用的 Storage Virtual Machine （如果管理 LIF 为集群 LIF ，则为必填项）	svm_nfs
用户名	用于连接到存储设备的用户名	vsadmin
密码	用于连接到存储设备的密码	secret
聚合	要配置的聚合（可选；如果设置了聚合，则必须将其分配给 SVM ）。对于 `ontap-nas-flexgroup`驱动程序、此选项将被忽略。分配给SVM的所有聚合均用于配置FlexGroup卷。	aggr1
limitAggregateUsage	可选，如果使用量超过此百分比，则配置失败	75%
nfsMountOptions	对NFS挂载选项进行精细控制；默认为"-o nfsvers=3"。仅适用于 **`ontap-nas`**和 **`ontap-nas-economy`**驱动程序。 "请参见此处的 NFS 主机配置信息"(英文)。	-o nfsvers=4
igroupName	Trident会将每个节点创建和管理 igroups`为 `netappdvp。 此值不能更改或省略。 仅适用于 **ontap-san** 驱动程序。	netappdvp

选项	Description	示例
limitVolumeSize	可要求的最大卷大小。	300g
qtreesPerFlexvol	每个 FlexVol 的最大 qtree 数必须在 50 ， 300 范围内，默认值为 200 。 *用于 ontap-nas-economy 驱动程序、此选项允许自定义每个qtree*的最大FlexVol 数。	300
sanType	*仅支持 ontap-san`驱动程序。*用于为iSCSI、`nvme`NVMe/TCP或基于光纤通道的 `fcp`SCSI (FC)选择`iscsi。	iscsi 如果为空
limitVolumePoolSize	* ontap-san-economy `ontap-san-economy`仅支持和驱动程序。*限制ONTAP ONTAP经济型和ONTAP经济型驱动程序中的FlexVol大小。	300g

您可以使用默认选项来避免在创建的每个卷上指定这些选项：

选项	Description	示例
s页面预留	空间预留模式； none (精简配置)或 volume (厚)	无
sSnapshot 策略	要使用的Snapshot策略、默认为 none	无
sSnapshot 预留	Snapshot预留百分比、默认值为""以接受ONTAP默认值	10
splitOnClone	创建克隆时将其从父级拆分、默认为 false	false
加密	在新卷上启用NetApp卷加密(NVE)；默认为`false`。要使用此选项，必须在集群上获得 NVE 的许可并启用 NVE 。 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，请参阅："Trident如何与NVE和NAE配合使用"。	true
unixPermissions	对于已配置的NFS卷、NAS选项默认为 777	777
snapshotDir	用于访问目录的NAS选项 .snapshot。	启用或禁用对 .snapshot 目录的访问。 有效值： true， false。默认行为可能因 ONTAP 和 Trident 版本而异。明确设置以确保行为一致。

选项	Description	示例
exportPolicy	要使用的NFS导出策略的NAS选项、默认为 default	default
securityStyle	用于访问已配置NFS卷的NAS选项。 NFS支持 mixed 和 unix 安全模式。默认值为 unix。	unix
fileSystemType	SAN选项要选择文件系统类型、默认为 ext4	xfs
分层策略	要使用的分层策略，默认为 none。	无
skipRecoveryQueue	删除卷时，绕过存储中的恢复队列，立即删除卷。	``

扩展选项

`ontap-nas`和 `ontap-san`驱动程序会为每个Docker卷创建一个ONTAP FlexVol。ONTAP支持每个集群节点多达1000个FlexVol、集群最多12、000个FlexVol卷如果您的Docker卷要求符合此限制、则该驱动程序是首选NAS解决方案、因为它 `ontap-nas`具有FlexVol提供的其他功能、例如Docker卷粒度快照和克隆。

如果所需的 Docker 卷数超出 FlexVol 限制所能容纳的范围，请选择 ontap-nas-economy" 或 ontap-san-economy" 驱动程序。

此 `ontap-nas-economy`驱动程序会在自动管理的FlexVol卷池中将Docker卷创建为ONTAP qtrees。qtree 的扩展能力远高于此，每个集群节点最多可扩展 100 ， 000 个，每个集群最多可扩展 2 ， 400 ， 000 个，但某些功能会受到影响。该 `ontap-nas-economy`驱动程序不支持Docker卷粒度快照或克隆。

备注	Docker Swarm目前不支持此 `ontap-nas-economy`驱动程序、因为Docker Swarm不会在多个节点之间编排卷创建过程。

此 `ontap-san-economy`驱动程序会在自动管理的FlexVol卷共享池中将Docker卷创建为ONTAP LUN。这样，每个 FlexVol 就不会仅限于一个 LUN ，并且可以为 SAN 工作负载提供更好的可扩展性。根据存储阵列的不同，ONTAP 每个集群最多支持 16384 个 LUN 。由于卷是下面的 LUN ，因此此驱动程序支持 Docker 卷粒度快照和克隆。

选择 `ontap-nas-flexgroup`一个驱动程序来提高单个卷的并行处理能力、该卷可能会增长到包含数十亿个文件的PB级范围。FlexGroup 的一些理想用例包括 AI/ML/DL ，大数据和分析，软件构建，流式传输，文件存储库等。配置FlexGroup卷时、Trident会使用分配给SVM的所有聚合。Trident 中的 FlexGroup 支持还需要注意以下事项：

- 需要 ONTAP 9.2 或更高版本。
- 截至本文撰写时， FlexGroup 仅支持 NFS v3 。
- 建议为 SVM 启用 64 位 NFSv3 标识符。
- 建议的最小FlexGroup成员/卷大小为 100 GiB。
- FlexGroup卷不支持克隆。

有关 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，您可以运行多个Docker卷插件实例，其中一个使用，另一个 `ontap-nas-economy`使用 `ontap-nas`。

Trident的自定义ONTAP角色

您可以创建Privileges最低的ONTAP集群角色、这样就不必使用ONTAP管理员角色在Trident中执行操作。如果在Trident后端配置中包含用户名、则Trident将使用您创建的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命令行界面

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系统管理器中执行以下步骤：

1. 创建自定义角色：
   a. 要在集群级别创建自定义角色，请选择*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*用户和角色*旁边的箭头图标(→)。
   c. 在*角色*下选择*+添加*。
   d. 定义角色的规则，然后单击*Save*。
2. 将角色映射到**Trident user**：+在*Users and Roles*页面上执行以下步骤：
   a. 在*用户*下选择添加图标*+*。

b. 选择所需的用户名，然后在下拉菜单中为*rouser*选择一个角色。

c. 单击 * 保存 * 。

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

▼ **<code>ontap-nas</code>** 驱动程序的**NFS**示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "defaults": {
    "size": "10G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

▼ **<code>ontap-nas-flexgroup</code>** 驱动程序的**NFS**示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-flexgroup",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "defaults": {
    "size": "100G",
    "spaceReserve": "none",
    "exportPolicy": "default"
  }
}
```

▼ **<code>ontap-nas-economy</code> 驱动程序的NFS示例**

```
{
  "version": 1,
  "storageDriverName": "ontap-nas-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1"
}
```

▼ **<code>ontap-san</code> 驱动程序的iSCSI示例**

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

▼ **<code>ontap-san-economy</code> 驱动程序的NFS示例**

```
{
  "version": 1,
  "storageDriverName": "ontap-san-economy",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.3",
  "svm": "svm_iscsi_eco",
  "username": "vsadmin",
  "password": "password",
  "aggregate": "aggr1",
  "igroupName": "netappdvp"
}
```

▼ **NVMe/TCP <code>ontap-san</code> 驱动程序示例**

```
{
  "version": 1,
```

```
  "backendName": "NVMeBackend",
  "storageDriverName": "ontap-san",
  "managementLIF": "10.0.0.1",
  "svm": "svm_nvme",
  "username": "vsadmin",
  "password": "password",
  "sanType": "nvme",
  "useREST": true
}
```

▼ **<code> ONTAP－</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，在使用 Element 软件（ NetApp HCI/SolidFire ）时，还可以使用这些选项。

选项	Description	示例
端点	<a href="https://<login>:<password>@<mvip>/json-rpc/<element-version>" class="bare">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
租户名称	要使用的 SolidFireF 租户（如果未找到，则创建）	`docker`
`InitiatorIFace`	将 iSCSI 流量限制为非默认接口时，请指定接口	`default`
类型	QoS 规范	请参见以下示例

选项	Description	示例
LegendPrefix	升级后的 Trident 安装的前缀。如果您使用的是1.3.2之前的版本的Trident并对现有卷执行升级、则需要设置此值才能访问通过volume-name方法映射的旧卷。	netappdvp-

solidfire-san 驱动程序不支持 Docker Swarm 。

Element 软件配置文件示例

```
{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://admin:admin@192.168.160.3/json-rpc/8.0",
  "SVIP": "10.0.0.7:3260",
  "TenantName": "docker",
  "InitiatorIFace": "default",
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,
        "burstIOPS": 8000
      }
    },
    {
      "Type": "Gold",
      "Qos": {
        "minIOPS": 6000,
        "maxIOPS": 8000,
        "burstIOPS": 10000
      }
    }
  ]
}
```

已知问题和限制

查找有关将Trident与Docker结合使用时的已知问题和限制的信息。

将 **Trident Docker** 卷插件从旧版本升级到 **20.10** 及更高版本会导致升级失败，并且不会显示此类文件或目录错误。

临时解决策

1. 禁用插件。

```
docker plugin disable -f netapp:latest
```

2. 删除此插件。

```
docker plugin rm -f netapp:latest
```

3. 通过提供额外的 `config` 参数来重新安装插件。

```
docker plugin install netapp/trident-plugin:20.10 --alias netapp --grant
-all-permissions config=config.json
```

卷名称的长度必须至少为 **2** 个字符。

备注	这是 Docker 客户端的限制。客户端会将单个字符名称解释为 Windows 路径。 "请参见错误25773"。

Docker Swarm的某些行为会使**Trident**无法为每个存储和驱动程序组合提供支持。

- Docker Swarm 目前使用卷名称而非卷 ID 作为其唯一卷标识符。
- 卷请求会同时发送到 Swarm 集群中的每个节点。
- 卷插件(包括Trident)必须在Swarm集群中的每个节点上单独运行。由于ONTAP的工作方式以及和 `ontap-san`驱动程序的工作方式 `ontap-nas`、它们是唯一能够在这些限制下运行的驱动程序。

其余驱动程序会受到争用条件等问题的影响、这些条件可能会导致为单个请求创建大量卷、而没有明确的"获奖者"；例如、Element的一项功能允许卷具有相同的名称但ID不同。

NetApp 已向 Docker 团队提供反馈，但没有任何迹象表明将来可以采用。

如果要配置 **FlexGroup** ，则在第二个 **FlexGroup** 具有一个或多个与要配置的 **FlexGroup** 相同的聚合时， **ONTAP** 不会配置第二个 **FlexGroup** 。

最佳实践和建议

Trident 部署建议

部署Trident时、请遵循此处列出的建议。

部署到专用命名空间

"命名空间"在不同应用程序之间实现管理隔离、是资源共享的障碍。例如，一个命名空间中的 PVC 不能从另一个命名空间中使用。Trident为Kubenetes集群中的所有名称区提供PV资源、从而利用提升了Privileges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。请务必确保应用程序用户和管理应用程序无法访问 Trident 对象定义或 Pod 本身。

使用配额和范围限制来控制存储消耗

Kubernetes 具有两项功能，这些功能结合使用后，可提供一种功能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基础上实施全局容量和对象计数消耗限制以及特定于存储类的限制。此外，使用 "范围限制" 确保在将 PVC 请求转发给配置程序之前，该请求同时处于最小值和最大值范围内。

这些值是按命名空间定义的，这意味着每个命名空间都应定义符合其资源要求的值。有关信息，请参见此处 "如何利用配额"。

Trident 存储配置的最佳实践

NetApp产品组合中的每个存储平台都具有独特的功能、无论应用程序是容器化还是非容器化、都能从中受益。

平台概述

Trident 可与 ONTAP 和 Element 结合使用。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，应考虑应用程序和设备管理团队的需求。

您应遵循使用所使用协议的主机操作系统的基线最佳实践。或者，您也可以考虑将应用程序最佳实践（如果有）与后端，存储类和 PVC 设置结合使用，以便为特定应用程序优化存储。

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 的准则，容器化工作负载会占用 Trident 动态配置的卷。应考虑并评估每个问题在您的环境中的适用性。

使用专用于 **Trident** 的 **SVM**

Storage Virtual Machine （ SVM ）可在 ONTAP 系统上的租户之间实现隔离和管理隔离。通过将 SVM 专用于应用程序，可以委派特权并应用最佳实践来限制资源消耗。

可通过多种方法管理 SVM：

- 在后端配置中提供集群管理接口以及相应的凭据，并指定 SVM 名称。
- 使用 ONTAP 系统管理器或命令行界面为 SVM 创建专用管理接口。
- 与 NFS 数据接口共享管理角色。

在每种情况下，接口都应位于 DNS 中，配置 Trident 时应使用 DNS 名称。这有助于在不使用网络身份保留的情况下实施某些灾难恢复方案，例如 SVM-DR 。

在为 SVM 配置专用管理 LIF 或共享管理 LIF 之间没有任何偏好，但是，您应确保网络安全策略与您选择的方法一致。无论如何，管理 LIF 应可通过 DNS 访问，以实现最大的灵活性 "SVM-DR" 与 Trident 结合使用。

限制最大卷数

ONTAP 存储系统具有最大卷数，具体取决于软件版本和硬件平台。请参见 "NetApp Hardware Universe" 以 ONTAP 确定确切限制。当卷计数用尽时，配置操作不仅会对 Trident 失败，而且会对所有存储请求失败。

Trident 的 `ontap-nas` 和 `ontap-san` 驱动程序为创建的每个 Kubernetes 永久性卷（ PV ）配置一个 FlexVolume 。`ontap-nas-economy` 驱动程序会为每 200 个 PV 创建大约一个 FlexVolume （可在 50 到 300 之间配置）。`ontap-san-economy`-经济 驱动程序会为每 100 个 PV 创建大约一个 FlexVolume （可配置为 50 到 200 ）。要防止 Trident 占用存储系统上的所有可用卷，您应对 SVM 设置限制。您可以从命令行执行此操作：

```
vserver modify -vserver <svm_name> -max-volumes <num_of_volumes>
```

`max-volumes` 的值因环境的特定条件而异：

- ONTAP 集群中现有卷的数量
- 希望在 Trident 之外为其他应用程序配置的卷数
- Kubernetes 应用程序预期占用的永久性卷数

`max-volumes` 值是在 ONTAP 集群中的所有节点上配置的总卷数，而不是在单个 ONTAP 节点上配置的总卷数。因此，在某些情况下， ONTAP 集群节点所配置的 Trident 卷可能远远多于或少于其他节点。

例如、双节点ONTAP集群最多可托管2000个FlexVol卷。将最大卷数设置为 1250 似乎非常合理。但是、如果仅 "聚合"从一个节点分配给SVM、或者无法配置从一个节点分配的聚合(例如、由于容量原因)、则另一个节点将成为所有Trident配置的卷的目标。这意味着、在达到此值之前、可能会达到该节点的卷限制 `max-volumes`、从而影响使用该节点的Trident和其他卷操作。* 您可以通过确保将集群中每个节点的聚合分配给 Trident 使用的 SVM 来避免这种情况。 *

克隆卷

NetApp Trident在使用时支持克隆卷 `ontap-nas`， `ontap-san` ， 和 `solidfire-san`存储驱动程序。使用时 `ontap-nas-flexgroup`或者 `ontap-nas-economy`驱动程序不支持克隆。从现有卷创建新卷将创建一个新的快照。

警告	避免克隆与其他 StorageClass 关联的 PVC。请在同一 StorageClass 内执行克隆操作，以确保兼容性并防止出现意外行为。

限制 **Trident** 创建的卷的最大大小

要为 Trident 可以创建的卷配置最大大小，请在 `backend.json` 定义中使用 `limitVolumeSize` 参数。

除了控制存储阵列上的卷大小之外，您还应利用 Kubernetes 功能。

限制**Trident**创建的**FlexVol**的大小上限

要配置用作ONTAP SAN经济型驱动程序和ONTAP NAS经济型驱动程序池的FlexVol的最大大小、请 `limitVolumePoolSize`在 `backend.json`定义中使用参数。

配置 **Trident** 以使用双向 **CHAP**

您可以在后端定义中指定 CHAP 启动程序以及目标用户名和密码，并在 SVM 上启用 Trident CHAP 。使用 `useCHAP` 参数在后端配置中、Trident使用CHAP对ONTAP 后端的iSCSI连接进行身份验证。

创建并使用 **SVM QoS** 策略

利用应用于 SVM 的 ONTAP QoS 策略，限制 Trident 配置的卷可使用的 IOPS 数量。这有助于实现 "防止抢占资源" 或控制不足的容器，以使其不会影响 Trident SVM 以外的工作负载。

您可以通过几个步骤为 SVM 创建 QoS 策略。有关最准确的信息，请参见适用于您的 ONTAP 版本的文档。以下示例将创建一个 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 版本支持此功能，则可以考虑使用最低 QoS 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。如果存在与 Kubernetes 部署无关的其他工作负载，则应注意利用存储资源，以确保这些工作负载不会意外受到不利影响。
- 容器中运行的预期工作负载。如果 IOPS 要求较高的工作负载将在容器中运行，则 QoS 策略较低会导致出现不良体验。

请务必记住，在 SVM 级别分配的 QoS 策略会导致配置到 SVM 的所有卷共享同一个 IOPS 池。如果一个或少量容器化应用程序的 IOPS 要求较高，则可能会成为其他容器化工作负载的抢占资源的应用程序。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。

重要说明	如果 ONTAP 版本早于 9.8 ，则应将此 QoS 策略组分配给 SVM * 仅 * 。

为 **Trident** 创建 **QoS** 策略组

服务质量（ QoS ）可确保关键工作负载的性能不会因争用工作负载而降级。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、请参见 "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，这些策略组将应用于该池或后端创建的每个卷。

ONTAP 有两种类型的 QoS 策略组：传统和自适应。传统策略组以 IOPS 为单位提供固定的最大（或最小）吞吐量。自适应 QoS 会根据工作负载大小自动扩展吞吐量，并在工作负载大小发生变化时保持 IOPS 与 TBSGB 的比率。如果您要在大型部署中管理数百或数千个工作负载，则这将带来显著优势。

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` 密钥。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
```

- 您应该对每个卷应用策略组，以便每个卷都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，请参见 "ONTAP命令参考"。

将存储资源访问限制为 **Kubernetes** 集群成员

限制对Trident创建的NFS卷、iSCSI LUN和FC LUN的访问是Kubbernetes部署安全防护的一个重要组成部分。这样可以防止不属于 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。假设同一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，但在绑定到 PVC 时，它们只能由同一命名空间中的 Pod 访问。* 请务必确保使用命名空间在适当时提供分隔。 *

大多数组织在 Kubernetes 环境中的数据安全方面的主要顾虑是，容器中的进程可以访问挂载到主机但不适用于

容器的存储。 "命名空间" 旨在防止这种类型的损害。但是，存在一个例外：特权容器。

有权限的容器是指运行时拥有比正常情况更多主机级别权限的容器。默认情况下，这些选项不会被拒绝，因此请确保使用禁用此功能 "POD 安全策略"。

对于需要从 Kubernetes 和外部主机访问的卷，应采用传统方式管理存储，并由管理员引入 PV ，而不是由 Trident 管理。这样可以确保只有在 Kubernetes 和外部主机断开连接且不再使用此卷时，才会销毁此存储卷。此外，还可以应用自定义导出策略，以便从 Kubernetes 集群节点和 Kubernetes 集群以外的目标服务器进行访问。

对于具有专用基础架构节点(例如OpenShift)或其他无法计划用户应用程序的节点的部署、应使用单独的导出策略进一步限制对存储资源的访问。其中包括为部署到这些基础架构节点的服务（例如 OpenShift 指标和日志记录服务）以及部署到非基础架构节点的标准应用程序创建导出策略。

使用专用导出策略

您应确保每个后端都有一个导出策略，该策略仅允许访问 Kubernetes 集群中的节点。 {\f270可以自动创建和管理导出策略} {\f151。} 通过这种方式， Trident 会限制对其配置给 Kubernetes 集群中节点的卷的访问，并简化节点的添加 / 删除。

或者，您也可以手动创建导出策略，并使用一个或多个导出规则来填充此策略，这些导出规则用于处理每个节点访问请求：

- 使用 `vserver export-policy create` ONTAP 命令行界面命令创建导出策略。
- 使用 `vserver export-policy rule create` ONTAP 命令行界面命令向导出策略添加规则。

通过运行这些命令，您可以限制哪些 Kubernetes 节点可以访问数据。

禁用 **`showmount`** 用于应用程序**SVM**

通过此 `showmount`功能、NFS客户端可以向SVM查询可用NFS导出的列表。部署到Kubnetes集群的Pod可以对发出 `showmount -e`命令、并接收可用挂载列表、包括其无权访问的挂载。虽然这本身并不会影响安全，但它确实会提供不必要的信息，可能有助于未经授权的用户连接到 NFS 导出。

您应使用 SVM 级别的 ONTAP 命令行界面命令禁用 `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** 帐户

每个 SolidFire 帐户都代表一个唯一的卷所有者，并接收自己的一组质询握手身份验证协议（ Challenge-Handshake Authentication Protocol ， CHAP ）凭据。您可以使用帐户名称和相对 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 2 ， 000 个卷，但一个卷只能属于一个帐户。

创建 QoS 策略

如果要创建并保存可应用于多个卷的标准化服务质量设置，请使用 SolidFire 服务质量（ QoS ）策略。

您可以按卷设置 QoS 参数。通过设置三个可配置的参数来定义 QoS ，可以确保每个卷的性能：最小 IOPS ，最大 IOPS 和突发 IOPS 。

以下是 4 KB 块大小的可能最小，最大和突发 IOPS 值。

IOPS 参数	定义	最小value	默认值	最大值（ 4 KB ）
最小 IOPS	卷的性能保障级别。	50	50	15000
最大 IOPS	性能不会超过此限制。	50	15000	200,000
突发 IOPS	在短时突发情形下允许的最大 IOPS 。	50	15000	200,000

备注	虽然最大 IOPS 和突发 IOPS 可设置为高达 200 ， 000 ，但卷的实际最大性能受集群使用情况和每节点性能的限制。

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 协议向后端对主机进行身份验证。卷访问组控制对其配置的卷的访问。NetApp 建议使用 CHAP 进行身份验证，因为它更简单，并且没有扩展限制。

备注	具有增强型 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 操作模式下使用。

只有基于帐户的访问控制才支持 CHAP 身份验证（验证启动程序是否为目标卷用户）。如果使用 CHAP 进行身份验证，则可以使用两个选项：单向 CHAP 和双向 CHAP 。单向 CHAP 使用 SolidFire 帐户名称和启动程序密钥对卷访问进行身份验证。双向 CHAP 选项可提供最安全的卷身份验证方法，因为卷会通过帐户名称和启动程序密钥对主机进行身份验证，然后主机通过帐户名称和目标密钥对卷进行身份验证。

但是，如果无法启用 CHAP 且需要使用 VAG ，请创建访问组并将主机启动程序和卷添加到此访问组。添加到访问组的每个 IQN 都可以使用或不使用 CHAP 身份验证访问组中的每个卷。如果将 iSCSI 启动程序配置为使用 CHAP 身份验证，则会使用基于帐户的访问控制。如果 iSCSI 启动程序未配置为使用 CHAP 身份验证，则会使用卷访问组访问控制。

如何查找更多信息

下面列出了一些最佳实践文档。搜索 "NetApp 库" 对于最新版本。

- ONTAP *
- "NFS 最佳实践和实施指南"
- "SAN管理"(适用于iSCSI)

- "适用于 RHEL 的 iSCSI 快速配置"
- Element 软件 *
- "配置适用于 Linux 的 SolidFire"
- NetApp HCI *
- "NetApp HCI 部署前提条件"
- "访问 NetApp 部署引擎"
- 应用程序最佳实践信息 *
- "基于 ONTAP 的 MySQL 最佳实践"
- "基于 SolidFire 的 MySQL 最佳实践"
- "NetApp SolidFire 和 Cassandra"
- "SolidFire 上的 Oracle 最佳实践"
- "SolidFire 上的 PostgreSQL 最佳实践"

并非所有应用程序都有特定的准则，与您的 NetApp 团队合作并使用非常重要 "NetApp 库" 以查找最新文档。

将 Trident 与 Kubernetes 集成

要集成Trident、需要集成以下设计和架构要素：驱动程序选择和部署、存储类设计、虚拟池设计、永久性卷请求(PVC)对存储配置的影响、卷操作以及使用Trident部署OpenShift服务。

驱动程序选择和部署

为存储系统选择并部署后端驱动程序。

ONTAP 后端驱动程序

ONTAP 后端驱动程序可通过所使用的协议以及在存储系统上配置卷的方式来区分。因此、在确定要部署的驱动程序时、请仔细考虑。

更高级别的是，如果您的应用程序中的组件需要共享存储（多个 Pod 访问同一个 PVC ），则基于 NAS 的驱动程序将成为默认选项，而基于块的 iSCSI 驱动程序则可满足非共享存储的需求。根据应用程序要求以及存储和基础架构团队的舒适程度选择协议。一般来说，对于大多数应用程序来说，它们之间没有什么区别，因此通常是根据是否需要共享存储（多个 POD 需要同时访问）来决定的。

可用的ONTAP 后端驱动程序包括：

- `ontap-NAS`： 配置的每个 PV 都是一个完整的 ONTAP FlexVolume 。
- `ontap-nas-economy.`： 配置的每个 PV 都是一个 qtree ，每个 FlexVolume 的 qtree 数量是可配置的（默认值为 200 ）。
- `ontap-nas-flexgroup`： 使用配置为完整 ONTAP FlexGroup 的每个 PV ，以及分配给 SVM 的所有聚合。
- `ontap-san`： 配置的每个 PV 都是其自身 FlexVolume 中的一个 LUN 。

- `ontap-san-economi`：配置的每个 PV 都是一个 LUN ，每个 FlexVolume 具有可配置的 LUN 数量（默认值为 100 ）。

在三个 NAS 驱动程序之间进行选择会对应用程序可用的功能产生一些影响。

请注意、在下表中、并非所有功能都通过Trident公开。如果需要某些功能，存储管理员必须在配置后应用这些功能。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	快照	克隆	动态导出策略	多连接	**QoS**	调整大小	**Replication**
`ontap-NAS`	是的。	是的。	是脚注：5[]	是的。	是脚注：1[]	是的。	是脚注：1[]
`ontap-nas-economy.`	注：3[]	注：3[]	是脚注：5[]	是的。	注：3[]	是的。	注：3[]
`ontap-nas-flexgroup`	是脚注：1[]	否	是脚注：5[]	是的。	是脚注：1[]	是的。	是脚注：1[]

Trident为ONTAP提供了2个SAN驱动程序、其功能如下所示。

ONTAP SAN 驱动程序	快照	克隆	多连接	双向 **CHAP**	**QoS**	调整大小	**Replication**
`ontap-san`	是的。	是的。	是脚注：4[]	是的。	是脚注：1[]	是的。	是脚注：1[]
`ontap-san-economy.`	是的。	是的。	是脚注：4[]	是的。	注：3[]	是的。	注：3[]

> 上表的脚注：是脚注：1[]：不受Trident管理是脚注：2[]：受Trident管理、但不受PV粒度注：3[]：不受Trident管理、不受PV粒度是是脚注：4[]：支持原始块卷脚注：5[]：受Trident支持

非 PV 粒度功能将应用于整个 FlexVolume ，而所有 PV （即共享 FlexVol 中的 qtree 或 LUN ）将共享一个通用计划。

如上表所示， `ontap-nas` 与 `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，因此这可能会特别影响灾难恢复和备份规划。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`-经济 驱动程序时，才可能实现这一点。

备注	`solidfire-san` 驱动程序还能够克隆 PVC 。

Cloud Volumes ONTAP 后端驱动程序

Cloud Volumes ONTAP 可为各种使用情形提供数据控制以及企业级存储功能，包括文件共享和为 NAS 和 SAN 协议（ NFS ， SMB/CIFS 和 iSCSI ）提供服务的块级存储。Cloud Volume ONTAP 的兼容驱动程序包括 `ontap-nas` ， `ontap-nas-economy`. `ontap-san` 和 `ontap-san-economy.`它们适用于适用于 Azure 的 Cloud Volume ONTAP ，适用于 GCP 的 Cloud Volume ONTAP 。

适用于**ONTAP** 的**Amazon FSX**后端驱动程序

借助Amazon FSx for NetApp ONTAP、您可以利用您熟悉的NetApp功能、性能和管理功能、同时还可以利用在AWS上存储数据的精简性、灵活性、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 的兼容驱动程序包括 `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 平台结合使用的 `solidfire-san` 驱动程序可帮助管理员根据 QoS 限制为 Trident 配置 Element 后端。如果您希望设计后端，以便为 Trident 配置的卷设置特定的 QoS 限制，请在后端文件中使用 `type` 参数。管理员还可以使用 `limitVolumeSize` 参数限制可在存储上创建的卷大小。目前， `solidfire-san` 驱动程序不支持卷大小调整和卷复制等 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	快照	克隆	多连接	**CHAP**	**QoS**	调整大小	**Replication**
`solidfire-san`	是的。	是的。	是脚注：2[]	是的。	是的。	是的。	是脚注：1[]

> 脚注：是脚注：1[]：不受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	快照	克隆	多连接	**QoS**	展开	**Replication**
`azure-netapp-files`	是的。	是的。	是的。	是的。	是的。	是脚注： 1[]

> 脚注：是脚注：1[]：不受Trident管理

Google Cloud NetApp Volumes 后端驱动程序

Trident 支持用于 NAS 工作负载的 `google-cloud-netapp-volumes`驱动程序和用于 `SAN（iSCSI）工作负载的 `google-cloud-netapp-volumes-san`。在 GCP 上运行的 OpenShift 集群上，从 Trident 26.10 开始，这两个驱动程序都支持 Google Cloud Workload Identity Federation。这包括自我管理的 OpenShift 容器平台（OCP，包括 IPI 部署）和 OpenShift Dedicated（OSD）。

有关后端配置和 WIF 先决条件，请参见"配置 Google Cloud NetApp Volumes"和"为 SAN 工作负载配置 Google Cloud NetApp Volumes"。

存储类设计

要创建 Kubernetes 存储类对象，需要配置并应用各个存储类。本节讨论如何为您的应用程序设计存储类。

特定后端利用率

可以在特定存储类对象中使用筛选功能来确定要将哪个存储池或一组池与该特定存储类结合使用。可以在存储类中设置三组筛选器： `storagePools` ， `addtionalStoragePools` 和 / 或 `excludeStoragePools` 。

`storagePools`参数可帮助将存储限制为与任何指定属性匹配的池集。
`additionalStoragePools`参数用于扩展Trident用于配置的池集以及通过属性和参数选择的池集
`storagePools`。您可以单独使用参数，也可以同时使用这两个参数，以确保选择适当的存储池集。

`excludeStoragePools` 参数用于明确排除列出的一组与属性匹配的池。

模拟**QoS**策略

如果要设计存储类以模拟服务质量策略，请创建一个存储类，其中 `mmedia` attribute 为 `HDD` 或 `sSD` 。根据存储类中提及的 `mmedia` 属性， Trident 将选择提供 `HDD` 或 `sSD` 聚合的相应后端，以便与介质属性匹配，然后将卷配置定向到特定聚合。因此，我们可以创建一个存储类高级版，该高级版会将 `mmedia` attribute 设置为 `sSD` ，该高级版可归类为高级 QoS 策略。我们可以创建另一个存储类标准，该标准会将介质属性设置为 `HDD' ，并可归类为标准 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定义为 QoS 策略的 Element 设备。

根据特定功能使用后端

存储类可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直接配置卷。要指定要使用的存储，请创建存储类，以指定启用了所需功能的相应后端。

虚拟池

所有Trident后端均可使用虚拟池。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。

通过虚拟池、管理员可以在后端创建一个抽象级别、并可通过存储类进行引用、从而提高卷在后端的灵活性和效率。可以使用相同的服务类定义不同的后端。此外、可以在同一后端创建多个存储池、但其特征不同。如果为存储类配置了具有特定标签的选择器、则Trident会选择一个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、则Trident将选择其中一个存储池来配置卷。

虚拟池设计

创建后端时，通常可以指定一组参数。管理员无法创建具有相同存储凭据和不同参数集的另一个后端。随着虚拟池的引入，这个问题得到了缓解。虚拟池是在后端和 Kubernetes 存储类之间引入的级别抽象，以便管理员可以定义参数以及可以通过 Kubernetes 存储类作为选择器引用的标签，以与后端无关的方式。可以使用Trident为所有受支持的NetApp后端定义虚拟池。该列表包括SolidFire/ NetApp HCI、 ONTAP以及Azure NetApp Files。

备注	定义虚拟池时、建议不要尝试在后端定义中重新排列现有虚拟池的顺序。此外，建议不要编辑 / 修改现有虚拟池的属性，而是定义新的虚拟池。

模拟不同的服务级别/**QoS**

可以为模拟服务类设计虚拟池。使用适用于 Azure NetApp Files 的云卷服务的虚拟池实施，让我们来了解一下

如何设置不同的服务类。使用多个标签配置Azure NetApp Files后端、以表示不同的性能级别。设置 `servicelevel` 添加适当的性能级别、并在每个标签下添加其他所需的方面。现在、创建可映射到不同虚拟池的不同Kubernetes存储类。使用 `parameters.selector` 字段中、每个StorageClass都会调用可用于托管卷的虚拟池。

分配特定的方面

可以从一个存储后端设计具有一组特定方面的多个虚拟池。为此，请为后端配置多个标签，并在每个标签下设置所需的方面。现在、使用创建不同的Kubernetes存储类 `parameters.selector` 要映射到不同虚拟池的字段。在后端配置的卷将在选定虚拟池中定义相关方面。

影响存储配置的 **PVC** 特征

创建PVC时、请求的存储类以外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，访问模式为必填字段之一。所需的模式可能会影响所选的托管存储请求的后端。

Trident 将尝试与根据下表指定的访问方法所使用的存储协议匹配。这独立于底层存储平台。

	ReadWriteOnce	**ReadOnlyMany**	读取写入任何
iSCSI	是的。	是的。	是（原始块）
NFS	是的。	是的。	是的。

如果在未配置 NFS 后端的情况下向 Trident 部署提交了 ReadWriteMany PVC 请求，则不会配置任何卷。因此，请求者应使用适合其应用程序的访问模式。

卷操作

修改永久性卷

除了两个例外，永久性卷是 Kubernetes 中不可变的对象。创建后，可以修改回收策略和大小。但是、这并不会阻止在Kubnetes之外修改卷的某些方面。为了针对特定应用程序自定义卷，确保容量不会意外占用，或者出于任何原因将卷移动到其他存储控制器，这一点可能是理想的。

备注	Kubernetes树内配置程序目前不支持对NFS、iSCSI或FC PVs执行卷大小调整操作。Trident支持扩展NFS、iSCSI和FC卷。

创建 PV 后，无法修改其连接详细信息。

创建按需卷快照

Trident 支持按需创建卷快照，并使用 CSI 框架从快照创建 PVC 。快照提供了一种维护数据时间点副本的便捷方法，并且生命周期独立于 Kubernetes 中的源 PV 。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 。为此、只需创建一个PerseentVolumeClaim、并将 `datasource`作为需要从中创建卷的所需快照。Trident 将使用快照上的数据创建卷来处理此 PVC 。通过此功能

，可以跨区域复制数据，创建测试环境，整体更换损坏或损坏的生产卷，或者检索特定文件和目录并将其传输到另一个连接的卷。

移动集群中的卷

存储管理员可以在 ONTAP 集群中的聚合和控制器之间无中断地将卷移动到存储使用者。只要目标聚合是Trident正在使用的SVM有权访问的聚合、此操作不会影响Trident或Kubornetes集群。重要的是、如果此聚合已新添加到SVM中、则需要通过将其重新添加到Trident来刷新后端。此操作将触发 Trident 来重新清点 SVM ，以便识别新聚合。

但是， Trident 不支持在后端之间自动移动卷。这包括在同一集群中的 SVM 之间，集群之间或不同存储平台上（即使该存储系统是连接到 Trident 的存储系统也是如此）。

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到 Trident 中。

展开卷

Trident支持调整 NFS、iSCSI 和 FC PV 的大小。这样用户就可以直接通过 Kubernetes 层调整卷的大小。所有主流NetApp存储平台（包括ONTAP）和SolidFire/ NetApp HCI后端均可进行卷扩展。为了便于日后扩展，请设置 `allowVolumeExpansion`到 `true`在与该卷关联的 StorageClass 中。每当需要调整持久卷的大小时，请编辑以下内容： `spec.resources.requests.storage`在持久卷声明中添加对所需卷大小的注释。Trident会自动处理存储集群上卷的大小调整。

将现有卷导入到 **Kubernetes** 中

卷导入功能允许将现有存储卷导入 Kubernetes 环境。目前这得到了以下方面的支持： `ontap-nas` ， `ontap-nas-flexgroup` ， `solidfire-san` ， 和 `azure-netapp-files`司机。将现有应用程序移植到 Kubernetes 或灾难恢复场景中，此功能非常有用。

使用ONTAP和驱动程序时 `solidfire-san`、请使用命令 `tridentctl import volume <backend-name> <volume-name> -f /path/pvc.yaml`将现有卷导入到要由Trident管理的Kubnetes中。导入卷命令中使用的 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。如果卷名称重复，请将卷克隆为唯一名称，以便卷导入功能可以区分它们。

如果 `azure-netapp-files`驱动程序已启用，请使用该命令 `tridentctl import volume <backend-name> <volume path> -f /path/pvc.yaml`将卷导入 Kubernetes 以便由Trident管理。这样就确保了卷号的唯一性。

执行上述命令后， Trident 将在后端找到卷并读取其大小。它会自动添加(并在必要时覆盖)已配置的PVC卷大小。然后， Trident 会创建新的 PV ， Kubernetes 会将 PVC 绑定到 PV 。

如果部署的容器需要特定的导入 PVC ，则容器将保持待定状态，直到通过卷导入过程绑定 PVC/PV 对为止。在绑定 PVC/PV 对后，如果没有其他问题，应启动容器。

注册表服务

有关为注册表部署和管理存储的文档，请参见 "netapp.io" 在中 "博客"。

日志记录服务

与其他 OpenShift 服务一样，日志记录服务也是使用清单文件（也称为）提供的配置参数 Ansible 部署的主机，提供给攻略手册。其中包括两种安装方法：在初始 OpenShift 安装期间部署日志记录以及在安装 OpenShift 之后部署日志记录。

注意	自 Red Hat OpenShift 3.9 版开始，官方文档出于对数据损坏的担忧，建议不要对日志记录服务使用 NFS 。这是基于 Red Hat 对其产品的测试得出的。ONTAP NFS服务器不存在这些问题、可以轻松地备份日志记录部署。最终，您可以选择日志记录服务的协议，只需了解这两种协议在使用 NetApp 平台时都能很好地发挥作用，如果您愿意，也没有理由避免使用 NFS 。

如果您选择将 NFS 与日志记录服务结合使用，则需要将 Ansible 变量 `OpenShift_enable_unsupported _configurations` 设置为 `true` ，以防止安装程序失败。

入门

可以选择为这两个应用程序以及 OpenShift 集群本身的核心操作部署日志记录服务。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` ，将创建两个服务实例。控制操作日志记录实例的变量包含 "ops" ，而应用程序实例则不包含 "ops" 。

要确保底层服务使用正确的存储、请务必根据部署方法配置Ansensant变量。让我们来看看每种部署方法的选项。

备注	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 攻略手册使用提供的详细信息为日志记录服务创建 PV 和 PVC 。与在 OpenShift 安装后使用组件安装攻略手册相比，此方法的灵活性明显降低，但是，如果您有可用的现有卷，则可以选择此方法。

变量	详细信息
`OpenShift_logging_storage_kind`	设置为 `NFS` ，以使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。此值应设置为虚拟机的dataLIF。
`OpenShift_logging_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷接合为 `/OpenShift_logging` ，则应使用该路径作为此变量。
`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 OpenShift 集群已在运行，因此已部署和配置 Trident ，则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_vc_dynamic`	设置为 true 可使用动态配置的卷。
`OpenShift_logging_es_vc_storage_class_name`	要在 PVC 中使用的存储类的名称。
`OpenShift_logging_es_vc_size`	在 PVC 中请求的卷大小。
`OpenShift_logging_es_vc_prefix`	日志记录服务使用的 PVC 的前缀。
`OpenShift_logging_es_ops_vc_dynamic`	设置为 `true` 可对操作日志记录实例使用动态配置的卷。

变量	详细信息
`OpenShift_logging_es_ops_vc_storage_class_name`	操作日志记录实例的存储类的名称。
`OpenShift_logging_es_ops_vc_size`	操作实例的卷请求大小。
`OpenShift_logging_es_ops_vc_prefix`	操作实例 PVC 的前缀。

部署日志记录堆栈

如果要在初始 OpenShift 安装过程中部署日志记录，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 完成后立即提供此服务。

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

指标服务可为管理员提供有关 OpenShift 集群的状态，资源利用率和可用性的宝贵信息。此外、POD自动扩展功能也需要使用此功能、许多组织会将来自指标服务的数据用于其成本分摊和/或成本分摊应用程序。

与日志记录服务和 OpenShift 作为一个整体一样， Ansible 用于部署指标服务。此外、与日志记录服务一样、指标服务也可以在集群初始设置期间或集群运行后使用组件安装方法进行部署。下表包含在为指标服务配置永久性存储时非常重要的变量。

备注	下表仅包含与存储配置相关的变量，因为这些变量与指标服务相关。文档中还有许多其他选项，应根据您的部署情况进行查看，配置和使用。

变量	详细信息
`OpenShift_metrics_storage_kind`	设置为 `NFS` ，以使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。此值应设置为SVM的dataLIF。
`OpenShift_metrics_storage_nfs_directory`	NFS 导出的挂载路径。例如，如果卷接合为 `/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 OpenShift 集群已在运行，因此已部署和配置 Trident ，则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_vc_prefix`	用于衡量指标 PVC 的前缀。
`OpenShift_metrics_Cassandra_vc_size`	要请求的卷的大小。
`OpenShift_metrics_Cassandra_storage_type`	要用于度量指标的存储类型，必须将此类型设置为动态， Ansible 才能创建具有相应存储类的 PVC 。

变量	详细信息
`OpenShift_metrics_cassanda_vc_storage_class_name`	要使用的存储类的名称。

部署指标服务

使用在主机 / 清单文件中定义的适当 Ansible 变量，使用 Ansible 部署服务。如果您在 OpenShift 安装时进行部署，则系统将自动创建和使用 PV 。如果您要使用组件操作手册进行部署、则在安装OpenShift后、Ansable会创建所需的任何PVC、并在Trident为其配置存储后部署该服务。

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。

Trident 数据保护和灾难恢复

了解Trident以及使用Trident创建的卷的保护和恢复选项。对于具有持久性要求的每个应用程序，您都应制定一个数据保护和恢复策略。

Trident复制和恢复

您可以创建备份、以便在发生灾难时还原Trident。

Trident复制

Trident使用Kubnetes CRD存储和管理自己的状态、并使用Kubnetes集群etCD存储其元数据。

步骤

1. 使用备份Kubenetes集群etcd "Kubenetes：备份etcd集群"。
2. 将备份项目放在FlexVol volume上

备注	NetApp建议您保护与另一个SVM具有SnapMirror关系的FlexVol所在的SVM。

Trident恢复

您可以使用Kubenetes CRD和Kubenetes集群etCD快照恢复Trident。

步骤

1. 从目标SVM中、将包含Kubnetes 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. 在下复制与Kubirnetes集群相关的所有必需证书 `/etc/kubernetes/pki` 和下的etcd成员文件 `/var/lib/etcd`。
3. 使用从etcd备份还原Kubenetes集群 "Kubornetes：还原etcd集群"。
4. 运行 `kubectl get crd` 验证所有的三端测试自定义资源是否已启动、并检索三端测试对象以验证所有数据是否可用。

SVM复制和恢复

Trident无法配置复制关系、但存储管理员可以使用 "ONTAP SnapMirror"来复制SVM。

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：

- 您应为启用了SVM-DR的每个SVM创建一个不同的后端。
- 将存储类配置为仅在需要时选择复制的后端、以避免将不需要复制的卷配置到支持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" 以创建SVM复制关系。

使用SnapMirror、您可以设置选项来控制要复制的内容。您需要知道在预成形时选择了哪些选项使用Trident恢复SVM。

- "-Identity保留true" 复制整个SVM配置。
- "-discard-configs network" 不包括LIP和相关网络设置。
- "-Identity保留false" 仅复制卷和安全配置。

使用Trident恢复SVM

Trident不会自动检测SVM故障。如果发生灾难、管理员可以手动启动通过三项功能故障转移到新SVM的操作。

步骤

1. 取消计划的和正在进行的SnapMirror传输、中断复制关系、停止源SVM、然后激活SnapMirror目标SVM。
2. 如果指定了 `-identity-preserve false` 或 `-discard-config network` 配置SVM复制时、请更新 `managementLIF` 和 `dataLIF` 在三端定义文件中。
3. 确认 `storagePrefix` 位于三端定义文件中。无法更改此参数。正在放弃 `storagePrefix` 发生原因后端更新是否会失败。
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`，则必须退回所有应用程序Pod。

备注	如果指定了 `-identity-preserve true`，则在激活目标SVM后，Trident配置的所有卷都会开始提供数据。

卷复制和恢复

Trident无法配置SnapMirror复制关系、但是存储管理员可以使用"ONTAP SnapMirror复制和恢复"来复制Trident创建的卷。

然后，您可以使用将恢复的卷导入到Trident中"tridentctrd卷导入"。

备注	`ontap-san-economy` 驱动程序支持卷导入。使用 `ontap-nas-economy` 驱动程序导入 Qtree 是一项技术预览功能，不支持在生产环境中使用。 `ontap-flexgroup-economy` 驱动程序不支持导入。有关详细信息，请参见 "导入 Trident 卷"。

Snapshot数据保护

您可以使用以下方式保护和还原数据：

- 外部快照控制器和CRD、用于为永久性卷(PVs)创建Kubbernetes卷快照。

  "卷快照"

- ONTAP快照、用于还原卷的全部内容或恢复单个文件或LUN。

  "ONTAP快照"

使用Trident实现有状态应用程序的故障转移自动化

Trident 的强制分离功能允许您自动将卷从 Kubernetes 集群中不健康的节点分离，从而防止数据损坏并确保应用程序的可用性。此功能在节点无响应或因维护而离线的情况下特别有用。

有关强制断开的详细信息

强制分离适用于 `ontap-san`， `ontap-san-economy` ， `ontap-nas` ， 和 `ontap-nas-economy`仅有的。在启用强制分离之前，必须在 Kubernetes 集群上启用非正常节点关闭 (NGNS)。Kubernetes 1.28 及以上版本默认启用 NGNS。有关详细信息，请参阅"Kubnetes：节点非正常关闭" 。

备注	使用或驱动程序时 `ontap-nas`、您需要将后端配置中的参数设置 `autoExportPolicy`为 `true`、以便Trident可以限制通过受管导出策略应用了此内容的Kubornetes `ontap-nas-economy`节点的访问。
警告	由于Trident依赖于Kubbernetes NGN、因此在重新计划所有不可允许的工作负载之前、请勿从运行状况不正常的节点中删除 `out-of-service`污点。不负责任地应用或删除该问题可能会危及后端数据保护。

当Kubnetes集群管理员已将此标记应用 `node.kubernetes.io/out-of-service=nodeshutdown:NoExecute`于节点并 `enableForceDetach`设置为 `true`时，Trident将确定节点状态并：

1. 停止对挂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`(对于新出版物不安全)。

备注	Trident控制器将拒绝新的发布卷请求，直到节点被Trident节点POD重新认定(标记为后)为止 `dirty`。使用挂载的PVC计划的任何工作负载(即使在集群节点运行状况良好且已准备就绪之后)都将无法接受、直到Trident能够验证该节点 `clean`(对于新发布的产品来说是安全的)。

在恢复节点运行状况并删除此污染后、Trident将：

1. 确定并清除节点上陈旧的已发布路径。
2. 如果此节点处于某个 `cleanable`状态(已删除服务中断、并且此节点处于 `Ready`状态)、并且所有陈旧的已发布路径均已清理、则`Trident`会将此节点重新提交为、并允许新的已发布卷访问此节点 `clean`。

有关自动故障转移的详细信息

您可以通过与以下系统的集成来自动执行强制分离过程："节点健康检查（NHC）操作符" 。当节点发生故障时，NHC 会通过在 Trident 的命名空间中创建 TridentNodeRemediation CR 来定义故障节点，从而触发Trident节点修复 (TNR) 并自动强制分离。TNR 仅在节点发生故障时创建，并在节点恢复上线或节点被删除后由 NHC 删除。

节点 **Pod** 移除过程失败

自动故障转移会选择要从故障节点移除的工作负载。创建 TNR 时，TNR 控制器会将节点标记为脏节点，阻止任何新的卷发布，并开始移除强制分离支持的 pod 及其卷附件。

所有受强制分离支持的卷/PVC均受自动故障转移支持：

- NAS 和使用自动导出策略的 NAS 经济型卷（尚不支持 SMB）。
- SAN 和 SAN 经济型卷。

参考[有关强制断开的详细信息]。

默认行为：

- 使用 force-detach 支持的卷的 Pod 将从故障节点中移除。Kubernetes 会将这些任务重新调度到健康的节点上。
- 使用不支持强制分离的卷（包括非 Trident 卷）的 Pod 不会从故障节点中移除。
- 无状态 Pod（非 PVC）不会从故障节点中移除，除非 Pod 注解另有规定。 `trident.netapp.io/podRemediationPolicy: delete`已设置。

覆盖 **pod** 移除行为：

可以使用 Pod 注解来自定义 Pod 移除行为： `trident.netapp.io/podRemediationPolicy[retain, delete]` 。发生故障转移时，会检查并使用这些注解。在 Kubernetes 部署/副本集 Pod 规范中添加注解，以防止故障转移后注解消失：

- `retain`- 在自动故障转移期间，Pod 不会从故障节点中移除。
- `delete`- 在自动故障转移期间，Pod 将从故障节点中移除。

这些注解可以应用于任何 pod。

警告	• 仅当发生故障的节点支持强制分离时，I/O 操作才会被阻塞。

- 对于不支持强制分离的卷，存在数据损坏和多重附加问题的风险。

TridentNode修复CR

TridentNodeRemediation (TNR) CR 定义了一个故障节点。TNR 的名称就是故障节点的名称。

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状态：
`kubectl get tnr <name> -n <trident-namespace>`

TNR（诱捕、绝育、放归）可能处于以下几种状态之一：

- 修复中：
  - 停止对强制分离挂载到该节点的卷的后端 I/O 访问。
  - Trident节点对象被标记为脏（不适合发布新内容）。
  - 从节点中移除 Pod 和卷附件
- *NodeRecoveryPending*:
  - 控制器正在等待节点重新上线。
  - 一旦节点上线，发布强制执行将确保节点干净且已准备好发布新卷。
- 如果节点从 K8s 中删除，TNR 控制器将移除 TNR 并停止协调。
- 成功：
  - 所有修复和节点恢复步骤均已成功完成。该节点已清理完毕，可以发布新的卷。
- 失败的：
  - 无法恢复的错误。错误原因设置在 CR 的状态消息字段中。

启用自动故障转移

先决条件：

- 请确保在启用自动故障转移之前已启用强制分离。更多信息，请参阅[有关强制断开的详细信息]。
- 在 Kubernetes 集群中安装节点健康检查 (NHC)。
  - "安装 operator-sdk"。
  - 如果集群中尚未安装 Operator Lifecycle Manager (OLM)，请安装它： `operator-sdk olm install` 。
  - 安装节点健康检查运算符： `kubectl create -f https://operatorhub.io/install/node-`

`healthcheck-operator.yaml` 。

备注	您还可以使用其他方法来检测节点故障，如下面的 [集成自定义节点健康检查解决方案] 部分所述。

看"节点健康检查操作符"了解更多信息。

步骤

1. 在Trident命名空间中创建 NodeHealthCheck (NHC) CR，以监控集群中的工作节点。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. 在节点健康检查 CR 中应用 `trident`命名空间。

```
kubectl apply -f <nhc-cr-file>.yaml -n <trident-namespace>
```

上述 CR 配置为监视 K8s 工作节点，以检测节点状态 Ready: false 和 Unknown。当节点进入 Ready: false 或 Ready: Unknown 状态时，将触发自动故障转移。

这 `unhealthyConditions`CR 中使用了 0 秒宽限期。这样一来，一旦 K8s 将节点状态 Ready: false 设置为 false（在 K8s 失去节点的心跳信号后设置），就会立即触发自动故障转移。K8s 在最后一次心跳后默认等待 40 秒，然后才将 Ready: false 设置为 false。该宽限期可在 K8s 部署选项中进行自定义。

有关其他配置选项，请参阅"节点健康检查操作符文档"。

其他设置信息

当Trident安装时启用了强制分离功能， Trident命名空间中会自动创建两个额外的资源，以方便与 NHC 集成：TridentNodeRemediationTemplate (TNRT) 和 ClusterRole。

TridentNodeRemediationTemplate (TNRT)：

TNRT 可作为 NHC 控制器的模板，NHC 控制器可根据需要使用 TNRT 生成 TNR 资源。

```
apiVersion: trident.netapp.io/v1
kind: TridentNodeRemediationTemplate
metadata:
  name: trident-node-remediation-template
  namespace: trident
spec:
  template:
    spec: {}
```

集群角色：

启用强制分离功能时，安装过程中还会添加集群角色。这使得 NHC 能够对Trident命名空间中的 TNR 进行授权。

```
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

为防止发生故障转移，在 K8s 维护或升级期间暂停自动故障转移，因为预计节点会停止运行或重新启动。您可以通过修改 NHC CR 的补丁来暂停该 CR（如上所述）：

```
kubectl patch NodeHealthCheck <cr-name> --patch
'{"spec":{"pauseRequests":["<description-for-reason-of-pause>"]}}' --type=merge
```

这将暂停自动故障转移。要重新启用自动故障转移，维护完成后，请从规范中删除 pauseRequests。

限制

- 仅对受强制分离支持的卷，在发生故障的节点上阻止 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分离在三叉戟控制器舱内运行。如果托管 trident-controller 的节点发生故障，自动故障转移将会延迟，直到 K8s 将 pod 迁移到健康的节点。

集成自定义节点健康检查解决方案

您可以将节点健康检查操作器替换为其他节点故障检测工具，以触发自动故障转移。为确保与自动故障转移机制兼容，您的自定义解决方案应：

- 当检测到节点故障时，创建 TNR，使用故障节点的名称作为 TNR CR 名称。
- 当节点恢复且 TNR 处于成功状态时，删除 TNR。

安全性

Trident 安全建议

请按照此处列出的建议确保Trident安装安全。

在自己的命名空间中运行Trident

请务必防止应用程序、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，请始终将Trident安装在其自身的Kubbernetes命名空间中(`trident`)。将Trident置于自己的命名空间中可确保只有Kubelnetes管理人员才能访问Trident Pod和存储在命名空间CRD对象中的项目(如适用、例如后端和CHAP密码)。您应确保仅允许管理员访问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支持对ONTAP SAN工作负载进行基于CHAP的身份验证(使用 `ontap-san`和 `ontap-san-economy`驱动程序)。NetApp建议对Trident使用双向CHAP在主机和存储后端之间进行身份验证。

对于使用SAN存储驱动程序的ONTAP后端，Trident可以通过设置双向CHAP并管理CHAP用户名和机密 `tridentctl`。要了解Trident如何在ONTAP后端配置CHAP、请参见"准备使用ONTAP SAN驱动程序配置后端"。

对 **NetApp HCI** 和 **SolidFire** 后端使用 **CHAP** 身份验证

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使用一个机密对象、其中每个租户包含两个CHAP密码。安装Trident后、它会管理CHAP密钥、并将其存储在相应PV的CR对象中 `tridentvolume`。创建PV时、Trident会使用CHAP密码启动iSCSI会话并通过CHAP与NetApp HCI和SolidFire系统进行通信。

备注	Trident创建的卷不与任何卷访问组关联。

将**Trident**与**NVE**和**NAE**结合使用

NetApp ONTAP 提供空闲数据加密、可在磁盘被盗、退回或重新利用时保护敏感数据。有关详细信息，请参见 "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 `""`、以创建启用了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NVE加密标志设置为 `false`(默认值)。

备注	在启用了NAE的后端的Trident中创建的卷必须已进行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加密并按卷使用特定的加密密钥。 • 如果在启用了NAE的后端将NVE加密标志设置为 `false`、则会创建启用了NAE的卷。您不能通过将NVE加密标志设置为来禁用NAE加密 `false`。

- 您可以通过将NVE加密标志显式设置为来在Trident中手动创建NVE卷 `true`。

有关后端配置选项的详细信息、请参见：

- "ONTAP SAN配置选项"
- "ONTAP NAS配置选项"

Trident Linux 统一密钥设置（LUKS）

Linux 统一密钥设置（LUKS）是一种磁盘加密规范。Trident 使用 LUKS 为 ONTAP SAN 和 ONTAP SAN ECONOMY 卷提供基于每个卷的主机端加密，并支持密码短语轮换和卷扩展。

在Trident中，根据的建议，经过LUN加密的卷使用AES-XTS-PLAN 64 cypher和模式"NIST"。

备注	ASA r2 系统不支持 LUKS 加密。有关 ASA r2 系统的信息，请参阅"了解ASA R2存储系统" 。

开始之前

- 工作节点必须安装加密设置2.1或更高版本(但低于3.0)。有关详细信息，请访问 "Gitlab：密码设置"。
- 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
```

如果未返回任何内容、则您的处理器不支持AES-NI。有关AES-NI的详细信息、请访问： "Intel：高级加密标准说明(AES-NI)"。

启用LUKS加密

您可以对ONTAP SAN和ONTAP SAN经济卷使用Linux统一密钥设置(Unified Key Setup、LUKS)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的后端配置选项的详细信息、请参见 "ONTAP SAN配置选项"。

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. 使用 ... `parameters.selector` 使用LUKS加密定义存储池。例如：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
```

```
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. 创建一个包含LUKS密码短语的密钥。例如：

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

将 **LUKS** 与 **RWX** 原始块卷一起使用

从 Trident 26.10 开始，您可以使用 `ReadWriteMany`（RWX）访问模式将 LUKS 加密与连接到多个 Kubernetes 节点的基于 iSCSI 的原始块卷结合使用。这对于 KubeVirt 虚拟机等工作负载非常有用，这些工作负载在实时迁移期间需要在目标节点上挂载相同的加密卷，而此时该卷仍挂载在源节点上。

RWX LUKS 支持仅限于原始块卷。不支持文件系统模式 RWX LUKS 卷。应用程序负责协调对解密数据的并发访问；Trident 仅协调 LUKS 操作。

按照启用LUKS加密中的说明，在后端、StorageClass和密码短语 Secret 上配置 LUKS。然后使用 `volumeMode: Block`和 `ReadWriteMany`创建 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-rwx-pvc
  namespace: my-app
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Block
  storageClassName: san-luks-rwx
  resources:
    requests:
      storage: 10Gi
```

多个节点上的 Pod 可以将此 PVC 作为原始块设备附加。Trident 在首次附加时使用 LUKS2 格式化该设备，并在后续节点上打开现有的加密卷。支持现有的 LUKS 操作，包括密码短语轮换和卷扩展。

为什么 Trident 序列化 LUKS 操作

LUKS2 在块设备的开头存储其加密标头的冗余副本。 `cryptsetup` 可以在检测到副本不同时修复标头，包括在执行看似只读的命令期间。它的锁定是节点本地的，无法协调其他节点上的操作。如果一个节点正在更新标头，而另一个节点正在运行 `cryptsetup`，则第二个节点可能会触发标头修复并中断更新，从而可能损坏标头。

为了防止并发标头访问， Trident 为每个 RWX LUKS 卷使用 Kubernetes 租约。在节点对该卷运行任何 `cryptsetup`命令之前，它必须持有该租约。Trident 在操作成功后释放该租约。如果在持有该租约期间另一个节点请求操作， Trident 会返回一个可重试的错误，并且 Kubernetes 会在稍后重试。不同卷上的操作是独立的。

如果 `cryptsetup` 操作失败，Trident 故意保留租约，因此只有同一个节点可以重试。带有原因 `LeaseRetained` 的警告事件会记录故障。租约不会自动过期；只有在确认其持有者节点上没有运行任何 `cryptsetup` 操作后，管理员才应删除租约。有关恢复步骤，请参见 "修复 RWX Linux 统一密钥设置（LUKS）卷的停滞租约"。

由于节点轮流打开卷，因此在许多不同节点上启动工作负载可能需要比使用未加密卷更长的时间。Kubernetes 在当前操作完成后会重试瞬态锁定错误。在大规模横向扩展期间，Pod 可能会在 `ContainerCreating`中停留更长时间；同一节点上的工作负载可以共享单个打开的设备映射。

限制

LUKS加密的卷无法利用ONTAP 重复数据删除和数据压缩功能。

用于导入**LUKS**卷的后端配置

要导入LUN卷、必须在后端将设置 `luksEncryption`为(`true。该 `luksEncryption`选项会告诉`Trident`卷是符合`LUN (`true`(兼容)还是不符合LUN (`false`(兼容)，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**UKS**卷的**PVC**配置

要动态导入LUKS卷、请将标注设置 `trident.netapp.io/luksEncryption`为 `true`、并在PVC中包含启用了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。

警告	请勿忘记密码短语、除非您确认任何卷、快照或密钥不再引用它。如果引用的密码短语丢失、您可能无法挂载此卷、并且数据将保持加密状态且无法访问。

关于此任务

如果在指定新的LUKS密码短语后创建了挂载卷的POD、则会发生LUKS密码短语轮换。创建新Pod时、Trident会将卷上的LUN密码短语与密钥中的活动密码短语进行比较。

- 如果卷上的密码短语与密钥中的活动密码短语不匹配、则会发生轮换。
- 如果卷上的密码短语与密钥中的活动密码短语匹配、则会显示 `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. 确定卷或快照上的现有密码短语。

Volume

```
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密钥以指定新密码短语和上一密码短语。确保 `previous-luke-passphrase-name` 和 `previous-luks-passphrase` 匹配上一个密码短语。

```
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. 创建一个新的装载卷的POD。这是启动轮换所必需的。
5. 验证密码短语是否已轮换。

Volume

```
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

仅在卷和快照上返回新密码短语时、才会轮换密码短语。

备注	如果返回两个密码短语、例如 `luksPassphraseNames: ["B", "A"]`、转出不完整。您可以触发新POD以尝试完成轮换。

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(测试版1.25以上)。请参见 "Kubernetes 1.25：使用机密进行节点驱动型CSI卷扩展" 了解详细信息。
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

启动联机存储扩展时、kubelet会将相应的凭据传递给驱动程序。

修复 RWX Linux 统一密钥设置（LUKS）卷的停滞租约

如果节点在持有 Kubernetes 租约时发生故障，ReadWriteMany (RWX) Linux 统一密钥设置 (LUKS) 卷可能会被阻止。在删除租约之前，请确认没有 `cryptsetup`操作仍在运行；在操作过程中删除租约可能会使 LUKS 标头面临风险。

了解租约

Trident 使用 Kubernetes Lease 来确保一次只有一个节点在 RWX LUKS 卷上运行 `cryptsetup`命令。租约在 Trident 命名空间中创建，租约名称为 PersistentVolume（PV）名称。持有者标识用于标识运行操作的节点。

Trident 在操作成功后删除租约。如果操作失败，Trident 会保留租约，以便只有同一个节点可以重试。租约不会自动过期或续订。带有原因 `LeaseRetained`的警告事件包含有关失败操作的详细信息。

识别可能卡住的租约

症状可能包括 Pod 卡在 `ContainerCreating`或 `Terminating`中、卷扩展失败或瞬时锁定错误。

1. 查找受影响的 pod 使用的 PVC：

```
kubectl get pod -n <app-namespace> <pod-name> \
  -o jsonpath='{.spec.volumes[*].persistentVolumeClaim.claimName}'
```

2. 查找绑定到 PVC 的 PV：

```
kubectl get pvc -n <app-namespace> <pvc-name> \
  -o jsonpath='{.spec.volumeName}'
```

3. 在 Trident 命名空间中列出租约，并使用 PV 名称标识租约：

```
kubectl get lease -n <trident-namespace>
```

4. 检查匹配的租约，包括其持有人、年龄和事件：

```
kubectl describe lease -n <trident-namespace> <pv-name>
```

`cryptsetup` 操作通常在几秒钟内完成。超过 5 至 10 分钟的租约可能表明存在死锁。检查 `LeaseRetained` 事件是否有重复错误，识别持有者节点，并检查节点健康状况：

```
kubectl get nodes
kubectl get pods -A -o wide --field-selector spec.nodeName=<lease-holder-
node>
```

如果节点运行状况良好，并且租约没有警告事件，请在断定租约卡住之前检查持有者节点上的 Trident 节点 pod 日志：

```
kubectl get pods -n <trident-namespace> -o wide \
  --field-selector spec.nodeName=<lease-holder-node>
kubectl logs -n <trident-namespace> <trident-node-pod> -c trident-main
```

如果底层节点或网络问题可以修复，请先执行此操作。当 Trident 成功完成操作时，租约即可解决。

安全解决卡住的租约

重要说明	当其持有者可能仍在运行 `cryptsetup`操作时，请勿删除租约。来自不同节点且未由租约协

调的并发 `cryptsetup`命令可能会损坏 LUKS 标头。

使用与节点条件匹配的修正：

- 持有者节点出现故障并已关闭电源：识别该节点持有的租约并删除受影响的租约。如果节点在其工作负载重新调度之前恢复，请允许 Trident 完成其操作并释放租约。如果工作负载已经移动，请等待恢复节点上的 Pod 完全终止，确认死锁仍然存在，然后删除过时的租约。
- 持有者节点已失去控制平面连接：节点可能仍在运行工作负载。最好在删除 Lease 之前关闭节点。如果节点仍然可访问，请停止该节点上的 Trident 节点容器，或者从其日志中验证它无法获取或删除 Lease 并且未在运行 `cryptsetup`。Trident 需要访问 Kubernetes API 才能在每次操作前获取 Lease；连接丢失后，它无法启动另一个操作。从创建 Lease 起至少等待五分钟，然后再考虑删除，以允许任何正在进行的命令完成。
- 持有者节点和网络正常： 检查租约警告事件和 Trident 节点日志，以识别失败的操作。如果需要将受影响的工作负载移动到另一个节点，请对持有者节点进行封锁，删除受影响的应用程序 Pod 而不强制删除，等待它们终止，然后取消对节点的封锁。仅在确认原始操作不再运行后才删除租约。

仅在确认可以安全执行此操作后，才删除租约：

```
kubectl delete lease -n <trident-namespace> <pv-name>
```

修复后监控受影响的 Pod 和卷操作。如果再次保留租约，请调查在租约事件中报告的 cryptsetup 错误，并解决根本问题。

Trident Kerberos 传输中加密

通过使用Kerberos传输中加密、您可以对受管集群和存储后端之间的流量启用加密、从而提高数据访问安全性。

Trident支持将ONTAP用作存储后端的Kerberos加密：

- 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。

您可以创建、删除、调整大小、创建快照、克隆、 只读克隆、并导入使用NFS加密的卷。

使用内部**ONTAP**卷配置传输中的**Kerberos**加密

您可以对受管集群与内部ONTAP存储后端之间的存储流量启用Kerberos加密。

备注	只有使用存储驱动程序、才支持对使用内部ONTAP存储后端的NFS流量进行Kerberos加密 `ontap-nas`。

开始之前

- 确保您可以访问该 `tridentctl` 实用程序。
- 确保您对ONTAP存储后端具有管理员访问权限。
- 确保您知道要从ONTAP存储后端共享的一个或多个卷的名称。
- 确保已准备好ONTAP Storage VM以支持NFS卷的Kerberos加密。有关说明、请参见 "在dataLIF上启

用Kerberos" 。

- 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ONTAP导出策略添加规则、或者创建新的导出策略、以便对ONTAP Storage VM根卷以及与上游Kubornetes集群共享的任何ONTAP卷支持Kerberos加密。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

您可以根据卷的需求配置以下三种不同版本的Kerberos加密之一：

- **Kerberos 5**-(身份验证和加密)
- **Kerberos 5i**-(身份验证和加密与身份保护)
- **Kerberos 5p**-(身份验证和加密、具有身份和隐私保护功能)

使用适当的访问权限配置ONTAP导出策略规则。例如、如果集群要挂载混合使用Kerberos 5i和Kerberos 5p加密的NFS卷、请使用以下访问设置：

Type	只读访问	读/写访问	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、请参见以下文档：

- "创建导出策略"
- "向导出策略添加规则"

创建存储后端

您可以创建包含Kerberos加密功能的Trident存储后端配置。

关于此任务

在创建用于配置Kerberos加密的存储后端配置文件时、您可以使用参数指定以下三种不同版本的Kerberos加密之一 `spec.nfsMountOptions`：

- `spec.nfsMountOptions: sec=krb5` (身份验证和加密)
- `spec.nfsMountOptions: sec=krb5i` (身份验证和加密以及身份保护)
- `spec.nfsMountOptions: sec=krb5p` (身份验证和加密以及身份和隐私保护)

请仅指定一个Kerberos级别。如果在参数列表中指定多个Kerberos加密级别、则仅会使用第一个选项。

步骤

1. 在受管集群上、使用以下示例创建存储后端配置文件。将括号<>中的值替换为您环境中的信息：

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. 使用您在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，则后端配置出现问题。您可以运行以下命令来查看日志以确定发生原因：

```
tridentctl logs
```

确定并更正配置文件中的问题后，您可以再次运行 create 命令。

创建存储类。

您可以创建存储类来配置采用Kerberos加密的卷。

关于此任务

创建存储类对象时、可以使用参数指定以下三种不同版本的Kerberos加密之一 `mountOptions`：

- `mountOptions: sec=krb5` (身份验证和加密)
- `mountOptions: sec=krb5i` (身份验证和加密以及身份保护)
- `mountOptions: sec=krb5p` (身份验证和加密以及身份和隐私保护)

请仅指定一个Kerberos级别。如果在参数列表中指定多个Kerberos加密级别、则仅会使用第一个选项。如果您在存储后端配置中指定的加密级别与您在存储类对象中指定的加密级别不同、则存储类对象优先。

步骤

1. 使用以下示例创建StorageClass Kubenetes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
```

您应看到类似于以下内容的输出：

```
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、您现在可以配置卷。有关说明，请参阅 "配置卷"。

为Azure NetApp Files卷配置传输中的Kerberos加密

您可以对受管集群与单个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启用Kerberos加密。

开始之前

- 确保已在受管Red Hat OpenShift集群上启用Trident。
- 确保您可以访问该 `tridentctl` 实用程序。
- 请注意要求并按照中的说明，确保已为Kerberos加密准备好Azure NetApp Files存储后端 "Azure NetApp Files 文档"。
- 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

在创建配置Kerberos加密的存储后端配置文件时、您可以对其进行定义、使其应用于以下两个可能的级别之一：

- 使用字段的*存储后端级别* `spec.kerberos`
- 使用字段的*虚拟池级别* `spec.storage.kerberos`

在虚拟池级别定义配置时、系统会使用存储类中的标签来选择该池。

在任一级别、您都可以指定以下三种不同版本的Kerberos加密之一：

- `kerberos: sec=krb5` (身份验证和加密)
- `kerberos: sec=krb5i` (身份验证和加密以及身份保护)
- `kerberos: sec=krb5p` (身份验证和加密以及身份和隐私保护)

步骤

1. 在受管集群上、根据需要定义存储后端的位置(存储后端级别或虚拟池级别)、使用以下示例之一创建存储后端配置文件。将括号<>中的值替换为您环境中的信息：

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>
```

```yaml
---
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

```yaml
---
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. 使用您在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，则后端配置出现问题。您可以运行以下命令来查看日志以确定发生原因：

```
tridentctl logs
```

确定并更正配置文件中的问题后，您可以再次运行 create 命令。

创建存储类。

您可以创建存储类来配置采用Kerberos加密的卷。

步骤

1. 使用以下示例创建StorageClass Kubenetes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
```

您应看到类似于以下内容的输出：

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

配置卷

创建存储后端和存储类后、您现在可以配置卷。有关说明，请参阅 "配置卷"。

使用Trident Protect 保护应用程序

了解Trident Protect

NetApp Trident Protect 提供高级应用程序数据管理功能，增强了由NetApp ONTAP存储系统和NetApp Trident CSI 存储供应器支持的有状态 Kubernetes 应用程序的功能和可用性。Trident Protect 简化了跨公有云和本地环境的容器化工作负载的管理、保护和迁移。它还通过其 API 和 CLI 提供自动化功能。

您可以通过创建自定义资源 (CR) 或使用Trident Protect CLI 来使用Trident Protect 保护应用程序。

下一步是什么?

您可以先了解Trident Protect 的相关要求，然后再进行安装：

- "Trident保护要求"

安装Trident Protect

Trident保护要求

首先，请验证您的运行环境、应用程序集群、应用程序和许可证是否准备就绪。确保您的环境满足部署和运行Trident Protect 的这些要求。

Trident Protect Kubernetes 集群兼容性

Trident Protect 与各种完全托管和自托管的 Kubernetes 产品兼容，包括：

- Amazon Elelic Kubelnetes Service (EKS)
- Google Kubernetes Engine （ GKEE ）
- Microsoft Azure Kubernetes Service （ AKS ）
- Red Hat OpenShift
- SUSE Harvester 1.7.0（ONTAP iSCSI）
- SUSE试用者
- VMware Tanzu产品组合
- 上游Kubernetes

备注	• Trident Protect备份仅支持Linux计算节点。Windows 计算节点不支持备份操作。 • 确保安装Trident Protect 的集群已配置运行中的快照控制器和相关的 CRD。要安装快照控制器，请参阅 "这些说明"。 • 确保至少存在一个 VolumeSnapshotClass。有关详细信息，请参阅 "VolumeSnapshotClass"。

- 安装 Trident Protect 需要 Helm 4.x 或更高版本。

Trident Protect 存储后端兼容性

Trident Protect 支持以下存储后端：

- 适用于 NetApp ONTAP 的 Amazon FSX
- Cloud Volumes ONTAP
- ONTAP存储阵列
- Google Cloud NetApp卷
- Azure NetApp Files

确保存储后端满足以下要求：

- 确保连接到集群的 NetApp 存储使用 Trident 24.02 或更新版本（建议使用 Trident 24.10）。
- 确保您有一个NetApp ONTAP存储后端。
- 确保已配置用于存储备份的对象存储分段。
- 创建您计划用于应用程序或应用程序数据管理操作的任何应用程序命名空间。Trident Protect 不会为您创建这些命名空间；如果您在自定义资源中指定了不存在的命名空间，则操作将失败。

NAS经济型卷的要求

Trident Protect 支持对 nas-economy 卷进行备份和恢复操作。目前不支持将快照、克隆和SnapMirror复制到 nas-economy 卷。您需要为计划与Trident Protect 一起使用的每个 nas-economy 卷启用快照目录。

备注

某些应用程序与使用Snapshot目录的卷不兼容。对于这些应用程序、您需要通过在ONTAP存储系统上运行以下命令来隐藏Snapshot目录：

```
nfs modify -vserver <svm> -v3-hide-snapshot enabled
```

您可以通过对每个NAS经济型卷运行以下命令来启用Snapshot目录、并将其替换 `<volume-UUID>`为要更改的卷的UUID：

```
tridentctl update volume <volume-UUID> --snapshot-dir=true --pool-level
=true -n trident
```

备注

通过将Trident后端配置选项设置为，您可以默认为 `true`新卷启用快照目录 `snapshotDir`。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在Trident Protect 的不同版本中有所不同，较新的版本通过 Helm chart 参数提供了简化的配置。

备注	在恢复操作期间，任何 `VirtualMachineSnapshots`为虚拟机 (VM) 创建的数据不会被恢复。

▼ **Trident Protect 25.10** 及更新版本

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，您可以使用以下方法禁用此行为： `vm.freeze` Helm Chart 安装过程中的参数。该参数默认启用。

```
helm install ... --set vm.freeze=false ...
```

▼ **Trident Protect 24.10.1** 至 **25.06**

从Trident Protect 24.10.1 开始， Trident Protect 会在数据保护操作期间自动冻结和解冻 KubeVirt 文件系统。您也可以使用以下命令禁用此自动行为：

```
kubectl set env deployment/trident-protect-controller-manager
NEPTUNE_VM_FREEZE=false -n trident-protect
```

▼ **Trident Protect 24.10**

Trident Protect 24.10 在数据保护操作期间不会自动确保 KubeVirt VM 文件系统的一致性状态。如果您想使用Trident Protect 24.10 保护您的 KubeVirt VM 数据，则需要在执行数据保护操作之前手动启用文件系统的冻结/解冻功能。这样可以确保文件系统处于一致状态。

您可以通过 "正在配置虚拟化" 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
```

要了解如何使用 Trident Protect 定义应用程序和还原虚拟机，请参见 "使用 Trident Protect 保护 KubeVirt 虚拟机"。

SnapMirror复制的要求

NetApp SnapMirror复制功能可与Trident Protect 配合使用，适用于以下ONTAP解决方案：

- 本地 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
- 适用于 NetApp ONTAP 的 Amazon FSX

SnapMirror复制的ONTAP集群要求

如果您计划使用SnapMirror复制、请确保ONTAP集群满足以下要求：

- * NetApp Trident *：使用ONTAP作为后端服务的源 Kubernetes 集群和目标 Kubernetes 集群上都必须存在NetApp Trident 。Trident Protect 支持使用NetApp SnapMirror技术进行复制，该技术使用以下驱动程序支

持的存储类：

- `ontap-nas`：NFS
- `ontap-san`：iSCSI
- `ontap-san`：足球俱乐部
- `ontap-san`：NVMe/TCP（要求最低 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、所有许可证均以NetApp许可证文件(NLG)的形式提供、NLG是一个支持多项功能的文件。有关详细信息、请参见 "ONTAP One附带的许可证" 。

备注	仅支持 SnapMirror 异步保护。

SnapMirror复制的对等注意事项

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**：ONTAP存储后端必须建立对等状态。有关详细信息、请参见 "集群和 SVM 对等概述" 。

备注	确保两个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 必须可供目标集群上的 NetApp Trident 使用。
- 托管后端：您需要在Trident Protect 中添加和管理ONTAP存储后端，以创建复制关系。

用于SnapMirror复制的Trident / ONTAP配置

Trident Protect 要求您至少配置一个支持源集群和目标集群复制的存储后端。如果源集群和目标集群相同，为了获得最佳弹性，目标应用程序应该使用与源应用程序不同的存储后端。

SnapMirror复制的 Kubernetes 集群要求

确保您的 Kubernetes 集群满足以下要求：

- **AppVault** 可访问性：源集群和目标集群都必须具有网络访问权限，才能从 AppVault 读取和写入应用程序对象复制。
- 网络连接：配置防火墙规则、存储桶权限和 IP 允许列表，以实现跨 WAN 的集群和 AppVault 之间的通信。

备注	许多企业环境在 WAN 连接中实施严格的防火墙策略。在配置复制之前，请与您的基础设施团队验证这些网络要求。

安装并配置Trident Protect

如果您的环境满足Trident Protect 的要求，您可以按照以下步骤在集群上安装Trident Protect。您可以从NetApp获取Trident Protect，或者从您自己的私有注册表中安装它。如果您的集群无法访问互联网，从私有注册表安装会很有帮助。

安装Trident Protect

从NetApp安装Trident Protect

步骤

1. 添加Trident Helm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。代替 `<name-of-cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect
--set clusterName=<name-of-cluster> --version 100.2610.0 --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
100.2610.0 --create-namespace --namespace trident-protect
```

调试日志记录有助于NetApp支持人员排除故障，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，您可以从私有镜像仓库安装Trident Protect。在这些示例中，请将括号中的值替换为您环境中的信息：

步骤

1. 将以下映像提取到本地计算机、更新标记、然后将其推送到您的私人注册表：

```
docker.io/netapp/controller:26.10.1
docker.io/netapp/trident-protect-utils:v6.0.0
docker.io/netapp/trident-protect-guestfs:v2.0.0
docker.io/netapp/kopiablockrestore:26.10.1
docker.io/netapp/trident-autosupport:26.10.1
docker.io/netapp/exechook:26.10.1
docker.io/netapp/resourcebackup:26.10.1
docker.io/netapp/resourcerestore:26.10.1
docker.io/netapp/resourcedelete:26.10.1
docker.io/netapp/filebrowse-transfer:26.10.1
```

例如：

```
docker pull docker.io/netapp/controller:26.10.1
```

```
docker tag docker.io/netapp/controller:26.10.1 <private-registry-url>/controller:26.10.1
```

```
docker push <private-registry-url>/controller:26.10.1
```

> 备注
>
> 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 100.2610.0 --repo https://netapp.github.io/trident-protect-helm-chart` 下载 Helm 图表，然后将生成的 `trident-protect-100.2610.0.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-100.2610.0.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. 创建用于私人注册表身份验证的拉机密：

```
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. 创建一个名为的文件 `protectValues.yaml`。请确保其中包含以下Trident Protect 设置：

```
---
imageRegistry: <private-registry-url>
```

```
imagePullSecrets:
  - name: regcred
```

备注 这 imageRegistry`和 `imagePullSecrets`这些值适用于所有组件图像，包括 `resourcebackup`和 `resourcerestore。如果您将镜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。这将确保所有图像都从此处提取。 <private-registry-url>/<image-name>:<tag>。

7. 使用 Helm 安装Trident Protect。代替 `<name_of_cluster>`集群名称将分配给集群，并用于标识集群的备份和快照：

```
helm install trident-protect netapp-trident-protect/trident-protect --set clusterName=<name_of_cluster> --version 100.2610.0 --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 100.2610.0 --create-namespace --namespace trident-protect -f protectValues.yaml
```

调试日志记录有助于NetApp支持人员排除故障，而无需更改日志级别或重现问题。

备注 有关其他 Helm Chart 配置选项，包括AutoSupport设置和命名空间过滤，请参阅 "定制Trident Protect 安装"。

安装Trident Protect CLI 插件

您可以使用Trident Protect 命令行插件，它是Trident的一个扩展。 `tridentctl`用于创建和与Trident Protect 自定义资源 (CR) 交互的实用程序。

安装Trident Protect CLI 插件

在使用命令行实用程序之前、您需要将其安装在用于访问集群的计算机上。根据您的计算机使用的是x64 CPU还是ARM CPU、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/26.10.1/tridentctl-protect-linux-amd64
```

下载适用于**Linux ARM64 CPU**的插件

步骤

1. 下载Trident Protect CLI 插件：

```
curl -L -o tridentctl-protect https://github.com/NetApp/tridentctl-protect/releases/download/26.10.1/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/26.10.1/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/26.10.1/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路径变量中定义的位置。例如、 `/usr/bin`或 `/usr/local/bin`(您可能需要提升Privileges)：

```
cp ./tridentctl-protect /usr/local/bin/
```

3. 您也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、建议确保此位置属于您的路径变量：

```
cp ./tridentctl-protect ~/bin/
```

备注	通过将插件复制到路径变量中的某个位置、您可以通过在任意位置键入或 `tridentctl protect`来使用此插件 `tridentctl-protect。

查看**Trident**命令行界面插件帮助

您可以使用内置插件的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，您可以为某些命令启用自动补全功能。

为**bash shell**启用自动完成

步骤

1. 创建完成脚本：

```
tridentctl-protect completion bash > tridentctl-completion.bash
```

2. 在主目录中创建一个新目录以包含该脚本：

```
mkdir -p ~/.bash/completions
```

3. 将下载的脚本移动到 `~/.bash/completions`目录：

```
mv tridentctl-completion.bash ~/.bash/completions/
```

4. 将以下行添加到 `~/.bashrc`主目录中的文件：

```
source ~/.bash/completions/tridentctl-completion.bash
```

为**Z shell**启用自动完成

步骤

1. 创建完成脚本：

```
tridentctl-protect completion zsh > tridentctl-completion.zsh
```

2. 在主目录中创建一个新目录以包含该脚本：

```
mkdir -p ~/.zsh/completions
```

3. 将下载的脚本移动到 `~/.zsh/completions`目录：

```
mv tridentctl-completion.zsh ~/.zsh/completions/
```

4. 将以下行添加到 `~/.zprofile`主目录中的文件：

```
source ~/.zsh/completions/tridentctl-completion.zsh
```

结果

下次shell登录时、您可以使用带有trdentcd-protect插件的命令自动完成。

定制Trident Protect 安装

您可以自定义Trident Protect 的默认配置，以满足您环境的特定要求。

指定Trident Protect 容器资源限制

安装Trident Protect 后，您可以使用配置文件来指定Trident Protect 容器的资源限制。设置资源限制可以控制Trident Protect 操作消耗集群资源的程度。

步骤

1. 创建一个名为的文件 `resourceLimits.yaml`。
2. 根据您的环境需求，在文件中填充Trident Protect 容器的资源限制选项。

   以下示例配置文件显示了可用设置、并包含每个资源限制的默认值：

```
---
jobResources:
  defaults:
    limits:
      cpu: 8000m
      memory: 10000Mi
      ephemeralStorage: ""
    requests:
      cpu: 100m
      memory: 100Mi
      ephemeralStorage: ""
  resticVolumeBackup:
```

```
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  resticVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeBackup:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
  kopiaVolumeRestore:
    limits:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
    requests:
      cpu: ""
      memory: ""
      ephemeralStorage: ""
```

3. 应用文件中的值 `resourceLimits.yaml`：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f resourceLimits.yaml --reuse-values
```

自定义安全上下文约束

安装Trident Protect 后，您可以使用配置文件来修改Trident Protect 容器的 OpenShift 安全上下文约束 (SCC)。这些约束定义了 Red Hat OpenShift 集群中 pod 的安全限制。

步骤

1. 创建一个名为的文件 `sccconfig.yaml`。
2. 将scc选项添加到文件中、然后根据环境的需要修改参数。

   以下示例显示了SCC选项参数的默认值：

   ```
   scc:
     create: true
     name: trident-protect-job
     priority: 1
   ```

   下表介绍了SCC选项的参数：

参数	Description	Default
创建	确定是否可以创建SCC资源。只有在将设置为 `true`且Helm安装过程标识OpenShift环境时、才会创建SCC资源 `scc.create`。如果不在OpenShift上运行，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保护作业
优先级	定义SCC的优先级。优先级值较高的SCC会在优先级值较低的SCC之前进行评估。	1

3. 应用文件中的值 `sccconfig.yaml`：

   ```
   helm upgrade trident-protect -n trident-protect netapp-trident-
   protect/trident-protect -f sccconfig.yaml --reuse-values
   ```

   此操作会将默认值替换为文件中指定的值 `sccconfig.yaml`。

配置其他**Trident Protect** 舵图设置

您可以自定义 AutoSupport 设置、命名空间筛选和 OpenShift webhook 证书管理，以满足您的特定要求。下表描述了可用的配置参数：

参数	Type	Description
自动支持代理	string	为NetApp AutoSupport连接配置代理 URL。使用此功能通过代理服务器路由支持包上传。例子：`http://my.proxy.url` 。

参数	Type	Description
自动支持.不安全	boolean	设置为时跳过AutoSupport代理连接的 TLS 验证 `true`。仅用于不安全的代理连接。 （默认： `false` ）
自动支持已启用	boolean	启用或禁用每日Trident Protect AutoSupport捆绑包上传。设置为 `false`每日定时上传功能已禁用，但您仍然可以手动生成支持包。 （默认： `true` ）
恢复跳过命名空间注释	string	要从备份和恢复操作中排除的命名空间注释的逗号分隔列表。允许您根据注释过滤命名空间。
restoreSkipNamespaceLabels	string	要从备份和恢复操作中排除的命名空间标签的逗号分隔列表。允许您根据标签过滤命名空间。
useOpenshiftServiceCA	boolean	启用与 OpenShift 的本地 Service CA Operator 的集成，以便为 Trident Protect 准入 webhook 自动创建和注入 TLS 证书。设置为 `true`时，`Service CA Operator` 会自动管理 `webhook TLS` 证书。仅在 `OpenShift` 集群上使用。（默认值： `false`）

要加密快照和备份元数据，请参阅 "加密 Trident Protect 快照和备份的 AppVault 元数据"。

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. 创建配置文件并命名 `values.yaml`。
2. 在您创建的文件中，添加您想要自定义的配置选项。

```
autoSupport:
  enabled: false
  proxy: http://my.proxy.url
  insecure: true
restoreSkipNamespaceAnnotations: "annotation1,annotation2"
restoreSkipNamespaceLabels: "label1,label2"
useOpenshiftServiceCA: true
```

3. 填充后 `values.yaml`具有正确值的文件，应用配置文件：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
```

```
protect/trident-protect -f values.yaml --reuse-values
```

使用 **CLI** 标志

步骤

1. 使用以下命令 `--set`标志来指定单个参数：

```
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}" \
  --set useOpenshiftServiceCA=true \
  --reuse-values
```

将**Trident Protect Pod** 限制在特定节点上

您可以使用 Kubernetes nodeSelector 节点选择约束，根据节点标签来控制哪些节点有资格运行Trident Protect pod。默认情况下， Trident Protect 仅限于运行 Linux 的节点。您可以根据需要进一步自定义这些限制条件。

步骤

1. 创建一个名为的文件 `nodeSelectorConfig.yaml`。
2. 将nodeSelector选项添加到文件中、然后修改文件以添加或更改节点标签、从而根据环境需求进行限制。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. 应用文件中的值 `nodeSelectorConfig.yaml`：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f nodeSelectorConfig.yaml --reuse-values
```

此操作会将默认限制替换为您在文件中指定的限制 `nodeSelectorConfig.yaml`。

管理Trident Protect

管理Trident Protect 授权和访问控制

Trident Protect 使用 Kubernetes 的基于角色的访问控制 (RBAC) 模型。默认情况下，Trident Protect 提供一个系统命名空间及其关联的默认服务帐户。如果您的组织拥有众多用户或特定的安全需求，则可以使用Trident Protect 的 RBAC 功能来更精细地控制对资源和命名空间的访问。

集群管理员始终可以访问默认命名空间中的资源 `trident-protect`、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、您需要创建更多的名分并将资源和应用程序添加到这些名分。

请注意、任何用户都不能在默认命名空间中创建应用程序数据管理CRS `trident-protect`。您需要在应用程序命名空间中创建应用程序数据管理CRS (最佳做法是、在与其关联的应用程序相同的命名空间中创建应用程序数据管理CRS)。

备注

只有管理员才能访问具有特权的Trident Protect 自定义资源对象，其中包括：

- **AppVault**：需要存储分段凭据数据
- **AutoSupportBundle**：收集指标、日志和其他敏感的Trident Protect数据
- *AutoSupportBundleSchedule：管理日志收集计划

作为最佳实践、请使用RBAC限制管理员对有权限的对象的访问。

有关RBAC如何控制对资源和称表的访问的详细信息，请参阅 "Kubbernetes RBAC文档"。

有关服务帐户的信息，请参见 "Kubbernetes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。集群管理员应完成以下任务、以创建一个环境、在此环境中、工程组和营销组各自只能访问分配给各自命名区域的资源。

第1步：创建一个命名空间以包含每个组的资源

通过创建命名空间、您可以从逻辑上分离资源、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
```

第2步：创建新的服务帐户、以便与每个命名空间中的资源进行交互

您创建的每个新命名空间都会附带一个默认服务帐户、但您应为每组用户创建一个服务帐户、以便将来根据需要在各个组之间进一步划分Privileges。

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
```

第3步：为每个新服务帐户创建一个密钥

服务帐户密钥用于向服务帐户进行身份验证、如果泄露、可以轻松删除和重新创建。

步骤

1. 为工程服务帐户创建一个密钥：

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: eng-user
  name: eng-user-secret
  namespace: engineering-ns
type: kubernetes.io/service-account-token
```

2. 为营销服务帐户创建密钥：

```
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
```

第4步：创建RoleBinding对象以将ClusterRole对象绑定到每个新服务帐户

安装Trident Protect 时会创建一个默认的 ClusterRole 对象。您可以通过创建和应用 RoleBinding 对象将此 ClusterRole 绑定到服务帐户。

步骤

1. 将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：授予对AppVault对象的访问权限

要执行备份和快照等数据管理任务、集群管理员需要向各个用户授予对AppVault对象的访问权限。

步骤

1. 创建并应用AppVault和机密组合YAML文件、以授予用户对AppVault的访问权限。例如、以下CR将授予用户对AppVault的访问权限 `eng-user`：

```
apiVersion: v1
data:
  accessKeyID: <ID_value>
  secretAccessKey: <key_value>
kind: Secret
metadata:
  name: appvault-for-eng-user-only-secret
  namespace: trident-protect
type: Opaque
---
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: appvault-for-eng-user-only
  namespace: trident-protect # Trident Protect system namespace
spec:
  providerConfig:
    azure:
      accountName: ""
      bucketName: ""
      endpoint: ""
    gcp:
```

```
      bucketName: ""
      projectID: ""
    s3:
      bucketName: testbucket
      endpoint: 192.168.0.1:30000
      secure: "false"
      skipCertValidation: "true"
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: appvault-for-eng-user-only-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: appvault-for-eng-user-only-secret
  providerType: GenericS3
```

2. 创建并应用角色CR、使集群管理员能够授予对命名空间中特定资源的访问权限。例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eng-user-appvault-reader
  namespace: trident-protect
rules:
- apiGroups:
  - protect.trident.netapp.io
  resourceNames:
  - appvault-for-enguser-only
  resources:
  - appvaults
  verbs:
  - get
```

3. 创建并应用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. 尝试检索所有名称库的AppVault对象信息：

   ```
   kubectl get appvaults -n trident-protect
   --as=system:serviceaccount:engineering-ns:eng-user
   ```

   您应看到类似于以下内容的输出：

   ```
   Error from server (Forbidden): appvaults.protect.trident.netapp.io is
   forbidden: User "system:serviceaccount:engineering-ns:eng-user"
   cannot list resource "appvaults" in API group
   "protect.trident.netapp.io" in the namespace "trident-protect"
   ```

   b. 测试用户是否可以获取他们现在有权访问的AppVault信息：

   ```
   kubectl auth can-i --as=system:serviceaccount:engineering-ns:eng-user
   get appvaults.protect.trident.netapp.io/appvault-for-eng-user-only -n
   trident-protect
   ```

   您应看到类似于以下内容的输出：

```
yes
```

结果

您为其授予了AppVault权限的用户应该能够使用授权的AppVault对象执行应用程序数据管理操作、并且不能访问分配的命名区之外的任何资源、也不能创建他们无权访问的新资源。

监控Trident保护资源

您可以使用 kube-state-metrics、Prometheus 和 Alertmanager 开源工具来监控Trident Protect 保护的资源的健康状况。

kube-state-metrics 服务从 Kubernetes API 通信生成指标。将其与Trident Protect 结合使用，可以显示有关环境中资源状态的有用信息。

Prometheus 是一个工具包，它可以接收 kube-state-metrics 生成的数据，并将其呈现为关于这些对象的易于阅读的信息。kube-state-metrics 和 Prometheus 共同提供了一种方法，让您可以监控使用Trident Protect 管理的资源的健康状况和状态。

警报管理器是一项服务、可接收Prometheus等工具发送的警报、并将其路由到您配置的目标。

备注

这些步骤中包含的配置和指导仅为示例；您需要对其进行自定义以符合您的环境。有关具体说明和支持、请参见以下官方文档：

- "Kube-state-metrics 文档"
- "Prometheus文档"
- "记录"

第1步：安装监控工具

要在Trident Protect 中启用资源监控，您需要安装和配置 kube-state-metrics、Promethus 和 Alertmanager。

安装Kube-state-metrics

您可以使用Helm安装Kube-state-metrics。

步骤

1. 添加Kube-state-metrics Helm图表。例如：

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

2. 将 Prometheus ServiceMonitor CRD 应用到集群：

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/main/example/prometheus-operator-
crd/monitoring.coreos.com_servicemonitors.yaml
```

3. 为Helm图表创建配置文件(例如 `metrics-config.yaml`)。您可以根据您的环境自定义以下示例配置：

metrics-config.yaml：Kube-state-metrics Helm图表配置

```
---
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图表来安装Kube-state-metrics。例如：

```
helm install custom-resource -f metrics-config.yaml prometheus-
community/kube-state-metrics --version 5.21.0
```

5. 按照以下说明配置 kube-state-metrics，以生成Trident Protect 使用的自定义资源的指标： "Kube-state-metrics自定义资源文档" 。

安装 Prometheus

您可以按照中的说明安装Prometheus "Prometheus文档"。

安装活动管理器

您可以按照中的说明安装提示管理器 "记录"。

第2步：配置监控工具以协同工作

安装监控工具后、您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以将警报路由到警报管理器。编辑Prometheus配置文件(`prometheus.yaml`)并添加以下部分：

   prometheus.yaml：向 Alertmanager 发送警报

   ```
   alerting:
     alertmanagers:
       - static_configs:
           - targets:
             - alertmanager.trident-protect.svc:9093
   ```

结果

Prometheus现在可以从Kube-state-metrics收集指标、并可向警报管理器发送警报。现在、您可以配置触发警报的条件以及警报的发送位置。

第3步：配置警报和警报目标

将这些工具配置为协同工作后、您需要配置触发警报的信息类型以及警报的发送位置。

警报示例：备份失败

以下示例定义了备份自定义资源的状态设置为5秒或更长时间时触发的严重警报 `Error`。您可以自定义此示例以匹配您的环境、并将此YAML段包含在您的配置文件中 `prometheus.yaml`：

rules.yaml：定义失败备份的 **Prometheus** 警报

```
rules.yaml: |
  groups:
    - name: fail-backup
        rules:
          - alert: BackupFailed
            expr: kube_customresource_backup_info{status="Error"}
            for: 5s
            labels:
              severity: critical
            annotations:
              summary: "Backup failed"
              description: "A backup has failed."
```

将警报管理器配置为向其他通道发送警报

通过在文件中指定相应的配置、您可以将警报管理器配置为向其他通道发送通知、例如电子邮件、PagerDty、Microsoft团队或其他通知服务 `alertmanager.yaml`。

以下示例将配置警报管理器、以便向SI延 时信道发送通知。要根据您的环境自定义此示例、请将此密钥的值替换 `api_url`为您的环境中使用的Slackwebhook URL：

alertmanager.yaml：向 **Slack** 频道发送警报

```
data:
  alertmanager.yaml: |
    global:
      resolve_timeout: 5m
    route:
      receiver: 'slack-notifications'
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - api_url: '<your-slack-webhook-url>'
            channel: '#failed-backups-channel'
            send_resolved: false
```

生成**Trident Protect** 支持包

Trident Protect 使管理员能够生成包含对NetApp支持有用的信息的捆绑包，包括有关受管理集群和应用程序的日志、指标和拓扑信息。如果您已连接到互联网，则可以使用自定义资源 (CR) 文件将支持包上传到NetApp支持站点 (NSS)。

使用**CR**创建支持包

步骤

1. 创建自定义资源(CR)文件并将其命名(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；请为您的环境选择一个唯一且合理的名称。
   - **spec.t**触发 器**Type**：(*required*)用于确定是立即生成支持包、还是按计划生成支持包。计划在UTC时间中午12点生成捆绑包。可能值：
     - 已计划
     - 手动
   - **spec.u**倍 加载已启用：(可 选)控制是否应在生成支持包后将其上传到NetApp支持站点。如果未指定，则默认为 `false`。可能值：
     - true
     - false (默认)
   - **spec.dataWindowStart**：(可 选) RFC 3339格式的日期字符串，指定支持包中包含的数据窗口应开始的日期和时间。如果未指定、则默认为24小时前。您可以指定的最早窗口日期是7天前。

     YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AutoSupportBundle
metadata:
  name: trident-protect-support-bundle
spec:
  triggerType: Manual
  uploadEnabled: true
  dataWindowStart: 2024-05-05T12:30:00Z
```

3. 填充后 `trident-protect-support-bundle.yaml`具有正确值的文件，应用 CR：

```
kubectl apply -f trident-protect-support-bundle.yaml -n trident-
protect
```

使用命令行界面创建支持包

步骤

1. 创建支持包、将括号中的值替换为您环境中的信息。 `trigger-type`确定是否立即创建分发包，或者是否由计划决定了创建时间，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
```

监视和检索支持包

使用任一方法创建支持包后，您可以监视其生成进度并将其检索到本地系统。

步骤

1. 等待 `status.generationState`到达 `Completed`状态。您可以使用以下命令监控生成进度：

```
kubectl get autosupportbundle trident-protect-support-bundle -n trident-
protect
```

2. 将支持包检索到您的本地系统。从已完成的AutoSupport包中获取复制命令：

```
kubectl describe autosupportbundle trident-protect-support-bundle -n
trident-protect
```

   找到 `kubectl cp`从输出中执行命令并运行它，用您喜欢的本地目录替换目标参数。

升级Trident保护

您可以将Trident Protect 升级到最新版本，以享受新功能或修复错误。

备注

- 从版本 24.10 升级时，升级期间运行的快照可能会失败。此失败不会阻止将来创建快照（无论是手动快照还是计划快照）。如果升级期间快照失败，您可以手动创建新快照以确保应用程序受到保护。

  为避免潜在的故障，您可以在升级前禁用所有快照计划，然后在升级后重新启用。但是，这会导致升级期间丢失所有计划的快照。

- 对于私有镜像仓库安装，请确保目标版本所需的 Helm Chart 和镜像在您的私有镜像仓库中可用，并验证您的自定义 Helm 值与新 Chart 版本兼容。更多信息，请参阅"从私有注册表安装Trident Protect"。

第1步：选择版本

Trident Protect 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 24.10（或任何 24.10 dot 版本）升级到 25.10。
- 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 24.10 并希望升级到 26.10：
  a. 首次从 24.10 升级到 25.06。
  b. 然后从 25.06 升级到 26.10。

步骤 2：升级 **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：

备注	如果您是从 25.06 之前的版本升级，则需要执行此步骤，因为 CRD 现在已包含在Trident Protect Helm 图表中。

   a. 运行此命令将 CRD 的管理从 `trident-protect-crds`到 `trident-protect`：

```
kubectl get crd | grep protect.trident.netapp.io | awk '{print $1}' |
xargs -I {} kubectl patch crd {} --type merge -p '{"metadata":
{"annotations":{"meta.helm.sh/release-name": "trident-protect"}}}'
```

   b. 运行此命令删除 `trident-protect-crds`图表：

备注	不要卸载 `trident-protect-crds`图表使用 Helm，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保护：

```
helm upgrade trident-protect netapp-trident-protect/trident-protect
```

```
--version 100.2610.0 --namespace trident-protect
```

备注

您可以通过添加以下内容来配置升级期间的日志级别。 `--set logLevel=debug`升级命令。默认日志级别为 `warn`。建议启用调试日志记录进行故障排除，因为它可以帮助NetApp支持人员诊断问题，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** 对象来管理存储桶。

Trident Protect 的存储桶自定义资源 (CR) 被称为 AppVault。AppVault 对象是存储桶的声明性 Kubernetes 工作流表示。AppVault CR 包含存储桶在保护操作（例如备份、快照、恢复操作和SnapMirror复制）中使用的必要配置。只有管理员才能创建应用保险库。

在应用程序上执行数据保护操作时，您需要手动或从命令行创建 AppVault CR。AppVaultCR 特定于您的环境，您可以使用本页上的示例作为创建 AppVault CR 的指南。

备注

确保 AppVault CR 位于安装了Trident Protect 的集群上。如果 AppVault CR 不存在或您无法访问，命令行会显示错误。

配置**AppVault**身份验证和密码

在创建 AppVault CR 之前，请确保您选择的 AppVault 和数据移动器可以向提供商和任何相关资源进行身份验证。

数据移动工具存储库密码

当您使用 CR 或Trident Protect CLI 插件创建 AppVault 对象时，您可以为 Restic 和 Kopia 加密指定带有自定义密码的 Kubernetes 密钥。如果您不指定密钥， Trident Protect 将使用默认密码。

- 手动创建 AppVault CR 时，使用 **spec.dataMoverPasswordSecretRef** 字段指定密钥。
- 使用Trident Protect CLI 创建 AppVault 对象时，请使用 `--data-mover-password-secret-ref`用于指定密钥的参数。

备注

要加密快照和备份元数据或更改保管库密码，请使用 `spec.vaultPasswordSecretRef` 字段。请参阅 "加密 Trident Protect 快照和备份的 AppVault 元数据"。

创建数据移动工具存储库密码密钥

请参考以下示例创建密码密钥。创建 AppVault 对象时，您可以指示Trident Protect 使用此密钥向数据移动器存储库进行身份验证。

备注

- 根据所使用的数据移动工具、您只需包含该数据移动工具的相应密码即可。例如、如果您正在使用Restic、并且将来不打算使用KONIA、则在创建密钥时只能包含Restic密码。
- 请将密码保存在安全的地方。您将需要它来还原同一集群或其他集群上的数据。如果集群或`trident-protect`命名空间被删除后，没有密码您将无法恢复备份或快照。

使用CR

```
---
apiVersion: v1
data:
  KOPIA_PASSWORD: <base64-encoded-password>
  RESTIC_PASSWORD: <base64-encoded-password>
kind: Secret
metadata:
  name: my-optional-data-mover-secret
  namespace: trident-protect
type: Opaque
```

使用命令行界面

```
kubectl create secret generic my-optional-data-mover-secret \
--from-literal=KOPIA_PASSWORD=<plain-text-password> \
--from-literal=RESTIC_PASSWORD=<plain-text-password> \
-n trident-protect
```

S3 兼容存储 IAM 权限

当您访问与 S3 兼容的存储（例如 Amazon S3、通用 S3）时， "StorageGRID S3" ， 或者 "ONTAP S3"使用Trident Protect 时，您需要确保提供的用户凭据具有访问存储桶的必要权限。以下是授予使用Trident Protect进行访问所需的最低权限的策略示例。您可以将此策略应用于管理 S3 兼容存储桶策略的用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

有关 Amazon S3 策略的更多信息，请参阅 "Amazon S3 文档" 。

用于 **Amazon S3**（**AWS**）身份验证的 **EKS Pod Identity**

Trident Protect 支持 Kopia 数据移动器操作的 EKS Pod Identity。此功能可实现对 S3 存储桶的安全访问，而无需将 AWS 凭证存储在 Kubernetes 机密中。

EKS Pod Identity 与**Trident Protect** 的要求

在将 EKS Pod Identity 与Trident Protect 结合使用之前，请确保以下事项：

- 您的 EKS 集群已启用 Pod Identity。
- 您已创建具有必要的 S3 存储桶权限的 IAM 角色。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 角色与服务账户关联的详细说明，请参阅 "AWS EKS Pod Identity 文档"。

AppVault 配置 使用 EKS Pod Identity 时，请使用以下配置配置您的 AppVault CR `useIAM: true`标记而不是明确的凭证：

```
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
```

云提供商的**AppVault**密钥生成示例

定义 AppVault CR 时，您需要包含凭证以访问提供商托管的资源，除非您使用 IAM 身份验证。如何生成凭证密钥将根据提供商的不同而有所不同。以下是几个提供商的命令行密钥生成示例。您可以使用以下示例为每个云提供商的凭证创建密钥。

Google Cloud

```
kubectl create secret generic <secret-name> \
--from-file=credentials=<mycreds-file.json> \
```

```
-n trident-protect
```

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

以下是每个提供程序的AppVault定义示例。

AppVault CR示例

您可以使用以下CR示例为每个云提供程序创建AppVault对象。

备注

- 您可以选择指定一个包含用于Restic和Koria存储库加密的自定义密码的Kubernetes密钥。有关详细信息、请参见 [数据移动工具存储库密码] 。
- 对于Amazon S3 (AWS) AppVault对象、您可以选择指定sSession令牌、如果使用单点登录(SSO)进行身份验证、则此令牌非常有用。此令牌是在中为提供程序生成密钥时创建的云提供商的AppVault密钥生成示例。
- 对于S3 AppVault对象、您可以选择使用密钥为出站S3流量指定出口代理URL `spec.providerConfig.S3.proxyURL`。

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
```

备注	对于使用 Pod Identity 和 Kopia 数据移动器的 EKS 环境，您可以删除 `providerCredentials` 部分并添加 `useIAM: true`根据 `s3`配置。

Microsoft Azure

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: azure-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: Azure
  providerConfig:
    azure:
      accountName: account-name
      bucketName: trident-protect-src-bucket
  providerCredentials:
    accountKey:
      valueFromSecret:
        key: accountKey
        name: azure-trident-protect-src-bucket-secret
```

通用 **S3**

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: generic-s3-trident-protect-src-bucket
```

```
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: GenericS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

ONTAP S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: ontap-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: OntapS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

StorageGRID S3

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: storagegrid-s3-trident-protect-src-bucket
  namespace: trident-protect
spec:
  dataMoverPasswordSecretRef: my-optional-data-mover-secret
  providerType: StorageGridS3
  providerConfig:
    s3:
      bucketName: trident-protect-src-bucket
      endpoint: s3.example.com
      proxyURL: http://10.1.1.1:3128
  providerCredentials:
    accessKeyID:
      valueFromSecret:
        key: accessKeyID
        name: s3-secret
    secretAccessKey:
      valueFromSecret:
        key: secretAccessKey
        name: s3-secret
```

使用**Trident Protect CLI** 创建 **AppVault** 的示例

您可以使用以下命令行界面命令示例为每个提供程序创建AppVault CRS。

备注

- 您可以选择指定一个包含用于Restic和Koria存储库加密的自定义密码的Kubernetes密钥。有关详细信息、请参见 [数据移动工具存储库密码] 。
- 对于S3 AppVault对象、您可以选择使用参数为出站S3流量指定出口代理URL `--proxy-url <ip_address:port>`。

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 `providerConfig.s3`配置选项

请参阅下表了解 S3 提供程序配置选项：

参数	Description	Default	示例
`providerConfig.s3.skipCertValidation`	禁用 SSL/TLS 证书验证。	false	“真”，“假”
`providerConfig.s3.secure`	启用与 S3 端点的安全 HTTPS 通信。	true	“真”，“假”
`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`	提供用于 SSL/TLS 验证的自定义根 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 标识。	false	true false

查看AppVault信息

您可以使用Trident Protect CLI 插件查看有关您在集群上创建的 AppVault 对象的信息。

步骤

1. 查看AppVault对象的内容：

```
tridentctl-protect get appvaultcontent gcp-vault \
--show-resources all \
-n trident-protect
```

示例输出：

```
+-------------+-------+----------+-----------------------------
+---------------------------+
|   CLUSTER   |  APP  |   TYPE   |            NAME             |
TIMESTAMP         |
+-------------+-------+----------+-----------------------------
+---------------------------+
|             | mysql | snapshot | mysnap                      | 2024-
08-09 21:02:11 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815180300 | 2024-
08-15 18:03:06 (UTC) |
```

```
| production1 | mysql | snapshot | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:06 (UTC) |
| production1 | mysql | snapshot | hourly-e7db6-20240815200300 | 2024-
08-15 20:03:06 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815180300 | 2024-
08-15 18:04:25 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815190300 | 2024-
08-15 19:03:30 (UTC) |
| production1 | mysql | backup   | hourly-e7db6-20240815200300 | 2024-
08-15 20:04:21 (UTC) |
| production1 | mysql | backup   | mybackup5                   | 2024-
08-09 22:25:13 (UTC) |
|             | mysql | backup   | mybackup                    | 2024-
08-09 21:02:52 (UTC) |
+-------------+-------+----------+-----------------------------
+---------------------------+
```

2. (可选)要查看每个资源的AppVaultPath，请使用标志 `--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` 。

删除**AppVault**

您可以随时删除AppVault对象。

备注	在删除AppVault对象之前、请勿 `finalizers`删除AppVault CR中的密钥。如果这样做、可能会导致AppVault存储分段中有残留数据、集群中会出现孤立资源。

开始之前

确保已删除要删除的AppVault正在使用的所有快照和备份CRS。

使用**Kubbernetes**命令行界面删除**AppVault**

1. 删除AppVault对象、替换 `appvault-name`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、替换 `appvault-name`为要删除的AppVault对象的名称：

```
tridentctl-protect delete appvault <appvault-name> \
-n trident-protect
```

加密 **Trident Protect** 快照和备份的 **AppVault** 元数据

您可以加密 Trident Protect 存储在 AppVault 中的快照和备份元数据，包括 Kubernetes Secrets 中的敏感信息。您还可以更改保护加密数据的密码。一次密码更改将重新加密元数据加密密钥，并将 AppVault 中的每个 Kopia 存储库和每个 Restic 存储库移动到新密码。

默认情况下禁用加密。启用后，Trident Protect 会在将元数据存储到 AppVault 之前加密新的快照和备份元数据。这将保护您的应用程序资源，包括 Secrets 中的敏感信息，除非您从应用程序快照或备份中明确排除这些资源。

保管库密码可以保护加密数据的密钥。请注意以下行为：

- 更改保管库密码时，Trident Protect 将使用新密码保护相同的密钥。它不会重写已加密的数据。
- 启用加密之前存储的数据仍然可读。
- 即使您稍后关闭新数据的加密，已加密的数据仍然可读。

关于此任务

启用加密时，除非您指定自定义密码，否则 Trident Protect 将使用不安全的默认密码。为每个 AppVault 创建一个自定义保管库密码，并在 `spec.vaultPasswordSecretRef`中引用它。

如果使用默认密码启用了加密，并在稍后添加自定义密码，则必须轮换密码，以便将现有数据和存储库移至自定义密码。请参阅 [轮换保管库密码]。

为每个 AppVault 使用唯一的自定义密码。每次为一个 AppVault 轮换密码，并在开始另一次密码更改之前确认轮换已完成。

开始之前

请确保您具备以下条件：

- 访问 `trident-protect`命名空间。
- 创建 Kubernetes Secrets 和修补 AppVault 自定义资源（CR）的权限。
- 为每个 AppVault 计划一个单独的密码 Secret。

启用元数据加密

您可以通过为 Trident Protect 设置 Helm 值来启用元数据加密。

步骤

1. 使用 `values.yaml`文件将 `resourceBackupEncryptionEnabled`设置为 `true`：
   a. 将以下内容添加到您的 `values.yaml` 文件中：

      ```
      resourceBackupEncryptionEnabled: true
      ```

   b. 应用此文件：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect -f values.yaml --reuse-values
```

2. 或者，使用 Helm `--set` 标志将 `resourceBackupEncryptionEnabled`设置为 `true`：

```
helm upgrade trident-protect -n trident-protect netapp-trident-
protect/trident-protect \
  --set resourceBackupEncryptionEnabled=true \
  --reuse-values
```

结果

Trident Protect 加密新的快照和备份元数据。

下一步是什么?

为每个 AppVault 创建自定义保管库密码。在此之前，Trident Protect 使用不安全的默认密码。

创建自定义保管库密码

要使用自己的密码，请创建一个包含该密码的 Kubernetes Secret，然后在 AppVault 中引用该 Secret。

关于此任务

如果在创建 AppVault 时配置自定义密码，则在存储任何数据之前，AppVault 会立即使用自定义密码。您无需轮换密码。

如果 AppVault 已经具有 Kopia 或 Restic 存储库，则添加新密码不会更改这些存储库的密码。请勿使用本节中的步骤。相反，请按照 [轮换保管库密码] 中的步骤操作。

该 `VAULT_PASSWORD`值必须满足以下要求：

- 不能为空
- 必须小于 64 KiB
- 不得包含 NUL 字符
- 不得包含换行符
- 不得以空格或其他空白字符开头或结尾

密码可以包含其他字符，包括非 ASCII 字符。

步骤

1. 在与 AppVault 相同的命名空间中创建一个 Secret。该 Secret 必须包含一个名为 `VAULT_PASSWORD`的键：

```
kubectl -n trident-protect create secret generic vault-password \
  --from-literal=VAULT_PASSWORD='<password>'
```

备注	如果 AppVault 报告密码无效，请根据上述要求检查该 `VAULT_PASSWORD`值。常见原因是空值、尾随换行或密码开头或结尾有空格。

2. 在 AppVault CR 的 `spec.vaultPasswordSecretRef`字段中引用 Secret：

```
apiVersion: protect.trident.netapp.io/v1
kind: AppVault
metadata:
  name: my-vault
  namespace: trident-protect
spec:
  vaultPasswordSecretRef: vault-password
  providerType: <provider-type>
  providerConfig:
    <provider-configuration>
```

如果 `spec.vaultPasswordSecretRef`为空，AppVault 将使用不安全的默认密码。

结果

AppVault 使用您的自定义密码来保护元数据加密密钥。

轮换保管库密码

您可以轮换已存储数据的 AppVault 的密码。一次轮换会重新加密元数据加密密钥，并将 AppVault 中的每个 Kopia 存储库和每个 Restic 存储库移动到新密码。

关于此任务

在轮换过程中，Trident Protect 会执行以下操作：

- 等待 AppVault 正在运行的备份和还原作业完成后再启动。
- 使用新密码保护元数据加密密钥 (`metadata-key.json`）。它不会重写归档对象。
- 查找 AppVault 前缀下的每个 Kopia 存储库和每个 Restic 存储库，并将其密码更改为新密码。已使用新密码打开的存储库不会更改。
- 跳过备份或还原作业仍在使用的存储库，稍后重试。
- 暂停引用 AppVault 的已启用计划。对于每个计划，Trident Protect 将 `spec.enabled`设置为 `false`，并添加以下注释：

```
protect.trident.netapp.io/vault-password-rotation-suspended:
<namespace>/<appvault-name>
```

轮换完成后，Trident Protect 将再次仅启用具有此注释的计划。如果轮换失败，它将从停止的位置恢复。已使用新密码的存储库不会再次更改。

开始之前

请确保您具备以下条件：

- 符合保管库密码要求的新密码。
- 没有其他 AppVault 正在轮换其密码。如果多个 AppVaults 共享相同的存储桶和前缀，则其中只有一个可以轮换密码。请参阅 管理共享存储桶的 AppVaults 的轮换。

步骤

1. 使用与当前 Secret 名称不同的名称创建新 Secret：

```
kubectl -n trident-protect create secret generic vault-password-new \
  --from-literal=VAULT_PASSWORD='<new-password>'
```

重要说明	在轮换完成之前，请勿编辑或删除当前的 Secret。在每个存储库使用新密码之前，Trident Protect 需要当前密码才能打开现有数据。如果过早删除当前 Secret，请使用相同的名称和原始 `VAULT_PASSWORD`值重新创建它。

2. 启用轮换并将 AppVault 设置为使用新密钥。将 `my-vault`替换为您 AppVault 的名称：

```
kubectl -n trident-protect patch appvault my-vault --type merge --patch
'
spec:
  enableRotation: true
  vaultPasswordSecretRef: vault-password-new
'
```

备注	您必须先将 `spec.enableRotation`设置为 `true`，然后才能更改密码。如果是 `false`，AppVault 将报告轮换已禁用，并且不会更改密码。

3. 按照 [监控密码轮换] 中的步骤监控轮换。
4. 轮换完成后，如果没有其他内容使用它，请删除上一个 Secret。

结果

元数据加密密钥以及 AppVault 中的每个 Kopia 和 Restic 存储库都受到新密码的保护，并且重新启用了 Trident Protect 暂停的计划。

监控密码轮换

您可以检查密码轮换的进度并确认其已完成。

步骤

1. 查看 AppVault：

```
kubectl -n trident-protect get appvault my-vault --output yaml
```

2. 在输出中，检查 `status.vaultPasswordRotation`部分。正在轮换时，如下所示：

```
status:
  vaultPasswordRotation:
    state: Rotating
    rotated: 3
    pending: 1
    message: <message>
```

下表描述了这些字段：

字段	Description
status.vaultPasswordRotation.state	当前旋转状态。
status.vaultPasswordRotation.rotated	现在使用新密码的存储库数。
status.vaultPasswordRotation.pending	由于备份或还原正在使用而跳过的存储库数。
status.vaultPasswordRotation.message	最新进度或失败消息。

结果

当满足以下所有条件时，轮换完成：

- `status.vaultPasswordRotation` 不存在。
- `status.previousRepositoryPasswordSecretRef` 为空。
- `status.repositoryPasswordSecretRef` 匹配 `spec.vaultPasswordSecretRef`。

下一步是什么？

如果轮换未完成，请检查以下内容：

- 轮换未开始： `spec.enableRotation`可能是 `false`。将其设置为 `true`，然后将 `spec.vaultPasswordSecretRef`设置为新的 Secret。
- 存储库保持待处理：备份或还原作业仍在使用它们。等待作业完成。Trident Protect 会自动重试待处理的存储库。
- 轮换未开始，因为另一个 **AppVault** 拥有锁：请参阅 管理共享存储桶的 AppVaults 的轮换。
- 消息字段报告失败：解决消息描述的问题。轮换从停止的位置恢复。

迁移自 dataMoverPasswordSecretRef

如果现有 AppVault 使用较旧的 `spec.dataMoverPasswordSecretRef`字段，则可以将其移动到 `spec.vaultPasswordSecretRef`。

关于此任务

Trident Protect 保留 `spec.dataMoverPasswordSecretRef` 以实现向后兼容。对新的 AppVaults 使用 `spec.vaultPasswordSecretRef`。

步骤

1. 将 `spec.dataMoverPasswordSecretRef`及其 Secret 保留在原位。
2. 使用 `spec.vaultPasswordSecretRef`将密码轮换为新的 Secret。按照[轮换保管库密码]中的步骤进行操作。
3. 轮换完成后，从 AppVault 中移除 `spec.dataMoverPasswordSecretRef`。
4. 如果没有其他对象使用它，请删除旧密钥。

管理共享存储桶的 **AppVaults** 的轮换

如果多个 AppVaults 使用相同的存储桶和前缀，则其中只有一个可以轮换密码。

关于此任务

密码轮换使用名为 `password-rotation.lock`的锁定文件。该锁定记录了允许轮换存储桶和前缀密码的 `AppVault` 的 `UID`。如果第二个 `AppVault` 具有 `spec.enableRotation: true`，并且该锁定指定了不同的 AppVault，则第二个 AppVault 无法轮换。

示例 **1.** 步骤

将轮换转移到另一个 **AppVault**

1. 在拥有该锁的 AppVault 上，清除 `spec.enableRotation`：

```
kubectl -n trident-protect patch appvault <current-owner> --type
merge --patch '
spec:
  enableRotation: false
'
```

2. 等待锁被移除。
3. 在下一个应轮换的 AppVault 上，将 `spec.enableRotation`设置为 `true`：

```
kubectl -n trident-protect patch appvault <new-owner> --type merge
--patch '
spec:
  enableRotation: true
'
```

移除已删除的 **AppVault** 遗留的锁定

如果拥有锁的 AppVault 不复存在且锁未被移除，例如在崩溃后，请执行这些步骤。

1. 从存储桶中的 AppVault 前缀中删除 `password-rotation.lock` 文件。
2. 在应轮换的 AppVault 上，将 `spec.enableRotation`设置为 `true`：

```
kubectl -n trident-protect patch appvault <appvault-name> --type
merge --patch '
spec:
```

```
    enableRotation: true
'
```

相关信息

- "使用Trident Protect AppVault 对象来管理存储桶。"
- "定制Trident Protect 安装"
- "保护应用程序"
- "使用Trident Protect 恢复应用程序"

使用Trident Protect 定义管理应用程序

您可以通过创建应用程序 CR 和关联的 AppVault CR 来定义要使用Trident Protect 管理的应用程序。

创建AppVault CR

您需要创建一个 AppVault CR，该 CR 将在对应用程序执行数据保护操作时使用，并且 AppVault CR 需要位于安装了Trident Protect 的集群上。AppVault CR 是针对您的特定环境的；有关 AppVault CR 的示例，请参阅："AppVault自定义资源。"

定义应用程序

您需要定义要使用Trident Protect 管理的每个应用程序。您可以通过手动创建应用程序 CR 或使用Trident Protect CLI 来定义要管理的应用程序。

备注	要定义通过名称而不是整个命名空间来保护特定 KubeVirt 虚拟机 (VM) 的 VM 范围应用程序，请使用应用程序 CR 中的 `spec.includedVirtualMachines`字段或 CLI 中的 `--virtual-machines`标志。有关详细信息、字段参考和 YAML 示例，请参阅 "使用 Trident Protect 保护 KubeVirt 虚拟机"。

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**:(*required_*)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属于此列表。标签选择器是可选的、可用于筛选每个指定命名空间中的资源。
      - 。**spec.includedClusterScopedResources**:(可 选)使用此属性指定要包含在应用程序定义中的集群范围资源。此属性允许您根据资源的组、版本、种类和标签来选择这些资源。

- **groupVersionKind**：(*required*)指定集群范围资源的API组、版本和类型。
- **labelSelecter**：(可 选)根据集群范围的资源的标签对其进行筛选。

- **spec.volumeGroups**: (可选) 使用此属性可将相关卷一起创建快照，作为崩溃一致的卷组快照。有关详细信息，请参阅 "将卷组快照与 Trident Protect 结合使用"。
- **metadata.annotations.protect.trident.netapp.io/skip-vm-freeze**: (可选) 此注解仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。指定此应用程序在快照期间是否可以写入文件系统。如果设置为 true，应用程序将忽略全局设置，并且可以在快照期间写入文件系统。如果设置为 false，应用程序将忽略全局设置，并且在快照期间文件系统将被冻结。如果指定了注解，但应用程序定义中没有虚拟机，则忽略该注解。如未特别说明，则申请流程如下："全球Trident Protect 冷冻设置" 。

如果您需要在创建应用程序后应用此标注、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"
```

+
YAML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*）如果需要，可以将资源筛选添加到同一 CR 以包括或排除特定资源：

   ◦ 通用过滤器示例：

   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：

     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
     - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
     - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
     - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
     - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
     - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
     - *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。例如：`"trident.netapp.io/os=linux"`。

备注	当两者 `resourceFilter`和 `labelSelector`被使用， `resourceFilter`首先运行，然后 `labelSelector`应用于结果资源。

   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"]
```

- 仅 **PVC** 过滤器示例：

  要定义仅限 PVC 的应用程序，还必须在资源筛选器中包含 `PersistentVolume`和 `VolumeSnapshotClass`。快照和备份操作依赖于 `PersistentVolume`（绑定到每个 PVC 的群集范围卷）和 `VolumeSnapshotClass`（快照驱动程序），如果没有它们，则会失败。例如：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: my-pvc-app
  namespace: my-app-namespace
spec:
  includedNamespaces:
  - namespace: my-app-namespace
  resourceFilter:
    resourceMatchers:
    - kind: PersistentVolumeClaim
      version: v1
    - kind: PersistentVolume
      version: v1
    - kind: VolumeSnapshotClass
      version: v1
    resourceSelectionCriteria: Include
```

2. 创建应用程序CR以匹配您的环境后、请应用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、将括号中的值替换为您环境中的信息。您可以使用逗号分隔列表和示例中显示的参数在应用程序定义中包括名称和资源。

创建应用时，您可以选择使用注解来指定应用在快照期间是否可以写入文件系统。这仅适用于从虚拟机定义的应用程序，例如 KubeVirt 环境，其中文件系统冻结发生在快照之前。如果您将注释设置为 `true`该应用程序忽略全局设置，可以在快照期间写入文件系统。如果你把它设置为 `false`该应用程序忽略全局设置，导致文件系统在快照期间冻结。如果使用了注解，但应用程序定义中没有虚拟机，则该注解将被忽略。如果您不使用注解，应用程序将遵循以下规则："全球Trident Protect 冷冻设置" 。

+ 使用 CLI 创建应用程序时，可以使用 `--annotation`标志指定注释。

+ * 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">
```

+ * 可以使用 `--resource-filter-include`和 `--resource-filter-exclude`标志根据 `resourceSelectionCriteria`（例如组、种类、版本、标签、名称和命名空间）来包含或排除资源，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"]}]'
```

+ *要定义仅限 PVC 的应用程序，您还必须在资源筛选器中包含 `PersistentVolume`和 `VolumeSnapshotClass`。快照和备份操作依赖于 `PersistentVolume`（绑定到每个 PVC 的集群范围卷）和 `VolumeSnapshotClass`（快照驱动程序），如果没有它们，则会失败。例如：

+

```
tridentctl-protect create app my-pvc-app --namespaces <my-app-
namespace> --resource-filter-include
'[{"Kind":"PersistentVolumeClaim","Version":"v1"},{"Kind":"PersistentVo
lume","Version":"v1"},{"Kind":"VolumeSnapshotClass","Version":"v1"}]'
-n <my-app-namespace>
```

使用Trident Protect 保护应用程序

您可以使用自动保护策略或临时保护策略，通过拍摄快照和备份来保护Trident Protect 管理的所有应用程序。

备注

- 您可以配置Trident Protect 在数据保护操作期间冻结和解冻文件系统。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。
- 要在同一时间点对应用程序的相关卷进行快照，请在应用程序中定义卷组。请参阅 "将卷组快照与 Trident Protect 结合使用"。

创建按需快照

您可以随时创建按需快照。

备注

如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：要创建快照的应用程序的Kubernetes名称。
   - **spec.appVaultRef**：(*required*)应存储快照内容(元数据)的AppVault的名称。
   - **spec.relaimPolicy**：(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，快照也 `Retain`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. 使用正确的值填充文件后 `trident-protect-snapshot-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-snapshot-cr.yaml
```

使用命令行界面创建快照

步骤

1. 创建快照、将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create snapshot <my_snapshot_name> --appvault
<my_appvault_name> --app <name_of_app_to_snapshot> -n
<application_namespace>
```

创建按需备份

您可以随时备份应用程序。

备注	如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、快照或克隆中。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的S3备份操作。如果令牌在备份操作期间过期、则操作可能会失败。

- 有关检查当前会话令牌到期时间的详细信息、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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要备份的应用程序的Kubernetes名称。
   - **spec.appVaultRef**：(*required*)应存储备份内容的AppVault的名称。
   - **spec.dataMover**：(可 选)一个字符串，指示用于备份操作的备份工具。可能值(区分大小写)：
     - `Restic`
     - `Kopia` (默认)
   - **spic.relaimPolicy**：(可 选)定义了从备份申请中释放备份时会发生什么情况。可能值：
     - `Delete`
     - `Retain` (默认)
   - **spec.snapshotRef**：（可选）：用作备份源的快照的名称。如果不提供此参数、则会创建和备份临时快照。

     YAML示例：

```
---
```

```
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. 使用正确的值填充文件后 `trident-protect-backup-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-backup-cr.yaml
```

使用命令行界面创建备份

步骤

1. 创建备份、将括号中的值替换为您环境中的信息。例如：

```
tridentctl-protect create backup <my_backup_name> --appvault <my-
vault-name> --app <name_of_app_to_back_up> --data-mover
<Kopia_or_Restic> -n <application_namespace>
```

您可以选择使用 `--full-backup`标志来指定备份是否应为非增量备份。默认情况下、所有备份均为增量备份。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、以最大程度地降低与还原相关的风险。

支持的备份注释

下表描述了创建备份 CR 时可以使用的注释：

标注	Type	Description	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` 创建非增量备份。最佳实践是定期执行完整备份，然后在两次完整备份之间执行增量备份，以最大限度地降低与恢复相关的风险。	false
protect.trident.netapp.io/snapshot-completion-timeout	string	完成整个快照操作允许的最长时间。	60米
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	卷快照达到可用状态所需的最长时间。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	string	创建卷快照允许的最长时间。	5米

标注	Type	Description	默认值
protect.trident.netapp.io/pvc-bind-timeout-sec	string	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位） `Bound`操作失败前的阶段。	1200（20分钟）

创建数据保护计划

保护策略通过按照定义的计划创建快照、备份或两者来保护应用程序。您可以选择每小时、每天、每周和每月创建快照和备份，并可以指定要保留的副本数量。您可以使用 full-backup-rule 注释来安排非增量式完整备份。默认情况下，所有备份都是增量的。定期执行完整备份以及其间的增量备份有助于降低与恢复相关的风险。

备注

- 您可以通过设置 `backupRetention`归零， `snapshotRetention`为大于零的值。环境 `snapshotRetention`为零意味着任何计划的备份仍将创建快照，但这些快照是临时的，并在备份完成后立即被删除。
- 如果在应用程序定义中明确引用了集群范围的资源、或者这些资源引用了任何应用程序命名源、则会将这些资源包括在备份、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；请为您的环境选择一个唯一且合理的名称。
   - **spec.dataMover**：(可 选)一个字符串，指示用于备份操作的备份工具。可能值(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**：(*required*)应存储备份内容的AppVault的名称。
   - **spec.backupRetention**: (必需) 要保留的备份数量。零表示不应创建备份（仅快照）。
   - **spec.backupReclaimPolicy**: (可选) 确定如果备份 CR 在其保留期内被删除，则备份会发生什么情况。保留期过后，备份文件总是会被删除。可能的值（区分大小写）：
     - `Retain` (默认)
     - `Delete`
   - **spec.snapshotRetention**: (必需) 要保留的快照数量。零表示不创建任何快照。
   - **spec.snapshotReclaimPolicy**: (可选) 确定如果快照 CR 在其保留期内被删除，则快照会发生什么情况。保留期过后，快照总是会被删除。可能的值（区分大小写）：
     - `Retain`
     - `Delete`（默认）
   - 。**spec.granularity**:计划的运行频率。可能值以及必需的关联字段：
     - `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**：（可选）计划应运行的月份日期（1 - 31）。如果粒度设置为 `Monthly`。该值必须以字符串形式提供。
- **spec.dayOfWeek**：（可选）计划应运行的星期几（0 - 7）。值 0 或 7 表示星期日。如果粒度设置为 `Weekly`。该值必须以字符串形式提供。
- **spec.hour**：（可选）计划应运行的小时数（0 - 23）。如果粒度设置为 `Daily`, `Weekly`, 或者 `Monthly`。该值必须以字符串形式提供。
- **spec.minute**：（可选）计划应运行的小时中的分钟数（0 - 59）。如果粒度设置为 `Hourly`, `Daily`, `Weekly`, 或者 `Monthly`。该值必须以字符串形式提供。
- **spec.runImmediately**：（可选）设置为 `true` 以在创建计划时触发一次性的即时基线运行（根据保留设置进行备份和/或快照）。默认为 `false`。这不会修改后续重复周期。

备份和快照计划的示例 YAML：

```yaml
---
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"
```

仅快照计划的示例 YAML：

```yaml
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-snapshot-schedule
spec:
```

```
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "0"
  snapshotRetention: "15"
  granularity: Daily
  hour: "2"
  minute: "0"
```

立即运行的计划的示例 YAML：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  namespace: my-app-namespace
  name: my-daily-schedule-run-immediately
spec:
  applicationRef: my-application
  appVaultRef: appvault-name
  backupRetention: "7"
  snapshotRetention: "7"
  granularity: Daily
  hour: "3"
  minute: "0"
  runImmediately: true
```

3. 使用正确的值填充文件后 `trident-protect-schedule-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-schedule-cr.yaml
```

使用命令行界面创建计划

步骤

1. 创建保护计划、将括号中的值替换为您环境中的信息。例如：

> 备注 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>
```

以下选项可让您对日程安排进行更多控制：

- 完整备份计划：使用 `--full-backup-rule` 标记以安排非增量式完整备份。此标志仅适用于 `--granularity Daily`。可能的值：
  - `Always`每天创建完整备份。
  - 具体工作日：指定一个或多个日期，用逗号分隔（例如， `"Monday,Thursday"`）。有效值：星期一、星期二、星期三、星期四、星期五、星期六、星期日。

备注	这 `--full-backup-rule` 该标志位不适用于按小时、按周或按月划分的粒度。

- 立即基线保护：使用 `--run-immediately true` 在创建计划时立即创建初始备份或快照，而不是等待第一个计划的运行时间。默认值为 `false`。
- 仅快照计划：设置 `--backup-retention 0` 并指定一个大于零的值 `--snapshot-retention`。

支持的日程注释

下表描述了创建计划变更请求 (CR) 时可以使用的注释：

标注	Type	Description	默认值
protect.trident.netapp.io/full-backup-rule	string	指定安排完整备份的规则。你可以将其设置为 `Always` 您可以根据需要进行持续完整备份或自定义备份。例如，如果您选择按日粒度进行备份，则可以指定应进行完整备份的星期几（例如， `"Monday,Thursday"`）。有效的工作日值为：星期一、星期二、星期三、星期四、星期五、星期六、星期日。请注意，此注释只能用于已包含以下内容的日程表： `granularity` 设置为 `Daily`。	未设置（所有备份均为增量备份）
protect.trident.netapp.io/snapshot-completion-timeout	string	完成整个快照操作允许的最长时间。	60米

标注	Type	Description	默认值
protect.trident.netapp.io/volume-snapshots-ready-to-use-timeout	string	卷快照达到可用状态所需的最长时间。	30米
protect.trident.netapp.io/volume-snapshots-created-timeout	string	创建卷快照允许的最长时间。	5米
protect.trident.netapp.io/pvc-bind-timeout-sec	string	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位） `Bound`操作失败前的阶段。	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。

备注	确保回收策略设置为 `Delete`从对象存储中删除所有备份数据。该策略的默认设置是 `Retain`以避免意外数据丢失。如果政策没有改变 `Delete`，备份数据将保留在对象存储中，需要手动删除。

步骤

1. 删除与备份关联的备份CR：

```
kubectl delete backup <backup_name> -n my-app-namespace
```

检查备份操作的状态

您可以使用命令行检查正在进行、已完成或失败的备份操作的状态。

步骤

1. 使用以下命令检索备份操作的状态、将括号中的值替换为环境中的信息：

```
kubectl get backup -n <namespace_name> <my_backup_cr_name> -o jsonpath
='{.status}'
```

为azure-ANF-files (NetApp)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 存储类且在Trident 24.06 之前创建的存储后端启用节省空间的备份和还原功能。此功能适用于 NFSv4 卷，并且不会占用容量池中的额外空间。

开始之前

确保满足以下要求：

- 您已安装Trident Protect。
- 您已在Trident Protect中定义了一个应用程序。在您完成此步骤之前，此应用程序的保护功能将受到限制。
- 您已 `azure-netapp-files` 选择作为存储后端的默认存储类。

▼ 展开以了解配置步骤

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-pv-files且与应用程序关联的NetApp启用Snapshot目录：

   ```
   tridentctl update volume <pv name> --snapshot-dir=true -n trident
   ```

   b. 确认已为每个关联PV启用Snapshot目录：

   ```
   tridentctl get volume <pv name> -n trident -o yaml | grep
   snapshotDir
   ```

   响应：

   ```
   snapshotDirectory: "true"
   ```

   +
   如果未启用快照目录， Trident Protect 将选择常规备份功能，该功能会在备份过程中暂时占用容量池中的空间。在这种情况下，请确保容量池中有足够的空间来创建与被备份卷大小相同的临时卷。

结果

该应用程序已准备好使用Trident Protect 进行备份和恢复。每个 PVC 也可供其他应用程序用于备份和恢复。

将卷组快照与 Trident Protect 结合使用

使用卷组快照在同一时间点捕获相关的持久卷声明（PVC）。这对于写入多个卷的应用程序非常有用，包括具有多个磁盘的虚拟机（VM）。

关于此任务

您可以在 `spec.volumeGroups`自定义资源 （CR） 的 `Application`字段中定义卷组。创建应用程序快照时，Trident Protect 使用 ONTAP 一致性组在同一时刻对卷组中的所有 PVC 执行快照。它为卷组中的每个命名空间创建一个一致性组。组中的每个卷也会获得其自己的快照，因此您可以单独还原卷。

Trident Protect 创建不在卷组中的 PVC 的单独快照。这些快照是逐个创建的，因此无法保证它们反映的是同一时间点。

卷组快照适用于按需快照、计划快照以及 Trident Protect 作为备份的一部分创建的快照。如果备份现有快照，Trident Protect 将使用该快照，不会创建新的组快照。复制也不会创建其他组快照。它使用组快照已创建的每卷快照。

Trident Protect 仅在满足以下所有条件时才会在同一时间点捕获卷：

- 它们位于同一个卷组中。
- 它们位于同一个命名空间中。
- 它们位于同一存储虚拟机（SVM）上。
- 此卷组在该命名空间中的卷数不超过 80 个。

如果 Trident Protect 无法创建组快照（例如，由于不满足要求），它会向 `status.warnings`添加警告，并为该组的 PVC 创建单独的快照。这些快照是逐个创建的，因此它们可能无法反映相同的时间点。应用程序快照仍会完成。

下表介绍了在常见情况下会发生的情况。

情形	结果
卷组在多个命名空间中选择 PVC	组快照属于一个命名空间，因此 Trident Protect 会在每个命名空间中创建一个单独的组快照。这些组快照并非在同一时刻创建。一个可能成功，而另一个仍在运行、失败或回退到单个快照。例如，在 `payments-a`和 `payments-b`中标记为 `app: payments-db`的 PVC 将成为两个单独的组快照。请勿将来自不同命名空间的组快照视为同一个还原点。
命名空间中的 PVC 位于多个 SVM 上	Trident Protect 不会按 SVM 或 Trident 后端拆分卷。ONTAP 一致性组不能包含来自多个 SVM 的卷，因此整个命名空间（包括共享一个 SVM 的卷）的组快照将失败。Trident Protect 会添加一条警告，删除组快照，并为每个卷创建单独的快照。要将卷一起捕获，请将它们放在同一个 SVM 上。
一个命名空间包含 80 多个匹配的 PVC	Trident Protect 仅跳过该命名空间并创建其 PVC 的单个快照。同一个卷组中的其他命名空间仍会获取组快照。
PVC 与多个 `volumeGroups`条目匹配	Trident Protect 将 PVC 添加到两个卷组，其中一个组快照可能会失败。失败的组快照将回退到单个快照。确保每个 PVC 仅匹配一个条目。
卷组包括使用不支持卷组快照的驱动程序或协议的 PVC	组快照失败，Trident Protect 会为这些 PVC 创建单独的快照。

开始之前

- PVC 必须使用支持卷组快照的存储驱动程序和协议。有关支持的驱动程序和协议，请参阅 "使用 Trident 卷组快照"。
- 集群必须运行 Kubernetes 1.36 或更高版本。
- Kubernetes 快照控制器必须为 8.5.0 或更高版本，并启用 `CSIVolumeGroupSnapshot`功能门控。
- 必须安装 `v1`的 `CRD`，适用于 `VolumeGroupSnapshot`、 `VolumeGroupSnapshotClass`和 `VolumeGroupSnapshotContent`。
- 您必须将 Trident 26.10 或更高版本与在线后端配合使用。 `CSIVolumeGroupSnapshot` 功能门控也必须在

Trident csi-snapshotter 容器上启用。

- 必须定义一个使用 csi.trident.netapp.io`驱动程序的 A `VolumeGroupSnapshotClass。如果只定义了一个类，Trident Protect 将使用该类。如果定义了多个类，请使用 snapshot.storage.kubernetes.io/is-default-class: "true"`注释 Trident Protect 应使用的类。否则，Trident Protect 将按字母顺序使用第一个类。您不能在 `Application`或 `Snapshot CR 中选择类。有关更多信息，请参阅"卷组快照类"。
- 必须允许 Trident 后端中的 ONTAP 帐户创建、获取和删除一致性组，并对其进行快照。请参阅 "管理应用程序一致性组" 和 "管理应用程序一致性组快照"。默认 SVM 管理员角色 (vsadmin）不包括这些权限。

定义卷组

Trident Protect 仅为属于某个卷组的卷创建组快照。将 spec.volumeGroups`添加到 `Application CR 以定义组。每个条目都是一个单独的组。按标签、VM 或两者选择卷。

关于此任务

- 您只能在 Application CR 中定义卷组。 tridentctl-protect CLI 没有卷组选项。
- Trident Protect 仅包含卷组中绑定的 PVC，并创建未绑定的 PVC 的单个快照。如果条目未选择绑定的 PVC ，Trident Protect 不会为该条目创建组快照。
- 请确保每个 PVC 仅匹配一个条目。

开始之前

定义要保护的应用程序。请参阅 "使用Trident Protect 定义管理应用程序"。

步骤

1. 如果按标签选择卷，请标记要分组的 PVC。您可以使用任何标签键和值。此页面上的示例使用 volume-group=my-volume-group。

```
kubectl label pvc <pvc_name_1> <pvc_name_2> volume-group=my-volume-group
-n <application_namespace>
```

2. 将 spec.volumeGroups`添加到您现有的 `Application CR 文件中。每个条目必须包含 labelSelector、 `vmSelector`或两者。如果一个条目包含两个字段，则卷组包含任一字段选择的每个卷。

   - **spec.volumeGroups[].labelSelector**：在应用程序中选择 PVC 的 Kubernetes 标签选择器 (matchLabels`或 `matchExpressions）。空选择器无效。
   - **spec.volumeGroups[].vmSelector**：KubeVirt 虚拟机列表，每个虚拟机都带有 namespace`和 `name。Trident Protect 包含每个虚拟机挂载的 PVC。

   以下示例定义了两个卷组。第一组包括带有 `volume-group: my-volume-group`标签的 PVC。第二组包括 `my-vm-1`和 `my-vm-2`挂载的 PVC。

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
```

```
  name: my-app-name
  namespace: my-namespace
spec:
  includedNamespaces:
    - namespace: my-namespace
  volumeGroups:
    - labelSelector:
        matchLabels:
          volume-group: my-volume-group
    - vmSelector:
        - namespace: my-namespace
          name: my-vm-1
        - namespace: my-namespace
          name: my-vm-2
```

+

备注	保留您的 `Application` CR 已包含的其他字段，例如 `spec.resourceFilter`。如果从应用的文件中省略某个字段， `kubectl apply` 会将其从 CR 中删除。

3. 应用更新的 CR：

```
kubectl apply -f <application_cr_file>.yaml
```

结果

`Application` CR 包含卷组。应用程序的下一个快照将使用它们。

创建快照

为应用程序创建按需快照、按需备份或数据保护计划。您无需添加任何字段。Trident Protect 使用您在 `Application` CR 中定义的卷组。有关说明，请参阅"使用Trident Protect 保护应用程序"。

备注	默认情况下，Trident Protect 会在冻结 VM 文件系统之前创建组快照。卷组中的卷在未冻结的情况下被捕获，因此不包括 VM 尚未写入磁盘的数据。不在卷组中的磁盘将在稍后捕获，即 Trident Protect 冻结 VM 后创建单个快照时。要同时捕获 VM 的所有磁盘，请将每个磁盘都包含在卷组中。有关冻结 VM 文件系统的详细信息，请参阅 "使用KubeVirt VM保护数据"。

查看结果

快照完成后，检查其状态以确认 Trident Protect 创建了组快照。

备注	已完成的应用程序快照并不意味着其组快照成功。即使每个组快照都被跳过或失败，`VolumeGroupSnapshotsCreated` 条件也会变为 `true`，并且 Trident Protect 不会创建 Kubernetes 事件或指标。要检测此情况，请监控 `status.warnings` 以及 `skipped` 和

failed 字段。

步骤

1. 获取快照的名称。要列出应用程序的快照，请运行以下命令：

```
kubectl get snapshots -n <application_namespace>
```

2. 检查快照的状态：

```
kubectl get snapshots <snapshot_name> -n <application_namespace> -o yaml
```

结果

该 status.volumeGroupSnapshots 字段对每个组快照都有一个条目。Trident Protect 为卷组中的每个命名空间创建单独的组快照。以下示例已缩略：

```
status:
  volumeGroupSnapshots:
    - name: vgs-<snapshot_uid_prefix>-<group_index>-<namespace>
      namespace: <namespace>
      ready: true
      memberPVCs:
        - pvcName: <pvc_name_1>
          pvcNamespace: <namespace>
        - pvcName: <pvc_name_2>
          pvcNamespace: <namespace>
```

组快照名称包括 Snapshot UID 的前八个字符、 `spec.volumeGroups`条目的索引（从 `0`开始）和命名空间，截断为 12 个字符。

- ready: true 表示组快照成功。组成该组的快照也列在 `status.volumeSnapshots`下方。这些是组的成员快照，而不是相同卷的其他快照。
- skipped: true 表示 Trident Protect 未尝试创建组快照，因为该卷组在该命名空间中的 PVC 超过 80 个。Trident Protect 会改为创建该组卷的各个快照，并将 skipReason 文本添加到 status.warnings。
- failed: true 表示 Trident Protect 尝试创建组快照，但失败了。Trident Protect 会改为创建该组各个卷的快照，并将 error 文本添加到 status.warnings 中。

临时 **PVC** 标签和清理

Trident Protect 会为组快照中的每个 PVC 添加一个标签，例如 protect.trident.netapp.io/vgs-group-673c0189-0=true。您自己的标签不会更改。该标签将保留在 PVC 上，直到您删除应用程序快照。当您删除应用程序快照时，Trident Protect 还会删除其卷组快照和成员快照，并从 PVC 中删除标签。无论快照的 reclaimPolicy`是 `Delete`还是 `Retain，都会发生这种情况。

下一步是什么？

以与任何其他快照相同的方式从快照还原应用程序。卷组快照没有单独的还原。Trident Protect 从成员快照还原，因此您可以还原单个 PVC 或 VM，而无需还原组中的每个 PVC。请参阅"从快照还原到其他命名空间"或"从快照还原到原始命名空间"。

相关信息

- "使用Trident Protect 定义管理应用程序"
- "使用Trident Protect 保护应用程序"
- "使用 Trident Protect 保护 KubeVirt 虚拟机"

还原应用程序

使用**Trident Protect** 恢复应用程序

您可以使用Trident Protect 从快照或备份中恢复您的应用程序。将应用程序恢复到同一集群时，从现有快照恢复速度会更快。

备注

- 还原应用程序时、为该应用程序配置的所有执行挂钩都会随该应用程序还原。如果存在还原后执行挂钩、则它会在还原操作中自动运行。
- 对于 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，对于 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来自定义恢复操作。欲了解更多信息，请参阅 "使用高级Trident Protect 恢复设置"。
- 要从存档还原特定 KubeVirt 虚拟机 (VM) 而不还原所有命名空间资源，请使用还原 CR 上的 `includedVirtualMachines`字段。有关支持的 CR、字段引用和 YAML 示例，请参阅 "使用 Trident Protect 保护 KubeVirt 虚拟机"。

从备份还原到其他命名空间

当您使用 BackupRestore CR 将备份还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或者制定保护计划。

备注

- 将备份还原到具有现有资源的其他命名空间不会更改与备份中的资源共享名称的任何资源。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、或者将备份还原到新命名空间。
- 使用 CR 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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"。

备注

当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制

Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

使用**CR**

步骤

1. 创建自定义资源(CR)文件并将其命名为 `trident-protect-backup-restore-cr.yaml`。
2. 在创建的文件中、配置以下属性：
   - 。**metadata.name**:(*required*_)此自定义资源的名称；请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**：(*required*)存储备份内容的AppVault的名称。
   - **spec.destinationApplicationName**：（可选）已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
   - 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: BackupRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appArchivePath: my-backup-path
       appVaultRef: appvault-name
       destinationApplicationName: my-new-app-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：

   备注 Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹

配。

- **resourceMatcher[].group**：(可 选)要筛选的资源的组。
- **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
- **resourceMatcher[].version**：(可 选)要筛选的资源版本。
- **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
- **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
- *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。例如：`"trident.netapp.io/os=linux"`。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 使用正确的值填充文件后 `trident-protect-backup-restore-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用命令行界面

步骤

1. 将备份还原到其他命名空间、将括号中的值替换为环境中的信息。此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的 `source1:dest1,source2:dest2`卷。例如：

```
tridentctl-protect create backuprestore <my_restore_name> \
--backup <backup_namespace>/<backup_to_restore> \
```

```
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name<custom_app_name>\
-n <application_namespace>
```

从备份还原到原始命名空间

您可以随时将备份还原到原始命名空间。当您执行就地恢复时，Trident Protect 会自动管理保护计划和正在进行的操作，以防止无效的恢复点：

- 在开始还原之前，将禁用应用程序的所有已启用的保护计划。这会阻止在还原应用程序资源时运行定时备份或快照。
- 成功完成还原后，仅重新启用还原之前启用的计划。已禁用的计划将保持禁用状态。
- 在恢复开始之前，任何正在进行的备份或快照操作都将被取消。如果操作未在 5 分钟内取消，还原将继续，并在还原 CR 状态下记录警告。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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"。

备注	当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。

使用**CR**

步骤

1. 创建自定义资源(CR)文件并将其命名为 `trident-protect-backup-ipr-cr.yaml`。
2. 在创建的文件中、配置以下属性：
   - 。**metadata.name**:(*required_*)此自定义资源的名称；请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**：(*required*)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、请添加包含或排除带有特定标签的资源的筛选：

备注	Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

- **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
  - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
    - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
    - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
    - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
    - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
    - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
    - *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。例如：`"trident.netapp.io/os=linux"`。

      例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
```

```
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 使用正确的值填充文件后 `trident-protect-backup-ipr-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-backup-ipr-cr.yaml
```

使用命令行界面

步骤

1. 将备份还原到原始命名空间、将括号中的值替换为环境中的信息。 `backup`参数使用格式为的命名空间和备份名称 `<namespace>/<name>`。例如：

```
tridentctl-protect create backupinplacerestore <my_restore_name> \
--backup <namespace/backup_to_restore> \
-n <application_namespace>
```

从备份还原到其他集群

如果原始集群出现问题、您可以将备份还原到其他集群。

备注

- 当您使用 Kopia 作为数据移动器恢复备份时，您可以选择在 CR 中指定注释或使用 CLI 来控制 Kopia 使用的临时存储的行为。请参阅 "文档"有关您可以配置的选项的更多信息。使用 `tridentctl-protect create --help`有关使用Trident Protect CLI 指定注释的更多信息，请参阅命令。
- 使用 CR 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

开始之前

确保满足以下前提条件：

- 目标集群已安装Trident Protect。
- 目标集群可以访问与存储备份的源集群相同的AppVault的分段路径。
- 确保在运行 AppVault CR 时，本地环境可以连接到 AppVault CR 中定义的对象存储桶。 `tridentctl-protect get appvaultcontent`命令。如果网络限制阻止访问，请改为从目标集群上的 pod 内运行Trident Protect CLI。
- 确保AWS会话令牌到期时间足以执行任何长时间运行的还原操作。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到期时间的详细信息、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 插件验证目标集群上是否存在 AppVault CR：

```
tridentctl-protect get appvault --context <destination_cluster_name>
```

备注 如果目标集群上不存在 AppVault CR，请按照 "使用Trident Protect AppVault 对象来管理存储桶。" 中的步骤创建它。

2. 查看目标集群上可用的 AppVault 的备份内容，并记录 `appArchivePath`要还原的备份：

```
tridentctl-protect get appvaultcontent <appvault_name> \
--show-resources backup \
--show-paths \
--context <destination_cluster_name>
```

运行此命令可显示AppVault中的可用备份、包括其原始集群、相应的应用程序名称、时间戳和归档路径。

示例输出：

```
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
|   CLUSTER   |    APP    |  TYPE  |      NAME       |         TIMESTAMP
|    PATH     |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
| production1 | wordpress | backup | wordpress-bkup-1| 2024-10-30
08:37:40 (UTC)| backuppath1 |
| production1 | wordpress | backup | wordpress-bkup-2| 2024-10-30
08:37:40 (UTC)| backuppath2 |
+-------------+-----------+--------+-----------------
+--------------------------+-------------+
```

3. 使用AppVault名称和归档路径将应用程序还原到目标集群：

备注 使用 CR 时，请确保目标集群上存在用于应用程序还原的命名空间。

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；请为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：(*required*)存储备份内容的AppVault的名称。

- **spec.appArchivePath**: (*Required*) AppVault 内存储备份内容的路径。使用步骤 2 中的命令查看备份内容并找到 `appArchivePath`要还原的备份。
- **spec.destinationApplicationName**：（可选）已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
- 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

  例如：

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: my-cr-name
  namespace: my-destination-namespace
spec:
  appVaultRef: appvault-name
  appArchivePath: my-backup-path
  destinationApplicationName: my-new-app-name
  namespaceMapping: [{"source": "my-source-namespace", "
destination": "my-destination-namespace"}]
```

3. 使用正确的值填充文件后 `trident-protect-backup-restore-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-backup-restore-cr.yaml
```

使用命令行界面

1. 使用以下命令还原应用程序、将括号中的值替换为环境中的信息。命名空间映射参数使用冒号分隔的命名空间将源命名空间映射到正确的目标命名空间、格式为SOURCE1：dest1、Source2：dest2。例如：

```
tridentctl-protect create backuprestore <restore_name> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--appvault <appvault_name> \
--path <backup_path> \
--destination-app-name <custom_app_name> \
--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) 文件从快照恢复数据，恢复到不同的命名空间或原始源命名空间。当您使用 SnapshotRestore CR 将快照还原到不同的命名空间时， Trident Protect 会在新的命名空间中还原应用程序，并为还原的应用程序创建一个应用程序 CR。为了保护已恢复的应用程序，可以创建按需备份或快照，或者制定保

护计划。

备注	• 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。如果您尝试恢复到 `StorageClass`如果使用不同的存储后端，则恢复操作将失败。 • 使用 CR 恢复到新命名空间时，必须先手动创建目标命名空间，然后再应用 CR。Trident Protect 仅在使用 CLI 时才会自动创建命名空间。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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；请为您的环境选择一个唯一且合理的名称。
   - **spec.appVaultRef**：(*required*)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.destinationApplicationName**：（可选）已还原应用程序的名称。如果提供，还原的应用程序将使用此名称。如果未提供，还原的应用程序将使用源应用程序名称。
   - 。**spec.namespaceMapping**:还原操作的源命名空间到目标命名空间的映射。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: my-destination-namespace
     spec:
       appVaultRef: appvault-name
       appArchivePath: my-snapshot-path
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (可 选)如果只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：

备注	Trident Protect 会自动选择一些资源，因为它们与您选择的资源存在关联。例如，如果您选择持久卷声明资源并且它有一个关联的 pod， Trident Protect 还会恢复关联的 pod。

   - **resourceFilter.resourceSourcedionCritera**：(筛选时需要)使用 `Include`或包含或 `Exclude`排除资源匹配备程序中定义的资源。添加以下resourceMatchers参数以定义要包括或排除的资源：
     - **resourceFilter.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。
       - **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
       - **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
       - *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。例如：`"trident.netapp.io/os=linux"`。

         例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "Include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 使用正确的值填充文件后 `trident-protect-snapshot-restore-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用命令行界面

步骤

1. 将快照还原到其他命名空间、将括号中的值替换为环境中的信息。

   ◦ `snapshot`参数使用格式为的命名空间和快照名称 `<namespace>/<name>`。

   ◦ 此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的 `source1:dest1,source2:dest2`卷。

   例如：

```
tridentctl-protect create snapshotrestore <my_restore_name> \
--snapshot <namespace/snapshot_to_restore> \
--namespace-mapping <source_to_destination_namespace_mapping> \
--destination-app-name <custom_app_name> \
-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。当您执行就地恢复时，Trident Protect 会自动管理保护计划和正在进行的操作，以防止无效的恢复点：

- 在开始还原之前，将禁用应用程序的所有已启用的保护计划。这会阻止在还原应用程序资源时运行定时备份或快照。
- 成功完成还原后，仅重新启用还原之前启用的计划。已禁用的计划将保持禁用状态。
- 在恢复开始之前，任何正在进行的备份或快照操作都将被取消。如果操作未在 5 分钟内取消，还原将继续，并在还原 CR 状态下记录警告。

开始之前

确保AWS会话令牌到期时间足以执行任何长时间运行的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到期时间的详细信息、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；请为您的环境选择一个唯一且合理的名称。

   ◦ **spec.appVaultRef**：(*required*)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
```

+ .(*Optional*) 如果仅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：

+ 注意：Trident Protect 会自动选择某些资源，因为这些资源与您所选资源之间存在关联关系。例如，如果您选择了持久卷声明资源，且该资源具有关联的 pod，则 Trident Protect 也将还原该关联的 pod。

+ * **resourceFilter.resourceSelectionCriteria**：（筛选必需）使用 `Include`或 `Exclude`以包含或排除在 `resourceMatchers` 中定义的资源。添加以下 `resourceMatchers` 参数以定义要包含或排除的资源： **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** 操作匹配，并且每个元素（**group**、**kind**、**version**）内的字段将作为 **AND** 操作匹配。* **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**：（***Optional***） 要筛选的资源的 **Kubernetes metadata.name** 字段中的名称。* **resourceMatchers[].namespaces**：（*Optional*） 要筛选的资源的 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"]
```

+ .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
```

使用命令行界面

步骤

1. 将快照还原到原始命名空间、将括号中的值替换为环境中的信息。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、已完成或失败的还原操作的状态。

步骤

1. 使用以下命令检索还原操作的状态、将括号中的值替换为环境中的信息：

```
kubectl get backuprestore -n <namespace_name> <my_restore_cr_name> -o
jsonpath='{.status}'
```

使用高级**Trident Protect** 恢复设置

您可以使用高级设置（例如注释、命名空间设置和存储选项）自定义恢复操作，以满足您的特定要求。

还原和故障转移操作期间的命名空间标注和标签

在还原和故障转移操作期间、目标命名空间中的标签和标注会与源命名空间中的标签和标注相匹配。此时将添加源命名空间中目标命名空间中不存在的标签或标注、并覆盖已存在的任何标签或标注、以便与源命名空间中的值匹配。仅存在于目标命名空间上的标签或标注保持不变。

备注	如果您使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。欲了解更多信息，请参阅"OpenShift安全上下文约束文档"。

在执行还原或故障转移操作之前、您可以通过设置Kubornetes环境变量来防止目标命名空间中的特定标注被覆盖 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。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
```

备注	执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置其他Trident Protect 舵图设置"。

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例显示了一个源和目标命名空间、每个命名空间都具有不同的标注和标签。您可以查看目标命名空间在操作前后的状态、以及标注和标签在目标命名空间中的组合或覆盖方式。

在执行还原或故障转移操作之前

下表说明了执行还原或故障转移操作之前示例源和目标名称卷的状态：

命名空间	标注	标签
命名空间ns-1 (源)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE"	• 环境=生产 • 合规性=HIPAA • name=ns-1
命名空间ns-2 (目标)	• 标注.One/键："TRUE" • 标注三个/项："false"	• Role=database

还原操作之后

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。已添加某些密钥、某些密钥已被覆盖、并且`name`标签已更新以与目标命名空间匹配：

命名空间	标注	标签
命名空间ns-2 (目标)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE" • 标注三个/项："false"	• name=nS-2 • 合规性=HIPAA • 环境=生产 • Role=database

支持的字段

本节介绍可用于恢复操作的其他字段。

存储类别映射

这 `spec.storageClassMapping`属性定义从源应用程序中的现有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别的集群之间迁移应用程序时或更改 BackupRestore 操作的存储后端时使用此功能。

- 示例： *

```
storageClassMapping:
  - destination: "destinationStorageClass1"
    source: "sourceStorageClass1"
  - destination: "destinationStorageClass2"
    source: "sourceStorageClass2"
```

支持的注释

本节列出了系统中支持配置各种行为的注解。如果用户未明确设置注解，系统将使用默认值。

标注	Type	Description	默认值
protected.trident.netapp.io/data-mover-timeout-sec	string	允许数据移动设备操作停止的最长时间（以秒为单位）。	“300”
protected.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	等待新创建的持久卷声明 (PVC) 到达的最大时间（以秒为单位）`Bound`操作失败前的阶段。适用于所有恢复 CR 类型（备份恢复、备份就地恢复、快照恢复、快照就地恢复）。如果您的存储后端或集群经常需要更多时间，请使用更高的值。	1200（20分钟）

标注	Type	Description	默认值
protect.trident.netapp.io/retain-mac-address	string	对于 KubeVirt 虚拟机，请设置为 `"true"` 以在还原时保留第一个网络接口的 MAC 地址，例如当应用程序许可证绑定到该地址时。适用于所有还原 CR 类型和 AppMirrorRelationship（在提升关系之前进行设置）。默认情况下，Trident Protect 会删除此 MAC 地址以防止冲突。如果源虚拟机仍在同一网络上运行，请先将其关闭或还原到隔离网络，以避免 MAC 地址重复。	未设置（MAC 地址已删除）

使用自定义资源还原资源时修改资源

资源转换使您能够在还原资源时对其进行修改。这在还原版本需要与原始版本不同时非常有用，例如，在将应用程序还原到其他集群时更新 ConfigMap 中的主机名。

资源修改的工作原理

`SnapshotRestore`、 `BackupRestore`、 `AppMirrorRelationship` 以及其他恢复资源中的 `transformations` 字段允许您在恢复过程中修改 Kubernetes 资源。当需要对恢复后的应用程序进行与原始应用程序不同的配置时（例如，更改主机名、注册表 URL、资源限制或环境变量），此功能非常有用。

资源转换使用 "RFC 6902 JSON Patch"操作和 "RFC 6901 JSON 指针"路径来定位和修改 Kubernetes 资源中的特定字段。

下面显示了包含资源转换的还原对象的基本结构：

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: my-restore
  namespace: target-namespace
spec:
  appVaultRef: my-vault
  appArchivePath: /path/to/snapshot
  namespaceMapping:
    - source: source-ns
      destination: target-ns
  transformations:
    - resource:
        kind: ConfigMap       # Required: resource kind
        group: ""             # Optional: API group (empty for core
resources)
        version: ""           # Optional: API version
        name: ""              # Optional: specific resource name
      operations:
        - op: replace         # Operation type
```

```
          path: "/data/key"    # JSON Pointer path
          value: "new-value"   # New value (for add/replace/test)
```

支持的资源

您可以将资源转换与符合以下标准的资源一起使用：

- **kind**（必填）：Kubernetes 资源类型（例如，`ConfigMap`、`Deployment`、`Pod`）
- **group**（可选）：API 组（例如，`apps`、`route.openshift.io`）——省略核心资源
- **version**（可选）：API 版本（例如，`v1`、`v1beta1`）
- 名称（可选）：仅按名称应用于特定资源

> 备注　当前不支持修改 PersistentVolumeClaims 和命名空间。

支持的操作

您可以使用以下操作修改资源：`add`、`copy`、`move`、`remove`、`replace`和 `test`。

向资源添加值

使用 `add` 操作在指定路径上添加新字段或值。您可以将数据添加到对象或数组。

使用**CR**

以下示例将节点选择器添加到 `Deployment`资源：

```
transformations:
  - resource:
      kind: Deployment
    operations:
      - op: add
        path: "/spec/template/spec/nodeSelector"
        value:
          "topology.kubernetes.io/zone": "us-east-1a"
          disktype: "ssd"
```

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'apps,v1,Deployment:add{"path":"/spec/template/spec/nodeSelector","valu
```

```
e":{"topology.kubernetes.io/zone":"us-east-1a","disktype":"ssd"}}'
```

复制资源中的值

使用 `copy` 操作将同一资源中的值从一个路径复制到另一个路径。源未更改。

使用**CR**

以下示例复制 `ConfigMap` 对象的数据键：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: copy
        from: "/data/source-key"
        path: "/data/backup-key"
```

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation ',v1,ConfigMap:copy{"from":"/data/source-
key","path":"/data/backup-key"}'
```

在资源内移动值

使用 `move` 操作将同一资源内的值从一个路径移动到另一个路径。源被删除，值被放置在目的地。

使用**CR**

以下示例重命名 `ConfigMap` 对象的数据键：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: move
        from: "/data/OLD_KEY"
```

```
        path: "/data/NEW_KEY"
```

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:move{"from":"/data/OLD_KEY","path":"/data/NEW_KEY"}'
```

从资源中删除值

使用 `remove` 操作可删除指定路径中的字段或值。

使用**CR**

以下示例从 `ConfigMap`资源中删除批注：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: remove
        path: "/metadata/annotations/kubectl.kubernetes.io~1last-
applied-configuration"
```

备注	在以上示例的路径中， `~1` 是 `/` 的 JSON 指针转义序列。

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:remove{"path":"/metadata/annotations/kubectl.kubernetes.
io~1last-applied-configuration"}'
```

替换资源中的值

使用 `replace` 操作替换指定路径下资源中的现有值。JSON 路径必须已存在。

使用**CR**

以下示例更改 `Route`对象的主机名：

```
transformations:
  - resource:
      kind: Route
      group: route.openshift.io
    operations:
      - op: replace
        path: "/spec/host"
        value: "prod.example.com"
```

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
'route.openshift.io,v1,Route:replace{"path":"/spec/host","value":"prod.
example.com"}'
```

测试资源修改

使用 `test`操作验证路径上的值是否与预期值匹配，然后再应用其他操作。如果测试失败，则回滚整个修改。

使用**CR**

仅当 `environment` 设置为 `staging` 时，以下示例才会更新 `database-host`：

```
transformations:
  - resource:
      kind: ConfigMap
    operations:
      - op: test
        path: "/data/environment"
        value: "staging"
      - op: replace
        path: "/data/database-host"
```

```
        value: "prod-db.example.com"
```

使用命令行界面

使用以下命令通过命令行运行此转换：

```
tridentctl-protect create snapshotrestore <restore_name> \
  --snapshot <namespace/snapshot-name> \
  --namespace-mapping <source-ns>:<dest-ns> \
  --transformation
',v1,ConfigMap:test{"path":"/data/environment","value":"staging"},repla
ce{"path":"/data/database-host","value":"prod-db.example.com"}'
```

使用NetApp SnapMirror复制应用程序

使用 **Trident Protect** 设置和管理应用程序复制

使用Trident Protect，您可以利用NetApp SnapMirror技术的异步复制功能，将数据和应用程序更改从一个存储后端复制到另一个存储后端，无论是在同一集群内还是在不同集群之间。

还原和故障转移操作期间的命名空间标注和标签

在还原和故障转移操作期间、目标命名空间中的标签和标注会与源命名空间中的标签和标注相匹配。此时将添加源命名空间中目标命名空间中不存在的标签或标注、并覆盖已存在的任何标签或标注、以便与源命名空间中的值匹配。仅存在于目标命名空间上的标签或标注保持不变。

备注	如果您使用 Red Hat OpenShift，请务必注意命名空间注释在 OpenShift 环境中的重要作用。命名空间注释确保恢复的 pod 遵守 OpenShift 安全上下文约束 (SCC) 定义的适当权限和安全配置，并且可以访问卷而不会出现权限问题。欲了解更多信息，请参阅"OpenShift安全上下文约束文档"。

在执行还原或故障转移操作之前、您可以通过设置Kubornetes环境变量来防止目标命名空间中的特定标注被覆盖 `RESTORE_SKIP_NAMESPACE_ANNOTATIONS`。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
```

备注	执行恢复或故障转移操作时，任何命名空间注释和标签都将生效。`restoreSkipNamespaceAnnotations` 和 `restoreSkipNamespaceLabels` 不参与恢复或故障转移操作。确保在初始 Helm 安装期间配置这些设置。欲了解更多信息，请参阅 "配置其

他Trident Protect 舵图设置"。

如果您使用 Helm 安装了源应用程序， `--create-namespace`国旗，给予特殊待遇 `name`标签键。在恢复或故障转移过程中， Trident Protect 会将此标签复制到目标命名空间，但如果源命名空间的值与源命名空间的值匹配，则会将值更新为目标命名空间的值。如果此值与源命名空间不匹配，则会将其复制到目标命名空间，而不做任何更改。

示例

以下示例显示了一个源和目标命名空间、每个命名空间都具有不同的标注和标签。您可以查看目标命名空间在操作前后的状态、以及标注和标签在目标命名空间中的组合或覆盖方式。

在执行还原或故障转移操作之前

下表说明了执行还原或故障转移操作之前示例源和目标名称卷的状态：

命名空间	标注	标签
命名空间ns-1 (源)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE"	• 环境=生产 • 合规性=HIPAA • name=ns-1
命名空间ns-2 (目标)	• 标注.One/键："TRUE" • 标注三个/项："false"	• Role=database

还原操作之后

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。已添加某些密钥、某些密钥已被覆盖、并且`name`标签已更新以与目标命名空间匹配：

命名空间	标注	标签
命名空间ns-2 (目标)	• 标注.One/键："updatedvalue" • 标注。双/键："TRUE" • 标注三个/项："false"	• name=nS-2 • 合规性=HIPAA • 环境=生产 • Role=database

备注 您可以配置Trident Protect 在数据保护操作期间冻结和解冻文件系统。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。

故障转移和反向操作期间的执行挂钩

当使用 AppMirror 关系保护您的应用程序时，您应该在故障转移和反转操作期间注意与执行挂钩相关的特定行为。

- 在故障转移期间，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，执行挂钩仍存在于应用程序中，并将在任何相关操作期间执行。
- 在反向同步或反向重新同步期间，应用程序上所有现有的执行钩子都将被移除。当源应用程序成为目标应用程序时，这些执行钩子将失效，并会被删除以阻止其执行。

要了解有关执行钩子的更多信息，请参阅"管理Trident Protect 执行钩子"。

设置复制关系

设置复制关系涉及以下方面：

- 选择Trident Protect 拍摄应用程序快照的频率（包括应用程序的 Kubernetes 资源以及应用程序每个卷的卷快照）。
- 选择复制计划(包括Kubbernetes资源以及永久性卷数据)
- 设置创建快照的时间

步骤

1. 在源集群上、为源应用程序创建AppVault。根据您的存储提供程序、修改中的示例"AppVault自定义资源"以适合您的环境：

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。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
- 。**spec.providerConfig**:(*required_*)存储使用指定提供程序访问AppVault所需的配置。为您的提供商选择一个BucketName和任何其他必要的详细信息。请记下所选的值、因为复制关系所需的其他CR文件会引用这些值。有关其他提供商的AppVault CRS示例、请参见"AppVault自定义资源"。
- 。**spec.providerCredentials**:(*required_*)存储对使用指定提供程序访问AppVault所需的任何凭据的引用。
  - 。**spec.providerCredentials.valueFromSecret**:(*required_*)表示凭据值应来自密钥。
    - **key**:(*required*)要从中选择的密钥的有效密钥。
    - **name**:(*required*)包含此字段值的机密的名称。必须位于同一命名空间中。
  - * spec.providerCredentials.secretAccessKey*:(*required_*)用于访问提供程序的访问密钥。名称*应与*。spec.providerCredentials.valueFromSecret.name*。
- 。**spec.providerType**:(*required_*)用于确定提供备份的内容；例如、NetApp ONTAP S3、通用S3、Google Cloud或Microsoft Azure。可能值：
  - aws
  - azure
  - GCP
  - 常规S3
  - ONTAP S3
  - StorageGRID S3

c. 使用正确的值填充文件后 `trident-protect-appvault-primary-source.yaml` 、应用CR：

```
kubectl apply -f trident-protect-appvault-primary-source.yaml -n
trident-protect
```

使用命令行界面创建**AppVault**

a. 创建AppVault、将括号中的值替换为您环境中的信息：

```
tridentctl-protect create vault Azure <vault-name> --account
<account-name> --bucket <bucket-name> --secret <secret-name> -n
trident-protect
```

2. 在源集群上、创建源应用程序CR：

使用**CR**创建源应用程序

a. 创建自定义资源(CR)文件并将其命名(例如 `trident-protect-app-source.yaml`)。

b. 配置以下属性：

- **metadata.name**:(*required_*)应用程序自定义资源的名称。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
- 。**spec.includedNamespaces**:(*required_*)一个由命名区域和关联标签组成的数组。使用命名空间名称、并可选择通过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属于此数组。

  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. 使用正确的值填充文件后 `trident-protect-app-source.yaml` 、应用CR：

```
kubectl apply -f trident-protect-app-source.yaml -n my-app-
namespace
```

使用命令行界面创建源应用程序

a. 创建源应用程序。例如：

```
tridentctl-protect create app <my-app-name> --namespaces
<namespaces-to-be-included> -n <my-app-namespace>
```

3. （可选）在源集群上，对源应用程序进行快照。此快照将用作目标集群上应用程序的基础。如果跳过此步骤，则需要等待下一次计划快照运行，以便获得最新的快照。要创建按需快照，请参阅 "创建按需快照"。
4. 在源集群上，创建复制计划 CR：

备注

除了下面提供的计划外，建议创建一个单独的每日快照计划，保留期为 7 天，以便在对等 ONTAP 集群之间维护通用快照。这可确保快照最多可用 7 天，但保留期可根据用户需求自定义。

如果发生故障转移，系统可以使用这些快照最多 7 天进行反向操作。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，而不是所有数据。

如果应用程序的现有计划已经满足所需的保留要求，则不需要额外的计划。

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.backup**䶵 刭：(*required*)此字段为必填字段、且值必须设置为0。
- *spec.enabled *：必须设置为true。
- 。**spec.granularity**:必须设置为 `Custom`。
- **spec.rec**发 规则：定义UTC时间的开始日期和重复间隔。
- **spec.snapshot**䶵 刭：必须设置为2。

YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: Schedule
metadata:
  name: appmirror-schedule
  namespace: my-app-namespace
```

```
spec:
  appVaultRef: my-appvault-name
  applicationRef: my-app-name
  backupRetention: "0"
  enabled: true
  granularity: Custom
  recurrenceRule: |-
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  snapshotRetention: "2"
```

i. 使用正确的值填充文件后 `trident-protect-schedule.yaml` 、应用CR：

```
kubectl apply -f trident-protect-schedule.yaml -n my-app-
namespace
```

使用 **CLI** 创建复制计划

a. 创建复制计划，并将括号中的值替换为您环境中的信息：

```
tridentctl-protect create schedule --name appmirror-schedule
--app <my_app_name> --appvault <my_app_vault> --granularity
Custom --recurrence-rule <rule> --snapshot-retention
<snapshot_retention_count> -n <my_app_namespace>
```

- 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. 在目标集群上，创建一个与您在源集群上应用的AppVault CR完全相同的源应用程序AppVault CR，并将其命名为(例如 `trident-protect-appvault-primary-destination.yaml`)。
6. 应用CR：

```
kubectl apply -f trident-protect-appvault-primary-destination.yaml -n
trident-protect
```

7. 在目标集群上为目标应用程序创建目标AppVault CR。根据您的存储提供程序、修改中的示例"AppVault自定义资源"以适合您的环境：

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。请记下您选择的名称、因为复制关系所需的其他CR文件会引用此值。
- 。**spec.providerConfig**:(*required_*)存储使用指定提供程序访问AppVault所需的配置。为您的提供商选择 `bucketName`以及任何其他必要的详细信息。请记下所选的值、因为复制关系所需的其他CR文件会引用这些值。有关其他提供商的AppVault CRS示例、请参见"AppVault自定义资源"。
- 。**spec.providerCredentials**:(*required_*)存储对使用指定提供程序访问AppVault所需的任何凭据的引用。
  - 。**spec.providerCredentials.valueFromSecret**:(*required_*)表示凭据值应来自密钥。
    - **key**:(*required*)要从中选择的密钥的有效密钥。
    - **name**:(*required*)包含此字段值的机密的名称。必须位于同一命名空间中。
  - * spec.providerCredentials.secretAccessKey*:(*required_*)用于访问提供程序的访问密钥。名称*应与*。spec.providerCredentials.valueFromSecret.name*。
- 。**spec.providerType**:(*required_*)用于确定提供备份的内容；例如、NetApp ONTAP S3、通用S3、Google Cloud或Microsoft Azure。可能值：
  - aws
  - azure
  - GCP
  - 常规S3
  - ONTAP S3
  - StorageGRID S3

c. 使用正确的值填充文件后 `trident-protect-appvault-secondary-destination.yaml` 、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。

备注	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 时才会自动创建命名空间。

使用**CR**创建**App**镜像 关系

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) App镜像 关系自定义资源的名称。
- **metadata.annotations.protect.trident.netapp.io/retain-mac-address**：（可选）设置为

`"true"`以保留目标集群上 KubeVirt 虚拟机的 MAC 地址。请参阅"使用高级Trident Protect 恢复设置"。

- 。**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**:(*required*_)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**:(*required*)此值必须与您在源应用程序CR中定义的源应用程序的名称匹配。
- **spec.sourceApplicationUID**: （必需）此值必须与您在源应用程序 CR 中定义的源应用程序的 UID 匹配。
- **spec.storageClassName**: (可选) 选择集群上有效的存储类的名称。存储类必须链接到与源环境建立对等连接的ONTAP存储 VM。如果未提供存储类，则默认使用集群上的默认存储类。
- **spec.storageClassMapping**：（*Optional*）将源存储类映射到同一 AMR 中各个 PVC 的目标存储类。每个条目都需要一个 `source`和 `destination`存储类名称。设置后，`storageClassMapping`将完全覆盖 `storageClassName`；没有匹配源条目的 PVC 使用集群默认存储类，而不是 `storageClassName`。

  在 AppMirrorRelationship CR 中使用 `storageClassMapping`时，请注意以下限制：

  - `storageClassMapping` 完全覆盖 `storageClassName`。当两个字段都设置时，`storageClassName` 将被忽略。
  - 源存储类在 `storageClassMapping`中没有匹配条目的 `PVC` 将使用集群默认存储类，而非 `storageClassName`。
  - 每个条目都需要同时提供 `source`和 `destination`值。
  - 更改目标存储类可能会强制删除并重新创建现有目标 PVC。
- **spec.rec发** 规则：定义UTC时间的开始日期和重复间隔。

  YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: AppMirrorRelationship
metadata:
  name: amr-16061e80-1b05-4e80-9d26-d326dc1953d8
  namespace: my-app-namespace
spec:
  desiredState: Established
  destinationAppVaultRef: generic-s3-trident-protect-dst-bucket-
8fe0b902-f369-4317-93d1-ad7f2edc02b5
  namespaceMapping:
    - destination: my-app-namespace
      source: my-app-namespace
  recurrenceRule: |-
```

```
    DTSTART:20220101T000200Z
    RRULE:FREQ=MINUTELY;INTERVAL=5
  sourceAppVaultRef: generic-s3-trident-protect-src-bucket-
b643cc50-0429-4ad5-971f-ac4a83621922
  sourceApplicationName: my-app-name
  sourceApplicationUID: 7498d32c-328e-4ddd-9029-122540866aeb
  storageClassName: sc-vsim-2
```

c. 使用正确的值填充文件后 `trident-protect-relationship.yaml` 、应用CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-
namespace
```

使用命令行界面创建**App**镜像 关系

a. 创建并应用 AppMirrorRelationship 对象，并将括号中的值替换为您环境中的信息：

```
tridentctl-protect create appmirrorrelationship
<name_of_appmirrorrelationship> --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> --storageclass-mapping
<src_class:dst_class> -n <application_namespace>
```

备注	使用 `--storageclass-mapping`将单个源存储类映射到目标存储类。指定一个或多个以逗号分隔的 `source:destination`对（例如， `"class-a:class-b,class-c:class-d"`）。使用`*:destination-class`作为通配符，将所有未映射的 PVC 映射到单个目标存储类。设置`--storageclass-mapping`后， `--storage-class`将被忽略。

使用单个目标存储类的示例 (**`--storage-class`**）：

```
tridentctl-protect create appmirrorrelationship my-amr
--destination-app-vault appvault2 --source-app-vault appvault1
--recurrence-rule
"DTSTART:20220101T000200Z\nRRULE:FREQ=MINUTELY;INTERVAL=5"
--source-app my-app --namespace-mapping "my-source-ns1:my-dest-
ns1,my-source-ns2:my-dest-ns2" --source-app-id 373f24c1-5769-
404c-93c3-5538af6ccc36 --storage-class my-storage-class -n my-
dest-ns1
```

9. (可 选)在目标集群上、检查复制关系的状态：

```
kubectl get amr -n my-app-namespace <relationship name> -o=jsonpath
='{.status}' | jq
```

故障转移到目标集群

使用Trident Protect，您可以将复制的应用程序故障转移到目标集群。此过程会停止复制关系，并将应用程序在目标集群上联机。如果源集群上的应用程序正在运行， Trident Protect 不会停止该应用程序。

此过程将故障转移到最新的快照。要故障转移到较旧的快照，请参阅 "使用 Trident Protect 故障切换到较旧的快照"。

步骤

1. (可选)要保留 KubeVirt 虚拟机的 MAC 地址，请在提升 AppMirrorRelationship 之前对其进行注释。请参见"使用高级Trident Protect 恢复设置"。

```
kubectl annotate appmirrorrelationship <relationship_name> -n
<application_namespace> protect.trident.netapp.io/retain-mac-address
="true"
```

2. 在目标集群上，编辑AppMirorRelationship CR文件(例如 `trident-protect-relationship.yaml`)，并将*spec.desiredState*的值更改为 `Promoted`。
3. 保存 CR 文件。
4. 应用CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. (可 选)在故障转移应用程序上创建所需的任何保护计划。
6. (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

重新同步操作将重新建立复制关系。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将被丢弃。

此过程会先停止目标集群上的应用程序、然后再重新建立复制。

重要说明	故障转移期间写入目标应用程序的所有数据都将丢失。

步骤

1. 可选：在源集群上、创建源应用程序的快照。这样可确保捕获源集群的最新更改。
2. 在目标集群上，编辑AppMirorRelationship CR文件(例如 `trident-protect-relationship.yaml`)，并将spec.desiredState的值更改为 `Established`。
3. 保存 CR 文件。
4. 应用CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

5. 如果您在目标集群上创建了任何保护计划来保护故障转移应用程序、请将其删除。任何保留的计划都会导致卷快照失败。

反向重新同步故障转移复制关系

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保留下来。

步骤

1. 在初始目标集群上、删除App镜像 关系CR。这会使目标成为源。如果新目标集群上仍有任何保护计划、请将其删除。
2. 通过将最初用于设置复制关系的CR文件应用于对等集群来设置复制关系。
3. 确保为新目标(初始源集群)配置了两个AppVault CRS。
4. 在另一个集群上设置复制关系、并配置反向值。

反转应用程序复制方向

当您反转复制方向时， Trident Protect 会将应用程序移动到目标存储后端，同时继续复制回原始源存储后端。Trident Protect 会停止源应用程序并将数据复制到目标位置，然后再故障转移到目标应用程序。

在这种情况下、您将交换源和目标。

步骤

1. 在源集群上、创建一个关闭快照：

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Sh关机Snapshot 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**：(*required*)此值必须与源应用程序CR文件的metadata.name字段匹配。

  YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ShutdownSnapshot
metadata:
  name: replication-shutdown-snapshot-afc4c564-e700-4b72-86c3-c08a5dbe844e
  namespace: my-app-namespace
spec:
  appVaultRef: generic-s3-trident-protect-src-bucket-04b6b4ec-46a3-420a-b351-45795e1b5e34
  applicationRef: my-app-name
```

c. 使用正确的值填充文件后 `trident-protect-shutdownsnapshot.yaml` 、应用CR：

```
kubectl apply -f trident-protect-shutdownsnapshot.yaml -n my-app-namespace
```

使用命令行界面创建关闭快照

a. 创建关闭快照、将括号中的值替换为环境中的信息。例如：

```
tridentctl-protect create shutdownsnapshot <my_shutdown_snapshot> --appvault <my_vault> --app <app_to_snapshot> -n <application_namespace>
```

2. 在源集群上、关闭快照完成后、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*shutdownfapp.statues.appArchivePath*的值，并记录文件路径的最后一部分(也称为基本名称；这将是最后一个斜杠之后的所有内容)：

```
k get shutdownsnapshot -n my-app-namespace <shutdown_snapshot_name> -o jsonpath='{.status.appArchivePath}'
```

4. 执行从新目标集群到新源集群的故障转移、并进行以下更改：

备注	在故障转移过程的第2步中、将字段包含 `spec.promotedSnapshot`在App镜像 关系CR文件中、并将其值设置为您在上述第3步中记录的基本名称。

5. 执行中的反向重新同步步骤[反向重新同步故障转移复制关系]。
6. 在新的源集群上启用保护计划。

结果

反向复制会导致以下操作：

- 系统会为原始源应用程序的Kubbernetes资源创建一个快照。
- 通过删除原始源应用程序的Kubernetes资源(保留PVC和PV)、可以正常停止原始源应用程序的Pod。
- 关闭Pod后、将为应用程序的卷创建快照并进行复制。
- SnapMirror关系将中断、从而使目标卷做好读/写准备。
- 此应用程序的Kubornetes资源将使用在初始源应用程序关闭后复制的卷数据从关闭前的快照中还原。
- 反向重新建立复制。

将应用程序故障恢复到原始源集群

使用Trident Protect，您可以通过以下步骤序列在故障转移操作后实现“故障恢复”。在此恢复原始复制方向的工作流程中， Trident Protect 会将任何应用程序更改复制（重新同步）回原始源应用程序，然后再反转复制方向。

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。

注意	请勿执行正常的重新同步操作、因为这会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复制关系]步骤。
2. 执行[反转应用程序复制方向]步骤。

删除复制关系

您可以随时删除复制关系。删除应用程序复制关系后、会导致两个单独的应用程序之间没有关系。

步骤

1. 在当前目标集群上、删除App镜像 关系CR：

```
kubectl delete -f trident-protect-relationship.yaml -n my-app-namespace
```

使用 **Trident Protect** 故障切换到较旧的快照

使用 Trident Protect 将复制的应用程序故障转移到较早的快照，从而允许从数据损坏或受损之前的某个时间点进行恢复。默认情况下，故障转移使用最近传输的快照。

注意	故障转移到较旧的快照会导致 ONTAP SnapMirror 删除目标集群上比所选快照更新的快照，因为它们不再有效。仅当可以接受此数据丢失时，才在此过程中添加注释。

开始之前

- AppMirrorRelationship 存在且处于 `Established`状态。要创建一个，请参阅"设置复制关系"。要验证状态，请运行 `kubectl get amr -n <namespace> <relationship_name> -o=jsonpath='{.status}'`。
- 源和目标集群使用带有 SnapMirror 复制的 ONTAP 存储后端。不支持 Azure NetApp Files 和其他非 ONTAP 后端。请参阅"SnapMirror复制的要求"。
- 源应用程序上存在一个快照计划（ `snapshotRetention` 大于 1）。

识别快照

每个快照都有一个完成时间。在源集群上使用 `kubectl`或使用 CLI 获取其归档路径。

备注	当源集群关闭或快照自定义资源不可用时，请使用 CLI。在目标集群或具有源 AppVault 的任何集群上运行该命令。

使用 **kubectl**

在源集群上使用 `kubectl`获取归档路径。

1. 在源集群上，列出快照并记下所需快照的 `completionTimestamp`：

```
kubectl get snapshots -n my-app-namespace
```

2. 获取其存档路径：

```
kubectl get snapshots <snapshot_name> -n my-app-namespace -o
jsonpath='{.status.appArchivePath}'
```

使用命令行界面

使用 Trident Protect CLI 获取存档路径。在目标集群上，或在任何具有源 AppVault 的集群上，列出快照。 `TIMESTAMP` 是完成时间， `PATH` 是存档路径。

```
tridentctl-protect get appvaultcontent <source_appvault_name> \
--app <app_name> \
--show-resources snapshot \
```

```
--show-paths
```

备注	源应用程序的 AppVault CR 必须存在于运行命令的集群上，并且您的环境必须能够连接到 AppVault CR 中定义的对象存储桶。如果网络限制阻止访问，请从集群上的 Pod 内运行 CLI。

basename 是路径中最后一个斜杠之后的所有内容。例如：

```
20231114173500_schedule-20231114173500_f8df8bd3-a727-4a14-b954-
9666aa618b55
```

备注	使用基本名称，而不是快照自定义资源的名称。

结果

将记录选定快照的基本名称。

故障切换到选定快照

指定要提升的快照，并通过添加注释来确认数据丢失。

步骤

1. 在目标集群上，将注释添加到 AppMirrorRelationship。如果没有注释，故障转移只能使用最近传输的快照。

   ```
   kubectl annotate appmirrorrelationship <relationship_name> -n my-app-
   namespace protect.trident.netapp.io/allow-promotion-with-older-
   snapshots="true"
   ```

2. 编辑 AppMirrorRelationship CR 文件（例如， `trident-protect-relationship.yaml`）：

   - 将 **spec.desiredState** 设置为 `Promoted`。
   - 将 **spec.promotedSnapshot** 设置为您在 [识别快照] 中记录的基础名称。

     例如：

   ```
   apiVersion: protect.trident.netapp.io/v1
   kind: AppMirrorRelationship
   metadata:
     name: <relationship_name>
     namespace: <namespace>
     annotations:
       protect.trident.netapp.io/allow-promotion-with-older-snapshots:
   "true"
   spec:
   ```

```
  desiredState: Promoted
  promotedSnapshot: <basename>
```

3. 应用CR：

```
kubectl apply -f trident-protect-relationship.yaml -n my-app-namespace
```

4. 检查关系的状态：

```
kubectl get amr -n my-app-namespace <relationship_name> -o=jsonpath
='{.status}' | jq
```

结果

关系移动到 `Promoted`状态，应用程序使用来自所选快照的数据在目标集群上运行。

如果关系保持在 `Promoting` 状态，并且 `status.error` 显示 `The snapshot provided is invalid or older than the most recently transferred snapshot`，则说明注释缺失或未设置为 `true`。请添加注释并重试。

下一步是什么？

要返回到原始集群，请执行"反向重新同步"，然后"反转复制方向"。

注意	要返回到原始集群，请不要执行正常的重新同步。正常的重新同步会丢弃故障转移的应用程序，并将关系返回到最新的源快照。

相关信息

- "使用 Trident Protect 设置和管理应用程序复制"
- "故障转移到目标集群"
- "重新同步故障转移复制关系"
- "将应用程序故障恢复到原始源集群"
- "创建数据保护计划"

使用Trident Protect 迁移应用程序

您可以通过恢复备份数据在集群之间或不同的存储类之间迁移您的应用程序。

备注	迁移应用程序时、为该应用程序配置的所有执行挂钩都会随该应用程序一起迁移。如果存在还原后执行挂钩、则它会在还原操作中自动运行。

备份和还原操作

要在以下情况下执行备份和还原操作、您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还原到同一集群。

步骤

1. 执行以下操作之一：
   a. "创建快照"(英文)
   b. "创建备份"(英文)
2. 在同一集群上、根据您是创建了快照还是备份、执行以下操作之一：
   a. "从快照还原数据"(英文)
   b. "从备份还原数据"(英文)

克隆到其他集群

要将应用程序克隆到不同的集群（执行跨集群克隆），请在源集群上创建备份，然后将备份还原到不同的集群。请确保目标集群上已安装Trident Protect。

备注	您可以使用在不同集群之间复制应用程序"SnapMirror 复制"。

步骤

1. "创建备份"(英文)
2. 确保已在目标集群上为包含备份的对象存储分段配置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另一个存储类。

例如(从还原CR中排除密钥)：

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: "${snapshotRestoreCRName}"
spec:
  appArchivePath: "${snapshotArchivePath}"
  appVaultRef: "${appVaultCRName}"
  namespaceMapping:
    - destination: "${destinationNamespace}"
      source: "${sourceNamespace}"
  storageClassMapping:
    - destination: "${destinationStorageClass}"
      source: "${sourceStorageClass}"
  resourceFilter:
    resourceMatchers:
```

```
    kind: Secret
    version: v1
  resourceSelectionCriteria: exclude
```

使用**CR**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；请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**：(*required*)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。将和 `my-destination-namespace`替换 `my-source-namespace`为您环境中的信息。

     ```
     ---
     apiVersion: protect.trident.netapp.io/v1
     kind: SnapshotRestore
     metadata:
       name: my-cr-name
       namespace: trident-protect
     spec:
       appArchivePath: my-snapshot-path
       appVaultRef: appvault-name
       namespaceMapping: [{"source": "my-source-namespace",
     "destination": "my-destination-namespace"}]
     ```

3. (可选)如果您只需要选择要还原的应用程序的某些资源、请添加包含或排除带有特定标签的资源的筛选：
   - **resourceFilter.resourceSourcesionCriteri**：(筛选时需要) `include or exclude`用于包含或排除在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.resourceMatcher**：resourceMatcher对象数组。如果在此数组中定义多个元素，它们将作为OR操作进行匹配，每个元素(组、种类、版本)中的字段将作为AND操作进行匹配。
       - **resourceMatcher[].group**：(可 选)要筛选的资源的组。
       - **resourceMatcher[].KIND**：(可 选)要筛选的资源种类。
       - **resourceMatcher[].version**：(可 选)要筛选的资源版本。

- **resourceMatcher[].names**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的名称。
- **resourceMatcher[].namespies**：(可 选)要筛选的资源的Kubernetes metadata.name字段中的命名空间。
- *resourceMatcher[].labelSelectors *：(可 选)资源的Kubernetes metadata.name字段中的标签选择器字符串，如中所定义 "Kubernetes 文档"。例如：`"trident.netapp.io/os=linux"`。

  例如：

```
spec:
  resourceFilter:
    resourceSelectionCriteria: "include"
    resourceMatchers:
      - group: my-resource-group-1
        kind: my-resource-kind-1
        version: my-resource-version-1
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
      - group: my-resource-group-2
        kind: my-resource-kind-2
        version: my-resource-version-2
        names: ["my-resource-names"]
        namespaces: ["my-resource-namespaces"]
        labelSelectors: ["trident.netapp.io/os=linux"]
```

4. 使用正确的值填充文件后 `trident-protect-snapshot-restore-cr.yaml` 、应用CR：

```
kubectl apply -f trident-protect-snapshot-restore-cr.yaml
```

使用命令行界面还原快照

步骤

1. 将快照还原到其他命名空间、将括号中的值替换为环境中的信息。

   - `snapshot`参数使用格式为的命名空间和快照名称 `<namespace>/<name>`。
   - 此 `namespace-mapping`参数使用冒号分隔的卷来将源卷的源卷映射到格式为的正确目标卷的`source1:dest1,source2:dest2`卷。

   例如：

```
tridentctl-protect create snapshotrestore <my_restore_name>
--snapshot <namespace/snapshot_to_restore> --namespace-mapping
<source_to_destination_namespace_mapping>
```

使用 **Trident Protect** 保护 **KubeVirt** 虚拟机

Trident Protect 支持您备份和恢复运行在 OpenShift Virtualization 上的 KubeVirt 虚拟机 (VM)。您可以一次性保护一个或多个命名空间中的所有 VM，也可以使用 `includedVirtualMachines` 字段按名称指定特定的 VM。Trident Protect 会在备份期间自动收集所有依赖的 VM 资源，在恢复期间，您可以恢复整个归档，也可以选择性地恢复单个 VM。

KubeVirt 保护模式

Trident Protect 支持两种方式来定义要保护的 KubeVirt 虚拟机。定义应用程序时，请选择一种模式：

模式	字段	**Description**
基于命名空间	`includedNamespaces`	保护一个或多个命名空间中的所有匹配资源。在命名空间条目上使用 `labelSelector`来控制动态包含哪些 VM。
VM 范围	`includedVirtualMachines`	通过命名空间和名称保护特定 KubeVirt 虚拟机。当 VM 分布在多个命名空间中或不希望保护所有命名空间资源时，请使用此模式。

您必须在 `ApplicationSpec` 中指定恰好一个字段 (`includedNamespaces` 或 `includedVirtualMachines`)。您不能在同一个应用程序中同时使用两者。

这两种模式都支持额外的筛选：

- 使用 `resourceFilter`可包括或排除特定资源。
- 使用 `includedClusterScopedResources`可包含兼容的集群范围的资源，例如 `StorageClass`对象。

备注

- Trident Protect 可以在数据保护操作期间冻结和解冻 KubeVirt VM 文件系统，以确保一致性。有关配置详细信息，请参见 "使用KubeVirt VM保护数据"。
- 要在同一时间点为 VM 的所有磁盘创建快照，请使用卷组快照。请参阅 "将卷组快照与 Trident Protect 结合使用"。

备份和快照期间收集的资源

当您为包含 KubeVirt 虚拟机的应用程序创建备份或快照时，Trident Protect 会收集选定的虚拟机资源并自动发现其依赖资源。

集合行为适用于两种情况：

- 使用 `includedVirtualMachines`的 VM 范围应用程序。
- 包含 KubeVirt `VirtualMachine` 资源的基于命名空间的应用程序。

VM 范围的集合

对于 VM 范围的应用程序，Trident Protect 会从 `includedVirtualMachines` 中列出的每个 `VirtualMachine` 开始，并发现还原该 VM 所需的依赖资源。

资源类型	范围	如何发现
`VirtualMachine`	命名空间	从 `includedVirtualMachines`列表中直接匹配。
`VirtualMachineInstance`	命名空间	与 VM 相同的命名空间和名称（如果 VM 正在运行）。
`VirtualMachineInstancetype`	命名空间	被 `spec.instancetype`引用，当 `kind: VirtualMachineInstancetype`时。
`VirtualMachineClusterInstancetype`	集群	由 `spec.instancetype` 引用，当 `kind` 未设置或为 `kind: VirtualMachineClusterInstancetype` 时。
`VirtualMachinePreference`	命名空间	由 `spec.preference`在 `kind: VirtualMachinePreference`时引用。
`VirtualMachineClusterPreference`	集群	在 `kind` 未设置或为 `kind: VirtualMachineClusterPreference` 时由 `spec.preference` 引用。
`DataVolume`	命名空间	被引用 `spec.template.spec.volumes[].dataVolume.name`。
`PersistentVolumeClaim`	命名空间	从 DataVolume 支持的 PVC 中发现，这些 PVC 使用与 DataVolume 相同的名称；`persistentVolumeClaim.claimName` 中的直接 PVC 引用；`memoryDump.claimName` 中的内存转储 PVC；以及 `ephemeral.persistentVolumeClaim.claimName` 中的临时 PVC。
`PersistentVolume`	集群	通过使用 `spec.volumeName`从每个已收集的 PVC 的后备 PV 中发现。
`Secret`	命名空间	从 `secret.secretName`中的卷机密发现；containerDisk 映像拉取机密；CloudInitNoCloud 和 CloudInitConfigDrive 用户数据和网络数据机密引用；Sysprep 机密；KernelBoot 容器映像拉取机密；以及 AccessCredentials SSH 或密码机密。
`ConfigMap`	命名空间	从卷 ConfigMaps 中发现，位于 `configMap.name` 和 Sysprep ConfigMaps。
`ServiceAccount`	命名空间	从 `serviceAccount.serviceAccountName`卷引用中发现。
`NetworkAttachmentDefinition`	命名空间	当网络名称不包含 `/` 时，从 `spec.template.spec.networks[].multus.networkName` 中的 Multus 网络中发现。仅收集相同命名空间的 NetworkAttachmentDefinition 引用。跳过跨命名空间引用。
`Pod`	命名空间	已为 `virt-launcher` Pods 通过使用标签选择器发现 `kubevirt.io=virt-launcher,vm.kubevirt.io/name=<vmName>`。
存储类、Trident 卷和 Trident 后端	集群	收集方式与基于命名空间的应用程序相同，基于为 VM 发现的 PVC。

备注	收集所有 VM 范围的命名空间资源后，Trident Protect 将应用标准 `resourceFilter`（如果已配置）。然后，集群范围的资源收集将正常进行。

基于命名空间的集合

对于基于命名空间的应用程序，Trident Protect 会扫描选定的命名空间并收集所有匹配的资源。如果发现任何 KubeVirt `VirtualMachine` 对象，Trident Protect 将自动收集其依赖资源，包括实例类型、首选项和 `virt-launcher` Pod，与 VM 范围内的应用程序相同。

这意味着，即使使用 `includedNamespaces`定义了应用程序，您的备份或快照中也会包含特定于 VM 的资源。

使用基于命名空间的应用程序定义保护虚拟机

创建基于命名空间的应用程序时，Trident Protect 会扫描选定的命名空间并收集所有资源，包括任何 KubeVirt VM。对于找到的每个虚拟机，Trident Protect 还会收集备份和还原该虚拟机所需的依赖资源。

基于命名空间的应用程序 CR 示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
```

创建应用程序后，请使用标准的 Trident Protect 工作流进行备份和快照：

- 有关快照 YAML 示例，请参见 "创建按需快照"。
- 有关备份 YAML 示例，请参见 "创建按需备份"。

还原基于命名空间的虚拟机应用程序的行为

由于基于命名空间的应用程序从所选命名空间收集资源，因此备份或快照可以包括 KubeVirt 运行时对象和 KubeVirt 托管元数据。其中某些对象不应直接还原，因为它们表示来自源集群的运行时状态。

在恢复期间，Trident Protect 应用 KubeVirt 感知转换来删除源集群元数据并跳过临时 KubeVirt 资源。这有助于 KubeVirt 在恢复的环境中重新创建虚拟机。

资源	转换
VirtualMachine	删除以 `kubevirt.io/`开头的标签和批注。
VirtualMachineInstance	删除以 `kubevirt.io/`开头的标签和批注。
VirtualMachineInstanceMigration	忽略而不还原。
VirtualMachineSnapshot	忽略而不还原。
VirtualMachineSnapshotContent	忽略而不还原。
Pod (`virt-launcher`)	当 Pod 具有 `kubevirt.io=virt-launcher`标签时跳过。

备注

- Trident Protect 以定义的顺序恢复 KubeVirt 资源，以避免依赖性问题。CDI DataVolumes

(`cdi.kubevirt.io`）将首先还原，以便在创建虚拟机对象之前提供虚拟机磁盘数据。之后，VirtualMachines (`kubevirt.io`）将在 VirtualMachineInstances 之前还原。

- 默认情况下，Trident Protect 会从每个还原的虚拟机的第一个网络接口中删除 MAC 地址。要保留它，请使用 `protect.trident.netapp.io/retain-mac-address` 注释。请参阅 "使用高级Trident Protect 恢复设置"。

使用标签选择器实现动态虚拟机保护

如果要在不列出每个虚拟机名称的情况下动态保护虚拟机，请使用带有标签选择器的基于命名空间的应用程序定义。

只有 VirtualMachine CR 需要标签。依赖资源（例如 DataVolumes、PVC、Secrets 和 ConfigMaps）会自动收集。

步骤

1. 使用共享标签标记虚拟机 CR：

```
kubectl label virtualmachine <vm_name 1> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_2> app=<label_value> -n
<vm_namespace>
```

2. 使用 `includedNamespaces`和 `labelSelector`创建应用程序 CR：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedNamespaces:
    - namespace: <vm_namespace>
      labelSelector:
        matchLabels:
          app: <label_value>
```

3. 使用标准 Trident Protect 工作流程创建备份和还原。

其他详细信息：

- 您可以随时通过更改标签来添加虚拟机或从保护中删除虚拟机：

```
kubectl label virtualmachine <vm_name_to_add> app=<label_value> -n
<vm_namespace>
kubectl label virtualmachine <vm_name_to_remove> app
```

```
=<different_label_value> -n <vm_namespace>
```

- 您还可以在多个命名空间中使用标签选择器：

```
spec:
  includedNamespaces:
    - namespace: <vm_namespace_1>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
    - namespace: <vm_namespace_2>
      labelSelector:
        matchLabels:
          <label_key>: <label_value>
```

使用 **includedVirtualMachines** 保护特定 **VM**

如果要保护一个或多个命名空间中的 VirtualMachines 命名集，并从存档中选择性地还原特定虚拟机，请使用 `includedVirtualMachines`此选项。

应用程序和还原 CR 都使用相同的 `includedVirtualMachines`选择器格式。

在不同上下文中使用此选择器：

- 在应用程序 CR 中，它定义了受保护的虚拟机。
- 在还原CR中，它定义了从备份或快照存档还原的虚拟机。

备注	在还原 CR 时， `includedVirtualMachines`是可选的。如果不包括此字段，则会恢复整个存档。指定后，仅还原列出的虚拟机及其从属资源。

下面的 YAML 显示了共享选择器结构：

```
includedVirtualMachines:
  - namespace: <vm_namespace_1>
    names:
      - <vm_name_1>
      - <vm_name_2>
  - namespace: <vm_namespace_2>
    names:
      - <vm_name_3>
```

备注	`includedVirtualMachines`中的多个条目可以引用不同的命名空间，从而支持多命名空间 VM 感知应用程序定义和还原。

字段	Type	Description
namespace	string	包含目标虚拟机对象的 Kubernetes 命名空间。
名称	字符串列表	该命名空间中的一个或多个虚拟机名称。

基于 **VM** 的应用程序示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace_1>
      names:
        - <vm_name_1>
        - <vm_name_2>
    - namespace: <vm_namespace_2>
      names:
        - <vm_name_3>
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

还原特定 KubeVirt VM

您可以在还原 CR 上使用 `includedVirtualMachines`，以仅还原备份、快照或复制快照存档中的选定 KubeVirt 虚拟机。

在还原 CR 上指定 `includedVirtualMachines`时，`Trident Protect` 仅还原列出的 `VirtualMachines` 及其从属资源，例如 `VirtualMachineInstance`、`DataVolume`、`PersistentVolumeClaim`、`PersistentVolume`、`Secret`、`ConfigMap`、`ServiceAccount`以及其他与 VM 相关的对象。

如果未在还原 CR 上指定 `includedVirtualMachines`，则 Trident Protect 将根据还原 CR 配置还原完整存档。

支持的恢复 **CR** 和 **CLI** 命令

下表列出了支持 `includedVirtualMachines`的恢复 CR 及其相应的 CLI 命令。

还原 **CR**	**CLI** 命令	Description
`BackupRestore`	`tridentctl-protect create backuprestore`	从备份还原到新的或不同的命名空间映射。支持 `namespaceMapping`和 `storageClassMapping`。

还原 CR	CLI 命令	Description
`BackupInplaceRestore`	`tridentctl-protect create backupinplacerestore`	从备份还原到原始命名空间或多个命名空间。
`SnapshotRestore`	`tridentctl-protect create snapshotrestore`	从快照还原到新的或不同的命名空间映射。支持 `namespaceMapping`和 `storageClassMapping`。
`SnapshotInplaceRestore`	`tridentctl-protect create snapshotinplacerestore`	从快照还原到原始命名空间或多个命名空间。
`ReplicateSnapshotRestore`	仅限 YAML，或使用受支持的 CLI 命令（如果在您的环境中可用）	从复制的快照还原。支持 `namespaceMapping`、`storageClassMapping`以及使用 `inPlaceRestore`标志进行就地还原。

备注	在恢复命令中， `tridentctl-protect` CLI 没有用于 `includedVirtualMachines`的标志。要恢复特定的 VM，您必须使用 `kubectl apply`直接应用恢复 CR YAML 文件。您可以从头开始编写 YAML，也可以结合使用 CLI 和 `--dry-run --output yaml`来生成初始文件，在 `spec`下添加 `includedVirtualMachines`字段，然后应用该文件。

VM 范围的还原行为

恢复 CR 上的 `includedVirtualMachines`字段使用与应用程序 CR 相同的选择器格式：

```
includedVirtualMachines:
  - namespace: <source_vm_namespace>
    names:
      - <vm_name_1>
      - <vm_name_2>
```

对于还原 CR， `includedVirtualMachines` 中的命名空间值是指备份、快照或复制快照存档中存在的源命名空间。

如果使用 `namespaceMapping`，Trident Protect 首先从源存档中选择请求的虚拟机，然后在还原期间应用命名空间映射。如果使用 `storageClassMapping`，Trident Protect 会在还原转换阶段应用存储类映射。

使用 CLI 生成还原 YAML

您可以使用 `tridentctl-protect`生成基本还原 `CR`，然后编辑 `YAML` 以添加 `includedVirtualMachines`。

```
tridentctl-protect create <restore_command> <restore_name> \
  <restore_options> \
```

```
  --dry-run --output yaml > <restore_name>.yaml
```

编辑生成的 YAML 并在 `spec` 下添加 `includedVirtualMachines` 字段，然后应用该文件：

```
kubectl apply -f <restore_name>.yaml
```

示例：定义基于 **VM** 的应用程序

以下应用程序保护两个命名空间中的三个特定虚拟机。

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: Application
metadata:
  name: production-vms
  namespace: prod
spec:
  includedVirtualMachines:
    - namespace: prod
      names:
        - app-server
        - database-server
    - namespace: monitoring
      names:
        - prometheus-vm
  includedClusterScopedResources:
    - group: storage.k8s.io
      kind: StorageClass
```

示例：创建备份

为基于 VM 的应用程序创建备份时，需使用标准备份 CR。备份 CR 上不需要特定于 VM 的字段。

```yaml
apiVersion: protect.trident.netapp.io/v1
kind: Backup
metadata:
  name: daily-backup
  namespace: prod
spec:
  applicationRef: production-vms
  appVaultRef: s3-vault
```

示例：从备份还原单个虚拟机

以下示例仅从备份存档还原 database-server。存档中的其他虚拟机（例如 app-server`和 `prometheus-vm）将被跳过。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupRestore
metadata:
  name: restore-db-only
  namespace: prod-vms-restore
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  namespaceMapping:
    - source: prod
      destination: prod-dr
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f restore-db-only.yaml
```

仅 `database-server`及其依赖资源（例如 DataVolumes、PVC、PV、Secrets、ConfigMaps 和相关 KubeVirt 资源）将被还原。

示例：从备份原位还原单个虚拟机

以下示例仅将 `database-server`从备份存档还原到其原始命名空间。

```
apiVersion: protect.trident.netapp.io/v1
kind: BackupInplaceRestore
metadata:
  name: ipr-db-only
  namespace: prod
spec:
  appVaultRef: s3-vault
  appArchivePath: backups/production-vms/2026-03-10T00-00-00Z
  includedVirtualMachines:
    - namespace: prod
      names:
        - database-server
```

```
kubectl apply -f ipr-db-only.yaml
```

示例：从快照还原单个虚拟机

以下示例仅从快照存档还原 `ubuntu-vm-blue-a627be38`。

```
apiVersion: protect.trident.netapp.io/v1
kind: SnapshotRestore
metadata:
  name: blue-vm-snap-restore
  namespace: blue
spec:
  appVaultRef: my-appvault
  appArchivePath: snapshots/protectctl-blue-vm/snap-1
  includedVirtualMachines:
    - namespace: blue
      names:
        - ubuntu-vm-blue-a627be38
```

```
kubectl apply -f blue-vm-snap-restore.yaml
```

将虚拟机还原与其他筛选器相结合

您可以在同一还原 CR 中将 `includedVirtualMachines`与 `resourceFilter`、`namespaceMapping`和 `storageClassMapping`结合使用，以微调还原的内容。

添加 `resourceFilter`时，Trident Protect 首先选择 `includedVirtualMachines`中列出的虚拟机及其依赖资源。然后，它应用资源过滤器来进一步缩小这些结果中包含或排除的内容。

```
spec:
  includedVirtualMachines:
    - namespace: <vm_namespace>
      names:
        - <vm_name>
  resourceFilter:
    resourceSelectionCriteria: Include
    resourceMatchers:
      - kinds: ["PersistentVolumeClaim"]
        names: ["<pvc_name>"]
```

对于 `BackupRestore`、`SnapshotRestore`和 `ReplicateSnapshotRestore`，`namespaceMapping`和 `storageClassMapping`与 `includedVirtualMachines`协同工作。`includedVirtualMachines`中的命名空间引用对应于存档中显示的源命名空间。命名空间和存储类映射在过滤完成后的还原转换阶段应用。

有关还原行为详细信息，请参阅 "使用Trident Protect 恢复应用程序"。

使用 **CLI** 创建 **VM** 范围的应用程序

使用 `--virtual-machines` 标志将应用程序的范围限定为特定 KubeVirt 虚拟机，而不是整个命名空间。`--virtual-machines` 标志与 `--namespaces` 互斥。

步骤

1. 通过运行以下命令之一创建应用程序，将括号中的值替换为您环境中的信息。 `--virtual-machines` 标志的格式为 `<namespace>(<vm_name>)`。对于多个虚拟机，请在括号内使用逗号分隔的名称：
   - 命名空间中的单个 VM：

```
tridentctl-protect create application <application_name> \
  -n <application_namespace> \
  --virtual-machines "<vm_namespace>(<vm_name>)"
```

   - 跨命名空间的多个虚拟机：

```
tridentctl-protect create application <application_name> \
  -n <application_namespace> \
  --virtual-machines "<vm_namespace_1>(<vm_name_1>,<vm_name_2>
),<vm_namespace_2>(<vm_name_3>)"
```

相关信息

- "定义要管理的应用程序"
- "保护应用程序"
- "使用Trident Protect 恢复应用程序"
- "使用高级还原设置"
- "使用NetApp SnapMirror和Trident Protect 复制应用程序"
- "安装Trident Protect CLI 插件"
- "使用AppVault对象管理分段"

管理**Trident Protect** 执行钩子

执行挂钩是一种自定义操作、您可以将其配置为与受管应用程序的数据保护操作结合运行。例如、如果您有一个数据库应用程序、则可以使用执行挂钩在快照之前暂停所有数据库事务、并在快照完成后恢复事务。这样可以确保应用程序一致的快照。

执行挂钩的类型

Trident Protect 支持以下几种执行钩子类型，具体取决于它们的运行时机：

- 预快照
- 快照后
- 预备份
- 备份后
- 还原后
- 故障转移后

执行顺序

运行数据保护操作时、执行钩事件按以下顺序发生：

1. 任何适用的自定义操作前执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义操作前挂钩、但操作前这些挂钩的执行顺序既不能保证也不可配置。
2. 如果适用，则会发生文件系统冻结。"了解更多关于使用Trident Protect 配置文件系统冻结的信息"。
3. 执行数据保护操作。
4. 如果适用、冻结的文件系统将被解除冻结。
5. 任何适用的自定义操作后执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义操作后挂机、但这些挂机在操作后的执行顺序既不能保证也不可配置。

如果创建多个相同类型的执行挂钩(例如、预快照)、则无法保证这些挂钩的执行顺序。但是、可以保证不同类型的挂钩的执行顺序。例如、以下是具有所有不同类型挂钩的配置的执行顺序：

1. 已执行预快照挂钩
2. 已执行后快照挂钩
3. 已执行备份前的挂钩
4. 已执行备份后挂钩

备注	只有在运行不使用现有快照的备份时、上述顺序示例才适用。
备注	在生产环境中启用执行钩脚本之前，应始终对其进行测试。您可以使用 "kubectl exec" 命令方便地测试脚本。在生产环境中启用执行挂钩后、请测试生成的快照和备份、以确保它们一致。为此、您可以将应用程序克隆到临时命名空间、还原快照或备份、然后测试应用程序。
备注	如果快照前执行挂钩添加、更改或删除了Kubornetes资源、则这些更改将包括在快照或备份以及任何后续还原操作中。

有关自定义执行挂钩的重要注意事项

在为应用程序规划执行挂钩时，请考虑以下几点。

- 执行挂钩必须使用脚本执行操作。许多执行挂钩可以引用同一个脚本。
- Trident Protect 要求执行钩子使用的脚本以可执行 shell 脚本的格式编写。
- 脚本大小限制为96 KB。

- Trident Protect 使用执行钩子设置和任何匹配条件来确定哪些钩子适用于快照、备份或恢复操作。

备注	由于执行挂钩通常会减少或完全禁用其运行的应用程序的功能，因此您应始终尽量缩短自定义执行挂钩运行所需的时间。如果使用关联的执行挂钩启动备份或快照操作、但随后将其取消、则在备份或快照操作已开始时、仍允许运行这些挂钩。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。对于在所有容器上使用相同容器映像的应用程序、筛选器非常有用、但可能会将每个映像用于不同的用途(例如Elasticsearch)。通过筛选器、您可以创建执行挂钩在某些容器上运行的方案、但不一定是所有相同的容器上运行的方案。如果为单个执行钩创建多个筛选器、则这些筛选器将与逻辑运算符和运算符结合使用。每个执行连接最多可以有10个活动筛选器。

添加到执行挂钩的每个过滤器都使用正则表达式来匹配集群中的容器。当钩子与容器匹配时，钩子将在该容器上运行其关联的脚本。过滤器的正则表达式使用正则表达式 2 (RE2) 语法，该语法不支持创建从匹配列表中排除容器的过滤器。有关Trident Protect 在执行钩子过滤器中支持的正则表达式语法的详细信息，请参阅 "正则表达式2(RE2)语法支持"。

备注	如果将命名空间筛选器添加到在还原或克隆操作之后运行的执行挂钩、并且还原或克隆源和目标位于不同的命名空间中、则命名空间筛选器仅会应用于目标命名空间。

执行钩示例

请访问 "NetApp Verda GitHub项目" 、下载适用于Apache cassandr和Elascearch等常见应用程序的真实执行挂钩。您还可以查看示例并了解如何构建自己的自定义执行挂钩。

备注	Verda 存储库中的脚本按原样提供，不受 NetApp 的正式支持。NetApp 支持仅涵盖 Trident Protect 中的执行挂钩框架。

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钩子。您需要拥有所有者、管理员或成员权限才能创建执行钩子。

使用**CR**

步骤

1. 创建自定义资源(CR)文件并将其命名为 `trident-protect-hook.yaml`。
2. 配置以下属性以匹配您的Trident Protect 环境和集群配置：
   - 。**metadata.name**:(*required_*)此自定义资源的名称；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要运行执行挂钩的应用程序的Kubernetes名称。
   - *spec.stage *：(*required*)一个字符串，指示执行挂钩应在操作期间的哪个阶段运行。可能值：
     - 预
     - 发布

- **spec.action**：(*required*)一个字符串，指示执行挂钩将执行的操作，假设指定的任何执行挂钩过滤器都匹配。可能值：
  - Snapshot
  - 备份
  - 还原
  - 故障转移
- *spec.enabled *：(可 选)指示此执行挂钩是启用还是禁用。如果未指定、则默认值为true。
- **spec.hookSource**：(*required*)包含base64编码的挂钩脚本的字符串。
- *spec.timeout *：(可 选)一个数字，用于定义允许执行挂钩运行多长时间(以分钟为单位)。最小值为1分钟、如果未指定、则默认值为25分钟。
- **spic.arge**件：(可 选)可为执行挂钩指定的YAML参数列表。
- *spec.匹 配Criteria：(可 选)标准键值对的可选列表，每个对构成执行挂钩筛选器。每个执行挂钩最多可以添加10个筛选器。
- **spec.匹 配Cricera.type**：(可 选)标识执行挂钩筛选器类型的字符串。可能值：
  - 内容管理器映像
  - 内容名
  - 播客名称
  - PodLabel
  - NamespaceName
- **spec.匹 配Cricery.value**：(可 选)用于标识执行挂钩筛选器值的字符串或正则表达式。

YAML示例：

```
apiVersion: protect.trident.netapp.io/v1
kind: ExecHook
metadata:
  name: example-hook-cr
  namespace: my-app-namespace
  annotations:
    astra.netapp.io/astra-control-hook-source-id:
/account/test/hookSource/id
spec:
  applicationRef: my-app-name
  stage: Pre
  action: Snapshot
  enabled: true
  hookSource: IyEvYmluL2Jhc2gKZWNobyAiZXhhbXBsZSBzY3JpcHQiCg==
  timeout: 10
  arguments:
    - FirstExampleArg
    - SecondExampleArg
```

```
  matchingCriteria:
    - type: containerName
      value: mysql
    - type: containerImage
      value: bitnami/mysql
    - type: podName
      value: mysql
    - type: namespaceName
      value: mysql-a
    - type: podLabel
      value: app.kubernetes.io/component=primary
    - type: podLabel
      value: helm.sh/chart=mysql-10.1.0
    - type: podLabel
      value: deployment-type=production
```

3. 使用正确的值填充CR文件后、应用CR：

```
kubectl apply -f trident-protect-hook.yaml
```

使用命令行界面

步骤

1. 创建执行挂钩、将括号中的值替换为环境中的信息。例如：

```
tridentctl-protect create exechook <my_exec_hook_name> --action
<action_type> --app <app_to_use_hook> --stage <pre_or_post_stage>
--source-file <script-file> -n <application_namespace>
```

手动运行执行挂钩

您可以手动运行执行挂钩以进行测试、或者在发生故障后需要手动重新运行挂钩。要手动运行执行挂钩、您需要具有所有者、管理员或成员权限。

手动运行执行挂钩包含两个基本步骤：

1. 创建资源备份、此备份用于收集资源并为其创建备份、从而确定挂钩的运行位置
2. 对备份运行执行挂钩

▼ 第**1**步：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；请为您的环境选择一个唯一且合理的名称。
   - **spec.applicationRef**：(*required*)要为其创建备份资源的应用程序的Kubernetes名称。
   - **spec.appVaultRef**：(*required*)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}'
     ```

     YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ResourceBackup
metadata:
  name: example-resource-backup
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
```

3. 使用正确的值填充CR文件后、应用CR：

```
kubectl apply -f trident-protect-resource-backup.yaml
```

使用命令行界面

步骤

1. 创建备份、将括号中的值替换为您环境中的信息。例如：

```
tridentctl protect create resourcebackup <my_backup_name> --app
<my_app_name> --appvault <my_appvault_name> -n <my_app_namespace>
--app-archive-path <app_archive_path>
```

2. 查看备份状态。您可以重复使用以下示例命令、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>
```

▼ 第2步：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；请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中的应用程序名称匹配。
   - **spec.appVaultRef**：(*required*)确保此值与您在步骤1中创建的ResourceBackup CR中的appVaultRef匹配。
   - **spec.appArchivePath**：确保此值与您在步骤1中创建的ResourceBackup CR中的appArchivePath匹配。

     ```
     kubectl get backups <BACKUP_NAME> -n my-app-namespace -o
     jsonpath='{.status.appArchivePath}'
     ```

   - **spec.action**：(*required*)一个字符串，指示执行挂钩将执行的操作，假设指定的任何执行挂钩过滤器都匹配。可能值：
     - Snapshot
     - 备份
     - 还原
     - 故障转移
   - *spec.stage *：(*required*)一个字符串，指示执行挂钩应在操作期间的哪个阶段运行。此挂钩运行不会在任何其他阶段运行挂钩。可能值：
     - 预
     - 发布

     YAML示例：

```
---
apiVersion: protect.trident.netapp.io/v1
kind: ExecHooksRun
metadata:
```

```
  name: example-hook-run
spec:
  applicationRef: my-app-name
  appVaultRef: my-appvault-name
  appArchivePath: example-resource-backup
  stage: Post
  action: Failover
```

3. 使用正确的值填充CR文件后、应用CR：

```
kubectl apply -f trident-protect-hook-run.yaml
```

使用命令行界面

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exech本 运行对象以查看最终详细信息和状态：

```
kubectl -n <my_app_namespace> describe exechooksrun
<my_exec_hook_run_name>
```

卸载Trident Protect

如果您要从试用版升级到完整版产品，可能需要移除Trident Protect 组件。

要移除Trident Protect，请执行以下步骤。

步骤

1. 删除Trident Protect CR 文件：

备注 | 25.06 及更高版本不需要此步骤。

```
helm uninstall -n trident-protect trident-protect-crds
```

2. 移除Trident保护：

```
helm uninstall -n trident-protect trident-protect
```

3. 移除Trident Protect 命名空间：

```
kubectl delete ns trident-protect
```

Trident和Trident Protect 博客

您可以在这里找到一些很棒的NetApp Trident和Trident Protect 博客：

Trident博客

- 2025年10月16日："面向 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 虚拟化中使用卷组快照"
- 2025年5月9日： "使用 Amazon EKS 插件自动为 FSx for ONTAP 配置 Trident 后端"
- 2025年4月15日： "NetApp Trident 与 Google Cloud NetApp Volumes 的 SMB 协议"
- 2025年4月14日： "利用Trident 25.02 的光纤通道协议实现 Kubernetes 上的持久存储"
- 2025年4月14日： "释放 NetApp ASA r2 系统对 Kubernetes 块存储的强大功能"
- 2025年3月31日： "使用新的认证操作员简化 Red Hat OpenShift 上的Trident安装"
- 2025年3月27日： "使用Google Cloud NetApp Volumes为 SMB 配置Trident"
- 2025年3月5日： "解锁无缝iSCSI存储集成：适用于AWS的ROSA集群上的FSxN指南"
- 2025年2月27日： "使用Trident、GKE 和Google Cloud NetApp Volumes部署云身份"
- 2024年12月12日： "介绍Trident中的光纤通道支持"
- 2024年11月11日： "NetApp Trident与Google Cloud NetApp Volumes"
- 2024年10月29日： "使用Trident在 AWS 上将Amazon FSx for NetApp ONTAP与 Red Hat OpenShift 服务 (ROSA) 结合使用"
- 2024年10月29日： "使用 ROSA 上的 OpenShift 虚拟化和Amazon FSx for NetApp ONTAP实时迁移虚拟机"
- 2024年7月8日： "使用 NVMe/TCP 在 Amazon EKS 上为现代容器化应用程序使用ONTAP存储"
- 2024年7月1日： "使用Google Cloud NetApp Volumes Flex 和Astra Trident实现 Kubernetes 无缝存储"
- 2024年6月11日： "ONTAP作为 OpenShift 中集成映像注册表的后端存储"

Trident Protect博客

- 2025年5月16日："利用Trident Protect 的恢复后钩子实现注册表故障转移的自动化，以进行灾难恢复"
- 2025年5月16日： "使用NetApp Trident Protect 进行 OpenShift 虚拟化灾难恢复"
- 2025年5月13日："使用Trident Protect 备份和恢复进行存储类迁移"
- 2025年5月9日："使用Trident Protect 恢复后钩子重新扩展 Kubernetes 应用程序"
- 2025年4月3日： "Trident Protect 升级：Kubernetes 复制保护与灾难恢复"
- 2025年3月13日："为OpenShift虚拟化VM执行崩溃状态一致的备份和还原操作"
- 2025年3月11日："利用NetApp Trident将GitOps模式扩展到应用程序数据保护"
- 2025年3月3日："Trident 25.02：借助激动人心的新功能提升Red Hat OpenShift体验"
- 2025年1月15日："隆重推出Trident Protect 基于角色的访问控制"

- 2024年11月11日："Kubernetes驱动的数据管理： Trident Protect开启新时代"

知识和支持

常见问题解答

查找有关Trident安装、配置、升级和故障排除的常见问题解答。

一般问题

Trident的发布频率如何?

从24.02版开始、Trident每四个月发布一次：2月、6月和10月。

Trident是否支持在特定版本的**Kubnetes**中发布的所有功能?

Trident通常不支持Kubbernetes中的Alpha功能。在 Kubernetes 测试版之后的两个 Trident 版本中， Trident 可能支持测试版功能。

Trident是否依赖于其他**NetApp**产品才能正常运行?

Trident不依赖于其他NetApp软件产品、它可作为独立应用程序运行。但是，您应具有 NetApp 后端存储设备。

如何获取完整的**Trident**配置详细信息?

使用 `tridentctl get`命令可获取有关Trident配置的详细信息。

我能否获得有关**Trident**如何配置存储的指标?

是。Prometheus端点、可用于收集有关Trident操作的信息、例如受管后端的数量、配置的卷数量、已用字节数等。您还可以使用"Cloud Insights"进行监控和分析。

使用**Trident**作为**CSI**配置程序时、用户体验是否会发生变化?

否。就用户体验和功能而言、没有变化。使用的配置程序名称为 `csi.trident.netapp.io`。如果要使用当前版本和未来版本提供的所有新功能、建议使用此方法安装Trident。

在**Kubbernetes**集群上安装和使用**Trident**

Trident是否支持从私有注册表进行脱机安装?

是的，Trident 可以脱机安装。请参见 "了解Trident安装"。

是否可以远程安装**Trident**?

是。Trident 18.10及更高版本支持从有权访问集群的任何计算机远程安装 `kubectl`。验证访问后 `kubectl`(例如、从远程计算机启动 `kubectl get nodes`命令进行验证)、请按照安装说明进行操作。

是否可以使用**Trident**配置高可用性?

Trident作为KubeNet部署(复制集)安装、具有一个实例、因此内置了HA。您不应增加部署中的副本数量。如果安装了Trident的节点丢失或无法访问此Pod、Kubnetes会自动将此Pod重新部署到集群中运行状况良好的节点。Trident仅支持控制平台、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Kubednetes API服务器读取数据、以确定应用程序何时请求新的PVC、因此需要访问Kube-system。

Trident使用哪些角色和**Privileges**?

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"。

是否可以在本地生成**Trident**用于安装的确切清单文件?

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"。

对于两个单独的**Kubnetes**集群的两个单独的**Trident**实例、我是否可以共享同一个**ONTAP**后端**SVM**?

尽管不建议这样做、但您可以对两个Trident实例使用相同的后端SVM。在安装期间为每个实例指定一个唯一的卷名称和/或在文件中指定一个唯一的 `StoragePrefix`参数 `setup/backend.json`。这是为了确保不会对这两个实例使用相同的FlexVol volume。

是否可以在存储**Linux (**以前称为**CoreTM OS)**下安装**Trident**?

Trident只是一个Kubbernetes Pod、可以安装在运行Kubbernetes的任何位置。

是否可以将**Trident**与**NetApp Cloud Volumes ONTAP**结合使用?

是的、Trident在AWS、Google Cloud和Azure上受支持。

故障排除和支持

NetApp是否支持**Trident**?

尽管Trident是开源的、并且是免费提供的、但只要您的NetApp后端受支持、NetApp就完全支持它。

如何提出支持案例?

要提交支持案例，请执行以下操作之一：

1. 请联系您的支持客户经理并获得帮助以提交服务单。
2. 联系以提出支持案例 "NetApp 支持"。

如何生成支持日志包?

您可以运行 `tridentctl logs -a` 来创建支持包。除了在捆绑包中捕获的日志之外，还可以捕获 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。

如果需要提出新功能请求，我该怎么办？

创建问题 "Trident Github"并在问题的主题和说明中提及*RFE*。

我应在何处提出缺陷？

在上创建问题 "Trident Github"。请务必包含与问题描述相关的所有必要信息和日志。

如果我对**Trident**有快速问题需要澄清、会发生什么情况？是否有社区或论坛？

如果您有任何疑问、问题或请求、请通过我们的Trident或GitHub与我们联系"渠道不和"。

我的存储系统密码已更改、**Trident**不再工作、如何恢复？

使用更新后端的密码 `tridentctl update backend myBackend -f </path/to_new_backend.json> -n trident`。替换 `myBackend` 在示例中、使用后端名称、和 `/path/to_new_backend.json` 路径正确 `backend.json` 文件

Trident找不到我的**Kubbernetes**节点。如何修复此问题？

Trident找不到Kubnetes节点的可能情形有两种。这可能是因为 Kubernetes 中的网络问题描述或 DNS 问题描述。在每个 Kubernetes 节点上运行的 Trident 节点取消设置必须能够与 Trident 控制器进行通信，以便向 Trident 注册该节点。如果在安装Trident后发生网络连接更改、则只有在添加到集群中的新Kubnetes节点上才会遇到此问题。

如果 **Trident POD** 被销毁，是否会丢失数据？

如果 Trident POD 被销毁，数据不会丢失。三元数据存储在CRD对象中。已由 Trident 配置的所有 PV 都将正常运行。

升级Trident

是否可以直接从旧版本升级到新版本（跳过几个版本）？

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 18.xx 升级到 19.xx ，从 19.xx 升级到 20.xx 等。在生产部署之前，您应在实验室中测试升级。

是否可以将 **Trident** 降级到先前版本？

如果您需要修复在升级、依赖关系问题或升级失败或不完整后发现的错误、则应"卸载Trident"按照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

是否需要在**ONTAP**后端定义文件中同时定义管理和数据**LIF**？

管理LIF为必填项。DataLIF因情况而异：

- ONTAP SAN：不为iSCSI指定。Trident使用"ONTAP 选择性LUN映射"发现建立多路径会话所需的iCI LUN。如果明确定义、则会生成警告 `dataLIF`。有关详细信息、请参见 "ONTAP SAN配置选项和示例" 。

- ONTAP NAS：NetApp建议指定 `dataLIF`。如果未提供此参数、则Trident将从SVM提取数据LIF。您可以指定要用于NFS挂载操作的完全限定域名(FQDN)、以便创建轮叫DNS来在多个dataLIF之间进行负载平衡。有关详细信息、请参见"ONTAP NAS配置选项和示例"

Trident是否可以为**ONTAP**后端配置**CHAP**?

是。Trident支持对ONTAP后端使用双向CHAP。这需要在后端配置中进行设置 `useCHAP=true`。

如何使用**Trident**管理导出策略?

从20.04版开始、Trident可以动态创建和管理导出策略。这样，存储管理员便可在其后端配置中提供一个或多个CIDR 块，并使 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会自动管理在给定CIDR中具有IP的节点的规则添加和删除。

IPv6地址是否可用于管理和**DataLIF**?

Trident支持为以下项定义IPv6地址：

- `managementLIF` 和 `dataLIF` 适用于ONTAP NAS后端。
- `managementLIF` 适用于ONTAP SAN后端。您无法指定 `dataLIF` 在ONTAP SAN后端。

必须使用标志(对于 `tridentctl`安装)、(对于`Trident operator`)或(对于 `tridentTPv6`Helm安装)安装`Trident `--use-ipv6`、`IPv6`才能使其在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 。

是否可以更新后端的**DataLIF**?

您只能在和 `ontap-nas-economy`上更新`DataLIF `ontap-nas`。

是否可以在**Trident**中为**Kubnetes**创建多个后端?

Trident可以同时支持多个后端、可以使用相同的驱动程序、也可以使用不同的驱动程序。

Trident如何存储后端凭据?

Trident将后端凭据存储为Kubnetes密码。

Trident如何选择特定后端?

如果无法使用后端属性自动为某个类选择合适的池，则会使用 `storagePools` 和 `addtionalStoragePools` 参数来选择一组特定的池。

如何确保**Trident**不会从特定后端进行配置?

`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。

如果有多个同类型的后端、Trident如何选择要使用的后端?

如果有多个已配置的相同类型的后端，Trident将根据和 `PersistentVolumeClaim`中的参数选择适当的后端 `StorageClass`。例如，如果有多个ONTAP－NAS驱动程序后端，则Trident会尝试匹配和 `PersistentVolumeClaim`中的参数， `StorageClass`并组合和匹配可满足和 `PersistentVolumeClaim`中所列要求的后端 `StorageClass`。如果有多个后端与请求匹配、则Trident会随机从其中一个后端中进行选择。

Trident是否支持使用Element或SolidFire的双向CHAP?

是的。

Trident如何在ONTAP卷上部署qtrees？一个卷可以部署多少个 qtree ？

该驱动程序可 `ontap-nas-economy`在同一个FlexVol volume中创建多达200个qtrees (可在50到300之间配置)、每个集群节点创建100、000个qtrees、每个集群创建240万个qtrees。当您输入由经济型驱动程序提供服务的新 `PersistentVolumeClaim`时、驱动程序将查看是否已存在可为新qtree提供服务的FlexVol volume。如果不存在可为qtree提供服务的FlexVol volume、则会创建一个新的FlexVol volume。

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不会在Kubbernetes中将挂载选项设置为任何值。要在Kubnetes存储类中指定挂载选项，请按照给定的示例进行操作"此处"。

如何将配置的卷设置为特定导出策略?

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。

如何使用ONTAP通过Trident设置卷加密?

您可以使用后端定义文件中的加密参数在 Trident 配置的卷上设置加密。有关详细信息、请参见："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` 。如果同时配置了这两者，则 `StorageClass` 优先。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，通过将 `backend spaceReserve` 参数 设置为 volume 来启用厚卷。

如何确保即使意外删除了 **PVC** 也不会删除所使用的卷?

从版本 1.10 开始， Kubernetes 会自动启用 PVC 保护。

是否可以增加**Trident**创建的**NFS PVC**的大小?

可以。您可以扩展由 Trident 创建的 NFS PVC。Trident Autogrow Policies 可以根据容量阈值自动扩展 PVC 。ONTAP 卷自动调整大小是独立的，适用于受支持的 ONTAP SAN 驱动程序：它会增加后备 FlexVol 的大小，而不改变 PVC 的大小。有关详细信息，请参见"ONTAP 卷自动调整大小"。

是否可以在卷处于 **SnapMirror** 数据保护（ **DP** ）或脱机模式时导入它?

如果外部卷处于 DP 模式或脱机，则卷导入将失败。您会收到以下错误消息：

```
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.
```

如何将资源配额转换为 **NetApp** 集群?

只要 NetApp 存储具有容量， Kubernetes 存储资源配额就应起作用。如果NetApp存储因容量不足而无法支持Kubbernetes配额设置、则Trident会尝试配置、但会出错。

是否可以使用**Trident**创建卷快照?

是。Trident支持按需创建卷快照以及从快照创建持久卷。要从快照创建PV,请确保 `VolumeSnapshotDataSource`已启用功能门。

哪些驱动程序支持**Trident**卷快照?

截至今日，我们的产品已提供按需快照支持。 `ontap-nas` ， `ontap-nas-flexgroup` ， `ontap-san` ， `ontap-san-economy` ， `solidfire-san` ， 和 `azure-netapp-files`后端驱动程序。

如何使用**ONTAP**为**Trident**配置的卷创建快照备份?

此功能可从 `ontap-nas` ， `ontap-san` 和 `ontap-nas-flexgroup` 驱动程序获得。您也可以在 FlexVol 级别为 `ontap-san-economy` 驱动程序指定 `snapshotPolicy` 。

此功能也适用于 `ontap-nas-economy`驱动程序、但适用于FlexVol volume级别粒度、而不适用于qtree级别粒度。要为Trident配置的卷创建快照、请将backend参数选项设置为ONTAP后端 `snapshotPolicy`上定义的所需快照策略。Trident无法识别存储控制器创建的任何快照。

是否可以为通过**Trident**配置的卷设置快照预留百分比?

可以、您可以通过在后端定义文件中设置属性来预留特定百分比的磁盘空间、用于通过Trident存储Snapshot副本 `snapshotReserve`。如果您已在后端定义文件中配置 `snapshotPolicy`和 `snapshotReserve`、则会根据后端文件中提及的百分比设置快照预留百分比 `snapshotReserve`。如果未提及此 `snapshotReserve`百分比数、则默认情况下、ONTAP会将快照预留百分比设置为5。如果此 `snapshotPolicy`选项设置为none、则快照

预留百分比将设置为0。

是否可以直接访问卷快照目录和复制文件?

可以，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 Snapshot 目录。

是否可以通过**Trident**为卷设置**SnapMirror**?

目前，必须使用 ONTAP 命令行界面或 OnCommand 系统管理器在外部设置 SnapMirror 。

如何将永久性卷还原到特定 **ONTAP** 快照?

要将卷还原到 ONTAP 快照，请执行以下步骤：

1. 暂停正在使用永久性卷的应用程序 POD 。
2. 通过 ONTAP 命令行界面或 OnCommand 系统管理器还原到所需的快照。
3. 重新启动应用程序 POD 。

Trident是否可以在配置了负载共享镜像的**SVM**上配置卷?

可以为通过NFS提供数据的SVM的根卷创建负载共享镜像。ONTAP 会自动为Trident创建的卷更新负载共享镜像。这可能会导致卷挂载延迟。使用Trident创建多个卷时、配置卷取决于ONTAP 更新负载共享镜像。

如何区分每个客户 / 租户的存储类使用情况?

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的特定存储类的使用。要拒绝特定命名空间对特定存储的访问，请将该存储类的资源配额设置为 0 。

故障排除

使用此处提供的指针解决您在安装和使用Trident时可能遇到的问题。

备注	要获得有关Trident的帮助、请使用创建支持包 `tridentctl logs -a -n trident`并将其发送给NetApp支持部门。

常规故障排除

- 如果 Trident Pod 无法正常启动（例如，当 Trident Pod 停留在 `ContainerCreating` 阶段且已准备好的容器少于两个时），则运行 `kubectl -n trident describe deployment trident` and `kubectl -n trident describe pod trident -%%.%.%.* -*` 可以提供更多见解。获取 kubelet 日志（例如，通过 `jlogalctl -xeu kubelet` ）也会很有帮助。
- 如果 Trident 日志中的信息不足，您可以根据安装选项将 ` -d` 标志传递到 install 参数，尝试为 Trident 启用调试模式。

  然后，使用 ` ./tridentctl logs -n trident` 并在日志中搜索 `level=debug msg` 来确认是否设置了调试。

随操作员一起安装

```
kubectl patch torc trident -n <namespace> --type=merge -p
'{"spec":{"debug":true}}'
```

此操作将重新启动所有 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" 列来检查此问题。

对于Trident 20.07和20.10，请使用 `tprov`代替 `torc`。

随 **Helm** 一起安装

```
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来获取每个后端的调试日志 `debugTraceFlags`。例如、include `debugTraceFlags: {"api":true, "method":true,}`可在Trident日志中获取API调用和方法迁移。现有后端可以 `debugTraceFlags`配置 `tridentctl backend update`。
- 使用Red Hat Enterprise Linux CoreTM OS (RHCOS)时、请确保 `iscsid`已在工作节点上启用并默认启动。可以使用 OpenShift MachineConfigs 或修改点燃模板来完成此操作。
- 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。
- 如果在将 PV 挂载到容器时出现问题，请确保 `rpcbind` 已安装并正在运行。对主机操作系统使用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` 或其等效项来检查 `rpcbind` 服务的状态。
- 如果 Trident 后端报告其处于 `Failed` 状态，尽管之前已正常工作，但这可能是由于更改了与后端关联的 SVM/ 管理员凭据所致。使用 `tridentctl update backend` 更新后端信息或放弃 Trident Pod 将修复此问题描述。
- 如果在容器运行时安装 Trident 时遇到权限问题，请使用 ` -in cluster=false` 标志尝试安装 Trident 。这不会使用安装程序 POD ，并可避免因 `trident` 安装程序 用户而出现权限问题。
- 使用 `uninstall` 参数 `<Uninstalling Trident >` 在运行失败后进行清理。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三端到功能、请先运行 `tridentctl uninstall` 用于删除Trident的命令。下载所需的 "Trident 版本" 并使用安装 `tridentctl install` 命令：
- 成功安装后，如果 PVC 停留在 `Pending` 阶段，则运行 `kubectl describe PVC` 可以提供追加信息，说明 Trident 为何无法为此 PVC 配置 PV 。

使用操作员无法成功部署TRIdent

如果使用操作员部署 Trident ，则 `TridentOrchestrator` 的状态将从 `Installing` 更改为 `Installed` 。如果您观察到 `failed` 状态，并且操作员无法自行恢复，则应运行以下命令来检查操作员的日志：

```
tridentctl logs -l trident-operator
```

跟踪 trident 操作器容器的日志可能会指向问题所在。例如，其中一个问题描述可能是无法从运行良好的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，您应查看 `TridentOrchestrator` 的状态。

```
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，因此操作员可确保在任意给定时间只存在一个可创建的活动 `TridentOrchestrator` 。

此外，观察 Trident Pod 的状态通常可以指示情况是否不正确。

```
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，由于未提取一个或多个容器映像， Pod 无法完全初始化。

要解决此问题，您应编辑 `TridentOrchestrator` CR 。或者，您也可以删除 `TridentOrchestrator` ，并使用修改后的准确定义创建一个新的。

使用无法成功部署TRIYent `tridentctl`

为了帮助您找出出现问题的原因，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
```

解决此问题后，您可以按如下所示清理安装，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.
```

完全删除Trident和CRD

您可以完全删除Trident和所有创建的CRD以及关联的自定义资源。

警告	此操作无法撤消。除非您需要全新安装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 运算符

要使用Trident运算符卸载Trident并完全删除CRD、请执行以下操作：

```
kubectl patch torc <trident-orchestrator-name> --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

掌舵

要使用Helm卸载Trident并完全删除CRD、请执行以下操作：

```
kubectl patch torc trident --type=merge -p
'{"spec":{"wipeout":["crds"],"uninstall":true}}'
```

<code>tridentctl</code>

使用卸载Trident后完全删除CRD `tridentctl`

```
tridentctl obliviate crd
```

使用Kubnetes 1.26上的rwx原始块命名区卸载NVMe节点失败

如果您运行的是Kubnetes 1.26、则在对rwx原始块命名区使用NVMe/TCP时、节点取消暂存可能会失败。以下场景提供了故障的临时解决策。或者、您也可以将Kubbernetes升级到1.27。

已删除命名空间和**POD**

假设您已将Trident托管命名空间(NVMe永久性卷)连接到Pod。如果直接从ONTAP后端删除命名空间、则取消暂存过程会在您尝试删除Pod后停滞。此情形不会影响Kubornetes集群或其他功能。

临时解决策

从相应节点卸载永久性卷(与该命名空间对应)并将其删除。

已阻止数据**LIF**

```
 If you block (or bring down) all the dataLIFs of the NVMe Trident
backend, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
```

```
Kubernetes node.
```
.临时解决策

启动`dataLIF`以恢复完整功能。

已删除命名空间映射

```
 If you remove the `hostNQN` of the worker node from the corresponding
subsystem, the unstaging process gets stuck when you attempt to delete the
pod. In this scenario, you cannot run any NVMe CLI commands on the
Kubernetes node.
```
.临时解决策

添加 `hostNQN` 返回到子系统。

当预期启用“v4.2-xattrs”时，NFSv4.2 客户端在升级ONTAP后报告“无效参数”

升级ONTAP后，NFSv4.2 客户端在尝试挂载 NFSv4.2 导出时可能会报告“无效参数”错误。当 `v4.2-xattrs` SVM 上未启用该选项。.解决方法启用 `v4.2-xattrs` 选项或升级到ONTAP 9.12.1 或更高版本，默认情况下此选项处于启用状态。

支持

NetApp 以多种方式为 Trident 提供支持。全天候提供广泛的免费自助支持选项、例如知识库(KB)文章和不和渠道。

Trident支持生命周期

Trident根据您的版本提供三个支持级别。请参阅 "NetApp软件版本支持定义"。

完全支持
: Trident提供自发布之日起12个月内的全面支持。

支持有限
: Trident在发布日期起的13到24个月内提供有限支持。

自助支持
: Trident文档自发布之日起的25到36个月内可用。

version	全面支持终止	停止有限支持	自助支持终止
"26.10"	2027年10月	2028年10月	2029 年 10 月
"26.06"	2027年7月	2028年7月	2029年7月
"26.02"	2027年2月	2028年2月	2029年2月
"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

自助支持

要查看完整的故障排除文章列表，请参阅 "NetApp 知识库（需要登录）"。

社区支持

我们的上有一个由容器用户(包括Trident开发人员)组成的活跃公共社区"渠道不和"。这是一个很好的地方，可以提出有关项目的一般问题，并与志同道合的同行讨论相关主题。

NetApp技术支持

要获得有关Trident的帮助，请使用创建支持包 `tridentctl logs -a -n trident`并将其发送到 `NetApp Support <Getting Help>`。

有关详细信息 ...

- "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）	后通道 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。

备注	在安装期间、可以使用更改活跃度/就绪性探测端口 `--probe-port` 标志。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

对于在非Kubnetes部署中使用Trident作为独立二进制文件的高级安装、REST API非常有用。

为了提高安全性、默认情况下、在Pod中运行时、Trident `REST API`仅限于本地主机。要更改此行为、您需要在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trdentctrd管理Trident"。

API 的工作原理如下：

获取

`GET <trident-address>/trident/v1/<object-type>`

列出该类型的所有对象。

`GET <trident-address>/trident/v1/<object-type>/<object-name>`

获得命名对象的详细信息。

发布

`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配置 JSON 。有关每种对象类型的规范，请参见"使用trdentctrd管理Trident"。
- 如果对象已存在，则行为会有所不同：后端更新现有对象，而所有其他对象类型将使操作失败。

删除

`DELETE <trident-address>/trident/v1/<object-type>/<object-name>`

删除命名资源。

备注 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trdentctrd管理Trident"。

命令行选项

Trident为Trident流程编排程序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(调试、信息、警告、错误、致命)。默认为 INFO 。

Kubernetes

-k8s_pod

使用此选项或 `-k8s_api_server` 以启用Kubernetes支持。如果设置此值，则 Trident 将使用其所属 POD 的 Kubernetes 服务帐户凭据来联系 API 服务器。只有当 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>

使用此选项或启用Kubornetes `-k8s_pod`支持。指定后， `Trident` 将使用提供的不安全地址和端口连接到 `Kubernetes API` 服务器。这样、`Trident`便可部署在`POD`之外；但是、它仅支持与`API`服务器的不安全连接。要安全连接、请使用选项在`POD`中部署`Trident` `-k8s_pod`。

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、而不侦听UNIX域套接字。

-config <file>

必需；您必须指定后端配置文件的此路径。

REST

-address <ip-or-host>

指定要侦听的三端存储服务器的地址。默认为 localhost 。在本地主机上侦听并在 Kubernetes Pod 中运行时，无法从 Pod 外部直接访问 REST 接口。使用 … `-address ""` 可从Pod IP地址访问REST接口。

注意	可以将 Trident REST 接口配置为仅以 127.0.0.1 （对于 IPv4 ）或（：： 1 ）（对于 IPv6 ）侦听和提供服务。

-port <port-number>

指定应侦听的三端存储服务器的端口。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来使用 REST API 与 Kubernetes 和 Trident 进行交互。Kubernetes 与 Trident ， Trident 与存储以及 Kubernetes 与存储之间的关系由多个资源对象决定。其中一些对象通过 Kubernetes 进行管理，而另一些对象则通过 Trident 进行管理。

对象如何相互交互？

了解对象，对象的用途以及对象交互方式的最简单方法可能是，遵循 Kubernetes 用户的单个存储请求：

1. 用户创建一个 `PersistentVolumeClaim` ，请求管理员先前配置的 Kubernetes `StorageClass` 中具有特定大小的新 `PersistentVolume` 。
2. Kubernetes `StorageClass` 将 Trident 标识为其配置程序，并包含一些参数，用于指示 Trident 如何为请求的类配置卷。
3. Trident 会查看自己的 `StorageClass` 并使用相同的名称来标识匹配的 `Backend` 和 `StoragePools` ，它可以使用这些卷为该类配置卷。
4. Trident 会在匹配的后端配置存储并创建两个对象： Kubernetes 中的 A `PersistentVolume` 用于告知 Kubernetes 如何查找，挂载和处理卷； Trident 中的一个卷用于保留 `PersistentVolume` 与实际存储之间的关系。
5. Kubernetes 会将 `PersistentVolumeClaim` 绑定到新的 `PersistentVolume` 。包含 `PersistentVolumeClaim` 的 Pod 会将此 PersistentVolume 挂载到其运行所在的任何主机上。
6. 用户使用指向 Trident 的 `VolumeSnapshotClass` 创建现有 PVC 的 `VolumeSnapshot` 。
7. Trident 标识与 PVC 关联的卷，并在其后端创建卷的快照。此外，它还会创建一个 `VolumeSnapshotContent` ，指示 Kubernetes 如何识别快照。
8. 用户可以使用 `VolumeSnapshot` 作为源创建 `PersistentVolumeClaim` 。
9. Trident 可识别所需的快照，并执行与创建 `PersistentVolume` 和 `Volume` 相同的一组步骤。

提示	要进一步了解 Kubernetes 对象，强烈建议您阅读 "永久性卷" Kubernetes 文档的一节。

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，如果用户要覆盖在后端配置中设置的默认值， Trident 还允许用户指定以下特定于卷的标注：

标注	卷选项	支持的驱动程序
trident.netapp.io/fileSystem	文件系统	ontap-san、solidfire-san、ontap-san-economy.
trident.netapp.io/cloneFromPVC	cloneSourceVolume	ontap-nas、ontap-san、solidfire-san、azure-netapp-files、ontap-san-economy

标注	卷选项	支持的驱动程序
trident.netapp.io/splitOnClone	splitOnClone	ontap-NAS ， ontap-san
trident.netapp.io/protocol	协议	任意
trident.netapp.io/exportPolicy	导出策略	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	块大小	solidfire-san
trident.netapp.io/skipRecoveryQueue	跳过恢复队列	ontap-nas、ontap-nas-economy 、ontap-nas-flexgroup、ontap-san、ontap-san-economy

如果创建的 PV 具有 `Delete` reclaime 策略，则在释放 PV 时（即用户删除 PVC 时）， Trident 会同时删除 PV 和后备卷。如果删除操作失败， Trident 会将 PV 标记为相应的 PV ，并定期重试此操作，直到操作成功或 PV 手动删除为止。如果 PV 使用 ` retain` 策略， Trident 会忽略该策略，并假定管理员将从 Kubernetes 和后端清理该策略，以便在删除卷之前对其进行备份或检查。请注意，删除 PV 不会通过发生原因 Trident 删除后备卷。您应使用 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数据源来克隆现有 PVC 。这样， PV 的时间点副本就可以以快照的形式公开给 Kubernetes 。然后，可以使用快照创建新的 PV 。请查看 ` + 按需卷快照 +` 了解其工作原理。

Trident还提供 `cloneFromPVC` 和 `splitOnClone` 用于创建克隆的标注。您可以使用这些标注克隆PVC、而无需使用CSI实施。

例如：如果用户已经有一个名为 `mysql` 的 PVC ，则用户可以使用标注创建一个名为 `mysqlclone` 的新 PVC ，例如 `trident.netapp.io/cloneFromPVC: mysql` 。设置了此标注后， Trident 将克隆与 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。
- 一个 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 标注 `trident.netapp.io/splitOnClone` 与 `trident.netapp.io/cloneFromPVC` 结合使用。当 `trident.netapp.io/splitOnClone` 设置为 `true` 时， Trident 会将克隆的卷与父卷拆分，从而将克隆的卷与其父卷的生命周期完全分离，从而降低存储效率。如果不设置 `trident.netapp.io/splitOnClone` 或将其设置为 `false` ，则会减少后端的空间占用，而会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。拆分克隆是有意义的一种情形，即克隆空数据库卷时，该卷及其克隆会发生很大的差异，无法从 ONTAP 提供的存储效率中受益。

。 `sample-input` 目录包含用于Trident的PVC定义示例。请参见 有关与三项技术的卷关联的参数和设置的完整问题描述。

Kubernetes `PersistentVolume` 对象

Kubernetes `PersistentVolume` 对象表示可供 Kubernetes 集群使用的一段存储。它的生命周期与使用它的 POD 无关。

备注	Trident 会创建 `PersistentVolume` 对象，并根据其配置的卷自动将其注册到 Kubernetes 集群中。您不应自行管理它们。

创建引用基于 Trident 的 `S存储类` 的 PVC 时， Trident 会使用相应的存储类配置新卷并为该卷注册新的 PV 。在配置已配置的卷和相应的 PV 时， Trident 会遵循以下规则：

- Trident 会为 Kubernetes 生成 PV 名称及其用于配置存储的内部名称。在这两种情况下，它都可以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匹配，但可能会根据平台将其取整为最接近的可分配数量。

Kubernetes `StorageClass` 对象

Kubernetes `StorageClass` 对象在 `PersistentVolumeClass` 中按名称指定，用于使用一组属性配置存储。存储类本身可标识要使用的配置程序，并按配置程序所了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 专用的，可告诉 Trident 如何为类配置卷。

存储类参数包括：

属性	Type	Required	Description
属性	map[string]string	否	请参见下面的属性部分
存储池	map[string]StringList	否	后端名称映射到中的存储池列表

属性	Type	Required	Description
附加 StoragePools	map[string]StringList	否	后端名称映射到中的存储池列表
排除 StoragePools	map[string]StringList	否	后端名称映射到中的存储池列表

存储属性及其可能值可以分类为存储池选择属性和 Kubernetes 属性。

存储池选择属性

这些参数决定了应使用哪些 Trident 管理的存储池来配置给定类型的卷。

属性	Type	值	优惠	请求	支持
介质[1]	string	HDD ，混合，SSD	Pool 包含此类型的介质；混合表示两者	指定的介质类型	ontap-nas ，ontap-nas-economy. ontap-nas-flexgroup ，ontap-san ，solidfire-san
配置类型	string	精简，厚	Pool 支持此配置方法	指定的配置方法	Thick：All ONTAP ；Thin ：All ONTAP & solidfire-san
后端类型	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	池	true false	Pool 支持具有快照的卷	启用了快照的卷	ontap-nas 、ontap-san 、solidfire-san
克隆	池	true false	Pool 支持克隆卷	启用了克隆的卷	ontap-nas 、ontap-san 、solidfire-san
加密	池	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	内部	正整数	Pool 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，请求的值直接影响配置；例如，请求厚配置会导致卷配置较厚。但是， Element 存储池会使用其提供的 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` 来为满足特定类需求所需的存储质量建模。Trident 会自动发现并选择与您指定的 **属性** 的 *all* 匹配的存储池。

如果您发现自己无法使用 `attributes` 自动为某个类选择合适的池，则可以使用 `storagePools` 和 `addtionalStoragePools` 参数进一步细化池，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的 `attributes` 匹配的池集。换言之， Trident 会使用 `attributes` 和 `storagePools` 参数标识的池的交叉点进行配置。您可以单独使用参数，也可以同时使用这两者。

您可以使用 `addtionalStoragePools` 参数扩展 Trident 用于配置的池集，而不管 `attributes` 和 `storagePools` 参数选择的任何池如何。

您可以使用 `excludeStoragePools` 参数筛选 Trident 用于配置的池集。使用此参数将删除任何匹配的池。

在 `storagePools` 和 `addtionalStoragePools` 参数中，每个条目的格式为 ` <backend>: <storagePoolList>` ，其中 ` <storagePoolList>` 是指定后端的存储池列表，以逗号分隔。例如， `addtionalStoragePools` 的值可能类似于 `ontapnas_192.168.1.100 ： aggr1 ， aggr2 ； solidfire_192.168.1.101 ： bronze` 。这些列表接受后端值和列表值的正则表达式值。您可以使用 `tridentctl get backend` 来获取后端及其池的列表。

Kubernetes 属性

这些属性不会影响 Trident 在动态配置期间选择的存储池 / 后端。相反，这些属性仅提供 Kubernetes 永久性卷支持的参数。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	**Type**	值	**Description**	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
允许卷扩展	boolean	true false	启用或禁用对增加 PVC 大小的支持	ontap-nas 、ontap-nas-economy、ontap-nas-flexgroup 、ontap-san 、ontap-san-economy、solidfire-san、azure-netapp-files	1.11 及更高版本

属性	Type	值	Description	相关驱动程序	Kubernetes 版本
卷绑定模式	string	即时，WaitForFirstConsumer"	选择何时进行卷绑定和动态配置	全部	1.19 - 1.26

提示

- 。 `fsType` 参数用于控制SAN LUN所需的文件系统类型。此外、Kubernetes还会使用 `fsType` 在存储类中以指示文件系统已存在。可以使用控制卷所有权 `fsGroup` 仅当出现此情况时、Pod的安全上下文才会显示 `fsType` 已设置。请参见 "Kubernetes ： 为 Pod 或容器配置安全上下文" 有关使用设置卷所有权的概述 `fsGroup` 环境。Kubernetes将应用 `fsGroup` 只有在以下情况下才为值：
  - 在存储类中设置 `FSType` 。
  - PVC 访问模式为 RW 。

  对于 NFS 存储驱动程序， NFS 导出中已存在文件系统。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。
- 请参见 "展开卷" 有关卷扩展的更多详细信息。
- Trident 安装程序包提供了几个示例存储类定义，可用于 `sample-input/storage-class-*` 。 `yaml` 中的 Trident 。删除 Kubernetes 存储类也会删除相应的 Trident 存储类。

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关联。每个卷快照都与一个卷快照类相关联。

要创建快照，管理员应定义 `VolumeSnapshotClass` 。此时将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` 指定给 Kubernetes ，由 Trident 处理对 `csI-snapclass` 类的卷快照请求。`deeltionPolicy` 指定必须删除快照时要执行的操作。如果将 `deletionPolicy` 设置为 `Delete` ，则在删除快照时，卷快照对象以及存储集群上的底层快照将被删除。或者，如果将其设置为 `Retain` ，则表示保留 `VolumeSnapshotContent` 和物理快照。

Kubernetes `VolumeSnapshot` 对象

Kubernetes `VolumeSnapshot` 对象是创建卷快照的请求。就像 PVC 代表用户对卷发出的请求一样，卷快照也是用户为现有 PVC 创建快照的请求。

收到卷快照请求后， Trident 会自动管理在后端为卷创建快照的操作，并通过创建唯一的 `VolumeSnapshotContent` 对象公开快照。您可以从现有 PVC 创建快照，并在创建新 PVC 时将这些快照用作 DataSource 。

备注	VolumeSnapshot 的生命周期与源 PVC 无关：即使源 PVC 被删除，快照仍然存在。删除具有关联快照的 PVC 时， Trident 会将此 PVC 的后备卷标记为 " 正在删除 " 状态，但不会将其完全删除。删除所有关联快照后，卷将被删除。

Kubernetes `VolumeSnapshotContent` 对象

Kubernetes `VolumeSnapshotContent` 对象表示从已配置的卷创建的快照。它类似于 `PersistentVolume` ，表示存储集群上配置的快照。与 `PersistentVolumeClaim` 和 `PersistentVolume` 对象类似，创建快照时， `VolumeSnapshotContent` 对象会与请求创建快照的 `VolumeSnapshot` 对象保持一对一映射。

`VolumeSnapshotContent` 对象包含用于唯一标识快照的详细信息，例如 `snapshotHandle` 。此 `snapshotHandle` 是 PV 名称和 `VolumeSnapshotContent` 对象名称的唯一组合。

收到快照请求后， Trident 会在后端创建快照。创建快照后， Trident 会配置一个 `VolumeSnapshotContent` 对象，从而将快照公开到 Kubernetes API 。

备注	通常、您不需要管理 `VolumeSnapshotContent`对象。但是、如果要在Trident外部创建、则会出现一个例外情况"导入卷快照"。

Kubbernetes `VolumeGroupSnapshotClass`对象

Kubbernetes `VolumeGroupSnapshotClass`对象类似于 `VolumeSnapshotClass`。它们有助于定义多种存储类别，并被卷组快照引用，以将快照与所需的快照类别关联。每个卷组快照都与单个卷组快照类别相关联。

一个 `VolumeGroupSnapshotClass`应由管理员定义，以便创建快照组。卷组快照类使用以下定义创建：

```
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
```

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` ，删除快照时，卷组快照对象以及存储集群上的底层快照也将被删除。或者、将其设置为 `Retain`表示将 `VolumeGroupSnapshotContent`保留和物理快照。

Kubbernetes `VolumeGroupSnapshot`对象

Kubernetes `VolumeGroupSnapshot`对象是创建多个卷快照的请求。正如 PVC 代表用户对卷的请求一样，卷组快照是用户为现有 PVC 创建快照的请求。

当卷组快照请求到达时，Trident 会自动管理后端卷的组快照的创建，并通过创建唯一的 `VolumeGroupSnapshotContent`目的。您可以从现有 PVC 创建快照，并在创建新 PVC 时将这些快照用作 DataSource 。

备注	VolumeGroupSnapshot 的生命周期与源 PVC 无关：即使源 PVC 被删除，快照仍然有效。删除具有关联快照的 PVC 时， Trident 会将此 PVC 的后备卷标记为 " 正在删除 " 状态，但不会将其完全删除。当所有关联的快照都被删除时，卷组快照也会被移除。

Kubbernetes `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。与和 `PersistentVolume`对象类似 `PersistentVolumeClaim`、创建快照时、 `VolumeSnapshotContent`对象会与请求创建快照的对象保持一对一映射 `VolumeSnapshot`。

这 `VolumeGroupSnapshotContent`对象包含识别快照组的详细信息，例如 `volumeGroupSnapshotHandle`以及存储系统上现有的各个volumeSnapshotHandles。

当快照请求到达时，Trident 会在后端创建卷组快照。创建卷组快照后，Trident 会配置一个 `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，由管理员定义并用于对类似对象进行分组。Kubernetes 支持创建自定义资源以存储对象集合。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。这样就无需为 Trident 创建单独的存储。

Trident使用 `CustomResourceDefinition`对象保留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一些定义卷快照所需的 CRD 。

CRD 是一种 Kubernetes 构造。上述资源的对象由 Trident 创建。例如，使用 `tridentctl` 创建后端时，会创建一个对应的 `tridentbackend` CRD 对象，供 Kubernetes 使用。

有关 Trident 的 CRD ，请注意以下几点：

- 安装 Trident 时，系统会创建一组 CRD ，并可像使用任何其他资源类型一样使用。
- 使用卸载Trident时 `tridentctl uninstall` 命令中、Trident Pod会被删除、但创建的CRD不会被清理。请参见 "卸载 Trident" 了解如何从头开始完全删除和重新配置 Trident 。

Trident `StorageClass`对象

Trident会为Kubernetes创建匹配的存储类 `StorageClass` 指定的对象 `csi.trident.netapp.io` 在其配置程序字段中。存储类名称与Kubernetes的名称匹配 `StorageClass` 它所代表的对象。

备注	使用 Kubernetes 时，如果注册了使用 Trident 作为配置程序的 Kubernetes `StorageClass` ，则会自动创建这些对象。

存储类包含一组卷要求。Trident 会将这些要求与每个存储池中的属性进行匹配；如果匹配，则该存储池是使用

该存储类配置卷的有效目标。

您可以使用 REST API 创建存储类配置以直接定义存储类。但是，对于 Kubernetes 部署，我们希望在注册新的 Kubernetes `StorageClass` 对象时创建这些部署。

Trident 后端对象

后端表示存储提供程序，其中 Trident 配置卷；单个 Trident 实例可以管理任意数量的后端。

备注	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。

有关如何构建这些对象的详细信息、请参见 "正在配置后端"。

Trident `StoragePool`对象

存储池代表每个后端可用于配置的不同位置。对于ONTAP而言，这些对应于 SVM 中的聚合。对于NetApp HCI/SolidFire，这些对应于管理员指定的 QoS 频段。每个存储池都有一组独特的存储属性，这些属性定义了其性能特征和数据保护特征。

与此处的其他对象不同，存储池候选对象始终会自动发现和管理。

Trident `Volume`对象

卷是基本配置单元、由后端端端点(例如NFS共享以及iSCSI和FC LUN)组成。在Kubnetes中，这些直接对应于 `PersistentVolumes`。创建卷时，请确保其具有存储类，此类可确定可配置该卷的位置以及大小。

备注	• 在 Kubernetes 中，这些对象会自动进行管理。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 * 正在删除 * 状态。要删除 Trident 卷，您应删除该卷的快照。

卷配置定义了配置的卷应具有的属性。

属性	Type	Required	Description
version	string	否	Trident API 版本（ "1" ）
name	string	是的。	要创建的卷的名称
存储类	string	是的。	配置卷时要使用的存储类
size	string	是的。	要配置的卷大小（以字节为单位）
协议	string	否	要使用的协议类型； "file" 或 "block"
内部名称	string	否	存储系统上的对象名称；由 Trident 生成
cloneSourceVolume	string	否	ONTAP （ NAS ， SAN ）和 SolidFire — * ： 要从中克隆的卷的名称

属性	Type	Required	Description
splitOnClone	string	否	ONTAP （ NAS ， SAN ）： 将克隆从其父级拆分
snapshotPolicy	string	否	Snapshot-*： 要使用的 ONTAP 策略
SnapshotReserve	string	否	Snapshot-*： 为快照预留的卷百分比 ONTAP
导出策略	string	否	ontap-nas*： 要使用的导出策略
snapshotDirectory	池	否	ontap-nas*： 是否显示快照目录
unixPermissions	string	否	ontap-nas*： 初始 UNIX 权限
块大小	string	否	SolidFire — *： 块 / 扇区大小
文件系统	string	否	文件系统类型
跳过恢复队列	string	否	删除卷时，绕过存储中的恢复队列，立即删除卷。

创建卷时， Trident 会生成 `internalName` 。这包括两个步骤。首先，它会将存储前缀（默认值 `trident` 或后端配置中的前缀）预先添加到卷名称中，从而使名称格式为 ` <prefix>-<volume-name>` 。然后，它将继续清理名称，替换后端不允许使用的字符。对于 ONTAP 后端，它会将连字符替换为下划线（因此，内部名称将变为 ` <prefix>_<volume-name>` ）。对于 Element 后端，它会将下划线替换为连字符。

您可以使用卷配置使用 REST API 直接配置卷，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，可用于配置新卷或还原状态。在 Kubernetes 中，这些对象直接对应于 `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` 对象包括以下属性：

属性	Type	Required	Description
version	string	是的。	Trident API 版本（ "1" ）
name	string	是的。	Trident Snapshot 对象的名称
内部名称	string	是的。	存储系统上 Trident Snapshot 对象的名称
volumeName	string	是的。	为其创建快照的永久性卷的名称
volumeInternalName	string	是的。	存储系统上关联的 Trident 卷对象的名称

备注 在 Kubernetes 中，这些对象会自动进行管理。您可以查看它们以查看 Trident 配置的内容。

创建 Kubernetes `VolumeSnapshot` 对象请求时， Trident 会在备用存储系统上创建 Snapshot 对象。此快照对象的 `internalName` 是通过将前缀 `snapshot-` 与 `VolumeSnapshot` 对象的 `UID` （例如， `snapshot-e8d8a0ca-9826-11e9-9807-525400f3f660` ）组合而生成的。`VolumeName` 和 `volumeInternalName` 可通过获取后备卷的详细信息来填充。

Trident `ResourceQuota`对象

Trident守护进程使用优先级类(KubeNet中可用的最高优先级类)、以确保Trident可以在正常节点关闭期间识别和清理卷、并允许Trident守护进程 `system-node-critical`Pod抢占资源压力较高的集群中优先级较低的工作负载。

为此、Trident会使用一个 `ResourceQuota`对象来确保满足Trident守护程序集上的"system-node critical"优先级类。在部署和创建守护进程之前、Trident会查找对象、如果未发现、则会应用该 `ResourceQuota`对象。

如果您需要对默认资源配额和优先级类进行更多控制、可以使用Helm图表生成`custom.yaml`或配置`ResourceQuota`对象。

以下是一个`ResourceQuota`对象的示例、该对象会优先处理Trident子集。

```
apiVersion: <version>
kind: ResourceQuota
metadata:
  name: trident-csi
  labels:
    app: node.csi.trident.netapp.io
spec:
  scopeSelector:
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values:
          - system-node-critical
```

有关资源配额的详细信息、请参见 "Kubernetes：资源配额"。

清理 `ResourceQuota` 如果安装失败

在极少数情况下、如果在创建`ResourceQuota`对象后安装失败、请先尝试 "正在卸载" 然后重新安装。

如果不起作用、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、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)和安全上下文限制(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。为了提供此自定义功能、Trident会在安装期间启用某些权限。以下各节详细介绍了Trident设置的权限。

备注	PSS将取代Pod安全策略(PSP)。PSP已在Kubernetes v1.21中弃用、并将在v1.25中删除。有关详细信息、请参见 "Kubernetes：安全性"。

所需的Kubernetes安全上下文和相关字段

权限	Description
特权	CSI要求挂载点为双向挂载点、这意味着Trident节点POD必须运行特权容器。有关详细信息，请参见 "Kubernetes：挂载传播"。
主机网络连接	iSCSI守护进程所需的。`iscsiadm`可管理iSCSI挂载、并使用主机网络与iSCSI守护进程进行通信。
主机IPC	NFS使用进程间通信(Interprocess Communication、IPC)与NFSD进行通信。
主机PID	启动NFS时需要此参数 `rpc-statd`。Trident会在挂载NFS卷之前查询主机进程以确定是否 `rpc-statd`正在运行。
功能	`SYS_ADMIN`功能是特权容器的默认功能的一部分。例如、`Docker`会为有权限的容器设置这些功能：`CapPrm：00003fffffffff``CapEff：00003fffffffff`
Seccomp	Seccomp配置文件始终处于"非受限"状态、因此无法在Trident中启用。
SELinux	在OpenShift上、有权限的容器在("超级特权容器")域中运行 `spc_t`、无权限的容器在域中运行 `container_t`。在上 `containerd`，安装后 `container-selinux`，所有容器都在域中运行 `spc_t`，从而有效地禁用SELinux。因此、Trident不会添加`seLinuxOptions`到容器中。
DAC	有权限的容器必须以root用户身份运行。非特权容器以root用户身份运行、以访问CSI所需的UNIX套接字。

POD安全标准(PSS)

Label	Description	Default
po-security.Kubernetes IO/enforce``po-security.Kubernetes IO/enforce版本	允许将Trident控制器和节点收入安装命名空间。请勿更改命名空间标签。	enforce：privileged``enice-version：<当前集群的版本或测试的最高PSS版本。>

警告 更改命名空间标签可能会导致Pod未计划、出现"创建时出错：..."或"警告：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	Description	Default
allowPrivilegeEscalation	有权限的容器必须允许权限升级。	true
允许CSIDrivers	Trident不使用实时CSI临时卷。	空
allowedCapabilities	非特权Trident容器所需的功能不会超过默认设置、而特权容器会获得所有可能的功能。	空
支持FlexVolumes	Trident不使用 "FlexVolume驱动程序"、因此它们不会包含在允许的卷列表中。	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好处。	空
allowedProcessMountTypes	Trident不使用任何`ProcMountTypes`。	空
允许UnsafeSysctls	Trident不需要任何不安全的`sysctls`。	空
defaultAddCapabilities	无需向有权限的容器添加任何功能。	空
defaultAllowPrivilegeEscalation	允许权限升级在每个Trident POD中进行处理。	false
forbiddenSysctls	不允许使用`sysctls`。	空
fsGroup	Trident容器以root身份运行。	RunAsAny
hostIPC	挂载NFS卷需要主机IPC才能与`nfsd`进行通信	true
hostNetwork	iscsiadm要求主机网络与iSCSI守护进程进行通信。	true
hostPID	需要使用主机PID来检查节点上是否正在运行`rpc-statd`。	true
hostPorts	Trident不使用任何主机端口。	空
特权	Trident节点Pod必须运行特权容器才能挂载卷。	true

字段	Description	Default
readOnlyRootFilesystem	Trident节点Pod必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点Pod运行有权限的容器、无法删除功能。	无
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
seLinux	Trident未设置`seLinuxOptions`、因为容器运行时间和Kubernetes分发程序处理SELinux的方式目前存在差异。	空
supplementalGroups	Trident容器以root身份运行。	RunAsAny
卷	Trident Pod需要这些卷插件。	hostPath、projected、emptyDir

安全上下文限制(SCC)

标签	Description	Default
allowHostDirVolumePlugin	Trident节点Pod挂载节点的根文件系统。	true
allowHostIPC	挂载NFS卷需要主机IPC才能与`nfsd`进行通信。	true
allowHostNetwork	iscsiadm要求主机网络与iSCSI守护进程进行通信。	true
allowHostPID	需要使用主机PID来检查节点上是否正在运行`rpc-statd`。	true
allowHostPorts	Trident不使用任何主机端口。	false
allowPrivilegeEscalation	有权限的容器必须允许权限升级。	true
allowPrivilegedContainer	Trident节点Pod必须运行特权容器才能挂载卷。	true
允许UnsafeSysctls	Trident不需要任何不安全的`sysctls`。	无
allowedCapabilities	非特权Trident容器所需的功能不会超过默认设置、而特权容器会获得所有可能的功能。	空
defaultAddCapabilities	无需向有权限的容器添加任何功能。	空
fsGroup	Trident容器以root身份运行。	RunAsAny
组	此SCC专用于Trident并绑定到其用户。	空

标签	Description	Default
readOnlyRootFilesystem	Trident节点Pod必须写入节点文件系统。	false
requiredDropCapabilities	Trident节点Pod运行有权限的容器、无法删除功能。	无
runAsUser	Trident容器以root身份运行。	RunAsAny
seLinuxContext	Trident未设置`seLinuxOptions`、因为容器运行时间和Kubernetes分发程序处理SELinux的方式目前存在差异。	空
seccompProfile	有权限的容器始终运行"无限制"。	空
supplementalGroups	Trident容器以root身份运行。	RunAsAny
用户	提供了一个条目、用于将此SCC绑定到Trident命名空间中的Trident用户。	不适用
卷	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、专利等的访问。

版权

"https://www.netapp.com/company/legal/copyright/"

商标

NetApp 、 NetApp 徽标和 NetApp 商标页面上列出的标记是 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

有关 NetApp 拥有的专利的最新列表，请访问：

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

隐私政策

"https://www.netapp.com/company/legal/privacy-policy/"

开放源代码

您可以在每个版本的notices文件中查看适用于Trident的NetApp软件中使用的第三方版权和许可证，网址为 https://github.com/NetApp/trident/。

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。 美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 http://www.netapp.com/TM 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。